>12.95

Designed for science teachers, students, and hobbyists, this book teaches the fundamental concepts of computer interfacing. Anyone with a basic understanding of electricity and of the BASIC programming language can quickly learn to measure and control experiments and instrumentation with microcomputers.

This comprehensive book offers a practical, hands-on approach to computer interfacing. By constructing simple, low-cost electronic interface circuits, beginners will gain an understanding of digital and analog electronics and of the Z80 microprocessor. Readers will also learn how to build input and output ports, convert analog signals to digital input, and generate the control signals used for advanced interfacing.

Computer Interfacing Techniques in Science includes 30 experiments to help readers apply the interfacing techniques discussed in each chapter. Designed for use with any Timex/Sinclair computer, these experiments help develop programming and circuit building skills while illustrating important concepts in the physical sciences. Each experiment includes step-by-step directions, a list of necessary components, and easy-to-follow schematics and diagrams. These experiments show how microcomputers can be readily adapted for practical uses in the school, university, home, or laboratory.

Paul E. Field is Professor of Physical Chemistry at Virginia Polytechnic Institute and State University in Blacksburg, Virginia. Dr. Field has been teaching courses and university extension workshop seminars on computer interfacing for college students and academic, government, and industrial scientists and engineers since 1977.

John A. Davies is a Lecturer in Physics at the Queensland Institute of Technology in Brisbane, Australia. He received his BSc Honors degree in Applied Physics from the City University of London, and has published numerous articles on electronics and microprocessors for scientific journals in Australia.

Scott, Foresman and Company

ISBN 0-673-18112-X

Field / Davies computer interfacing techniques in science

computer interfacing techniques in science

with experiments for Sinclair and Timex / Sinclair computers

Paul E. Field John A. Davies

computer interfacing techniques in science

computer interfacing techniques in science

Paul E. Field □ □ □ John A. Davies

Scott, Foresman and Company

Glenview, Illinois □ London

ISBN 0-673-18112-X

Library of Congress Cataloging in Publication Data

Field, Paul E.
Computer interfacing techniques in science.

Includes index.
1. Computer interfaces. I. Davies, John A.
II. Title.
TK7887.5.F49 1985 621.3819'5832 84-26714
ISBN 0-673-18112-X

1 2 3 4 5 6-KPF-90 89 88 87 86 85

contents

List of Figures

preface

Computer interfacing is the means for connecting a computer to sensors and actuators. It is the "bridge" that spans the gap between computer science and electronic technology. The principles of computer interfacing are relatively simple and can be learned by anyone who has the patience and is willing to invest the time and care to read and perform some simple step-by-step experiments. This is not to imply that there are not very sophisticated and elaborate activities that can be accomplished once the techniques are mastered. Rather, our point is that the technology has been so well developed that you do not have to be a specialist to apply it.

The plan of this book is to introduce you to the concepts of computer interfacing, assuming you have no prior experience in digital electronics. Although you may be able to learn some by just reading, we are convinced that mastery of the techniques will only come by doing "hands on" experiments in programming and circuit building. To this end we have written this book to be used with a computer. In particular we have chosen the Timex/Sinclair computers because they are very inexpensive yet very sophisticated microcomputers. These include the ZX81, TS1000, TS1500, Spectrum, and TS2068 models. If you are willing to make the effort to learn the subject of computer interfacing, we strongly recommend that you make the modest investment in a personal computer to go along with your study.

There is a saying that the only way to learn computer programming is to write computer programs. It is equally true that the only way to learn computer interfacing is to build interfaces. You do not have to be a computer scientist to become a programmer nor do you have to be an electrical engineer to become an interfacer. With the fantastic growth in personal computing over the past few years and the promise of even greater growth in the immediate years to come, the majority of users will be content simply to operate their computers with programs and devices developed by others. But there are those who will discover that there is an even greater adventure in creating and discovering ways to use personal computers that goes beyond "plugging and chugging." This book is for them. It is the outgrowth of the authors' experience in teaching this subject to high school and university students and teachers, industrial and government technicians, engineers, and scientists. It is the authors' hope that this book will be useful not only to individual students, scientists, engineers, and hobbyists, but also to teachers in high school and beyond so that they can introduce these techniques to their students.

In Chapter 1, we survey those aspects of scientific experimentation that pertain to computer automation for measurement and control. We also discuss some fundamental concepts dealing with numbers and codes that will be helpful in understanding the principles developed in the rest of the book. Chapters 2 through 7 cover the various aspects of computer interfacing in a logical sequence starting with digital electronics (Chapter 2) and ending with a description of the control signals used for advanced interfacing (Chapter 7). Chapters 2 through 6 consist of discussions of the principles of the topic under consideration and a series of six experiments which serve to illustrate

the important concepts discussed. A survey of the logical structure of the Z80 microprocessor and machine code programming is presented in Chapter 3. The principles of input and output ports presented in Chapter 4 is built on the material developed in the preceding two chapters. Chapter 5 and 6 apply the principles of input/output ports. Chapter 5 deals with digital input and output and analog output while in Chapter 6 we consider the requirements of signal conditioning for obtaining digital input from analog signals. As noted above, we conclude in Chapter 7 with some principles of the more advanced techniques used in automation.

The authors would like to express their thanks and appreciation to David G. Larsen for his interest and cooperation. Our thanks also to Roger J. Combs for his assistance. One of us (J.A.D.) would also like to acknowledge the Chemistry Department, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, for their provision of facilities during the development of this book and also to the Queensland Institute of Technology, Brisbane, Australia, for providing financial support for the project to be completed in Blacksburg. We would like to dedicate this book to our wives, Barbara D. and Jewell F., for their encouragement and support.

□ 1 □

bits and pieces

One purpose of computer interfacing is automation, for example, using a computer to assist in making measurements and controlling devices. The measurement can be as simple as determining if a device is on or off, and the control can be as simple as turning a device on or off. Of course, both the measurement and control can be as sophisticated as the imagination and resources of the interfacer allow. One thing that becomes obvious as you learn the techniques of interfacing is that there are many useful and clever things that can be done using very simple and inexpensive techniques. A device can be as simple as a light bulb or as complicated as a robot or an industrial plant.

The two essential elements of computers are software (programs encoded with bits) and hardware (the pieces of equipment). *Hardware* consists of the electronic circuitry and electromechanical devices that make up the computer. It includes the integrated circuits, keyboards, video displays, disks and tape recorders, printers, and all other such peripheral devices. *Software* is the collective name given to the programs which make the hardware elements function in a coordinated way to achieve the purposes of the user.

MEASUREMENT AND CONTROL

To the items in the list of peripheral devices we could also add scientific instruments. It is of considerable importance to any individual in today's high technology world, whether high school science or vocation student, or research scientist, to appreciate the potential for automation of scientific instrumentation and what is involved. There are two concepts that are useful to serve as our point of departure. We shall start with a dictionary definition of each taken from Webster's New Collegiate Dictionary (G. & C. Merriam Co., 1980) and elaborate from there. The first is the definition of *experiment.*

> An experiment is an operation carried out under controlled conditions in order to discover an unknown effect or law, to test or establish an hypothesis, or to illustrate a known law.

Most experiments involve the operation of measurement. What is emphasized in this definition is that those measurements must be made under controlled conditions. In practically every experiment, measurement of some property of interest is of value only if other experimental properties (parameters) are held constant. In many experiments, the property to be measured is of particular interest as some (one) other experimental property or parameter is systematically varied. The property of interest is the *dependent variable*, the property which is systematically varied is the *independent variable*, and those properties which are not varied are *experimental constants*. Usually the data obtained from the experiment are illustrated by a graph. The values of the dependent variable are plotted on the vertical axis versus the corresponding values of the independent variable on the horizontal axis. The results that are sought are often some property of this graph, such as its shape, slope (steepness), or intercept. During the course of the experiment, not only the dependent variable but also the independent variable and constant parameters must be measured to ensure that they are well behaved. The experiment must be designed to control all of the experimental conditions. In general, we can consider a scientific instrument as an automated experiment involving the measurement and control of the required experimental parameters.

The second definition we should consider is of the term *data processing*.

> Data processing: the converting of raw data to machine readable form and its subsequent processing (as storing, updating, combining, rearranging, or printing out) by a computer.

We can see from this definition that instrument automation is one aspect of data processing. Without taking exception to this definition, it is more convenient from our point of view to divide instrument automation into the two areas of (1) computer interfacing and (2) data processing. Here we consider computer interfacing as both the conversion of raw data into machine readable form for data acquisition, as well as the conversion of machine data for instrument control. Note that we reserve the "subsequent processing" given in the definition to be the main emphasis of the term data processing. The distinction between the conversion of raw data and its subsequent processing thus becomes predominately a distinction between hardware and software. Figure 1.1 illustrates the relationship of the terms we have discussed.

PROGRAMMING AND PERSONAL COMPUTERS

We assume that you have some familiarity with personal computers. You should at least be familiar with the more elementary BASIC programming commands, such as: PRINT, INPUT, LET, GOTO, FOR. . .NEXT, IF. . .THEN. If you have not had any experience in writing simple BASIC programs, you can learn all you need to know from the User Manual that comes with the computer. If you have had some experience, you know how the computer needs to be programmed in order to perform some desired task. In BASIC, the program consists of a numbered list of lines with a BASIC command written on each line. This program is stored in the computer's memory, and when RUN is ENTERed from the keyboard, the computer executes the

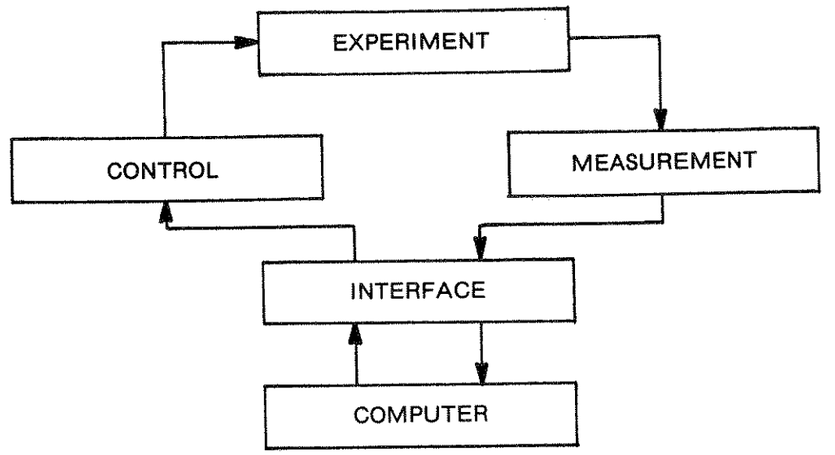


Figure 1.1 The Automated Instrument.

program by starting with the lowest line numbered command and executing commands in succession of increasing line numbers unless commanded to GOTO some other line number out of sequence.

The memory that your BASIC program is stored in is commonly called RAM (meaning *Random Access Memory*) but should properly be called Write And Read Memory (WARM). (WARM is an acronym suggested by the authors.) It is called *volatile memory* because it is lost when power is removed from the computer. When the computer loses power then the memory becomes COOL (Cleaned Out Of Logic)! Personal computers have varying amounts of memory usually measured in kilobytes, 1KB = 1024 memory locations. In addition to WARM, personal computers also have an additional 8 KB, 16 KB, or 24 KB of nonvolatile *Read Only Memory* (ROM), which holds the computer's operating system. The operating system is also a program. It is the program that handles all of the business of interacting with the keyboard and the video as well as executing your BASIC program when you enter RUN. The type of operating system used in the simpler personal computers is called a BASIC Interpreter. It is written in *machine language*, the set of instruction codes that the microprocessor in the computer can execute. Each seemingly elementary BASIC command consists of many instructions in machine language.

The advantage of using a so-called high level language such as BASIC is that its commands do considerably more sophisticated operations than the simpler single instructions of machine language. Therefore, it is much easier to write very complex programs using BASIC. The disadvantage of high level languages is that they are much slower (about 1000 times) than machine language routines. This is because the machine language routines are much more specific and do not have to handle the different contingencies that must be allowed for in the BASIC commands.

A personal computer is a microcomputer that includes a high level language operating system. It has only been in the last few years, as the prices of personal computers have so drastically decreased, that teaching courses in microcomputer interfacing could be done with personal computers. Until recently, schools could not afford enough personal computers to have a workable student/computer ratio for hands-on experience. Today, schools cannot afford not to have laboratories equipped to teach computer interfacing. Before this recent turn of events, interfacing was taught on small microcomputers having very limited memory and using only machine language.

With a personal computer such as the Timex/Sinclair, the interfacer has the best of both possible worlds. Simple machine language routines can be incorporated into BASIC programs with the USR command. They can be used for the very fast requirements of data acquisition, yet complicated mathematical and display operations can be performed on that data very easily using the high level language.

INTERFACING WITH TIMEX/SINCLAIR COMPUTERS

The experiments in this book have been designed to be performed with the Sinclair ZX81, Spectrum and Timex/Sinclair 1000, 1500, and 2000 computers. The Sinclair ZX81 was the first computer to sell for under $100 in America. When it was first introduced it had 1 KB of WARM and an 8 KB ROM operating system. The Timex/Sinclair 1000 is identical to the Sinclair ZX81 except that it has 2 KB of WARM. Timex subsequently marketed the TS1500 in North America with 16 KB of WARM, and virtually the same 8 KB ROM operating system. These three models all use black-and-white televisions for display. Throughout this book we shall refer to them as the B&W models. The Sinclair Spectrum and the Timex/Sinclair 2068 have color display features and considerably larger operating systems. The Spectrum has 16 KB of WARM while the Timex/Sinclair 2068 has 48 KB of WARM. The TS2068 has a 24 KB ROM operating system providing a total memory allocation of 72 KB for the basic machine. We shall refer to the Spectrum and TS2000 models as the Color models.

As we shall learn in Chapter 4, there are no hardware differences between the B&W and Color models that will affect our experiments. We have already seen that one of the major differences in the five models is the amount of WARM memory. Therefore, the only software difference we need to be concerned about is where to store the machine code routines we will use with the experiments. The routines themselves will be identical in operation, however, because there are instances when the program needs to refer to its own location in memory, there will be some differences in the values of the code numbers stored in the machine language programs. Fortunately, we can use one technique for all three B&W models and a second technique for the two Color models. We shall wait until Experiment 3.1 (Chapter 3) to give the details of the two techniques. Suffice it to say at this point that our interfacing experiments can work with any one of the five Timex/Sinclair computers.

In addition to the computer, the two additional pieces of equipment we will need to perform our interfacing experiments are a "breadboard" on which to build the circuits

and an interface buffer to connect a circuit to the computer. The *breadboard* is a unit that permits easy wiring connections between circuit components. The type we shall use is a 6.5 inch long by 2.25 inch wide polymer board having 64 rows of five solderless wire insertion sockets arranged on either side of a center channel running lengthwise on the board. Adjacent rows are on 0.1 inch centers, and the distance between opposite rows across the channel measures 0.3 inch. Standard Dual In-line Packaged (DIP) integrated circuits (IC) can straddle the channel leaving four common wire insertion sockets available for wiring connections made to each pin of the integrated circuit. (More detailed description for wiring the socket is given in Chapter 2.) The breadboard fits neatly either to the side or on the topside of the computer behind the keyboard.

Interfacing experiments require some means of bringing the computer's signal and power lines out from its printed circuit board. The physical connection to the computer's lines is made through 46 contacts arranged in rows of 23 contact pads on the top and bottom sides along the right rear edge of the computer's printed circuit (PC) board. We will use a 2×23 contact open-ended edge-connector socket having contacts on 0.10 inch centers, which can be inserted onto the computer PC board pads to bring the lines from the computer.

NEVER CONNECT THE EDGE CONNECTOR TO THE COMPUTER WITH POWER ON!!!

The edge connector should have a keyway inserted on the third pair of contacts from its left end to ensure alignment with a slot cut into the computer PC board. The edge connector is mounted onto a 3 inch wide by 3.5 inch high board to provide space for the integrated circuits of the buffer circuit and cable connectors. (The circuit of an Interface Buffer will be described in Chapter 4.) Two 14-conductor flat ribbon cables (each 6 to 9 inches long) plug into IC sockets located at the top of the interface board. The cables are terminated on wirewrap IC sockets for ease of insertion into the breadboard. The wirewrap socket pins also provide physical strength to the breadboard connection and ease of accessibility to the signals. These 28 lines bring out all the properly buffered signals necessary for input/output interfacing to the breadboard socket.

NUMBER SYSTEM PRELIMINARIES

Before we proceed to get into the details of digital processing and microcomputer fundamentals in the following chapters, it will be to our advantage to summarize some concepts of numbers. We all can understand what the value of a particular number means, and we all know how to represent that value as a line or string of decimal digits. For example, if you were asked to write the number "one hundred and twenty-three," you would write the digits 123. You would expect anyone else to understand that what you wrote means that there is one 100, two 10s, and three 1s. The decimal number system is as common as the fingers on your hands! The decimal system requires that

we use a set of ten quantity symbols called numerals: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; and that we represent each number value by listing from right to left the quantity of ones, tens, hundreds, thousands, etc. The right-most digit is the smallest or Least Significant Digit (LSD) and the left-most digit is the largest or Most Significant Digit (MSD). Each digit in the list is ten times bigger than its neighbor to the right. The decimal system is said to be base 10 number system.

When working with computers, there are times when it is necessary to use number systems other than base 10. In fact, there are four other bases that are handy. These are: base 2 having the numerals 0 and 1 only; base 8 with numerals 0, 1, 2, 3, 4, 5, 6, and 7; base 16 with numerals 0, . . ., 9, A, B, C, D, E, F, where we have to introduce six new number symbols whose decimal values are A(10), B(11), C(12), D(13), E(14), and F(15); and finally, base 256. No one is about to suggest a list of 256 unique symbols, so we are content to use up to three-digit decimal representation for the numerals in this base. Unlike base 10, the four new bases are all integer powers of 2: namely, 2, 2**3, 2**4, and 2**8 in increasing base order. The symbol ** is one way to denote a number raised to a power. It is used in Sinclair BASIC. For example, 2**3 means to multiply 2 by itself three times: 2 * 2 * 2 = 8. The * symbol is used in computer programming to indicate the multiplication sign.

The trick to dealing with the different number base systems is to apply the rules for representing a number value in a manner consistent with your experience in using the decimal system. A dozen will always be a dozen, that is, the number value you expect to find in a carton of eggs; but the representation of the value of a dozen will differ in each base: for these systems it becomes 1100 (base 2), 14 (base 8), C (base 16), 12 (base 256), and, naturally, 12 (base 10). The values, for example, of the digit positions in a three-digit number are:

BASE	MSD		LSD
10	100	10	1
2	4	2	1
8	64	8	1
16	256	16	1
256	65536	256	1.

We can also rewrite this table in a more systematic way as:

BASE	MSD		LSD
10	10**2	10**1	10**0
2	2**2	2**1	2**0
8	8**2	8**1	8**0
16	16**2	16**1	16**0
256	256**2	256**1	256**0,

because any number raised to a power of 0 is 1. We see the value of the digit in a particular position is the base raised to an exponent denoting that position.

All of this new numbering is a consequence of the fact that microcomputers actually deal with 8 and 16 base 2 (binary) digit numbers. Another name for a binary digit is *bit*, and the name for an eight-bit number is *byte*. If we consider an eight-bit number as a set of eight boxes and recall that each box may only hold a 1 (full) or 0 (empty), we find that we actually have the equivalent to a number base of 256. The largest possible number is when each bit is a 1, and because the values of each box (bit) progressively double from right to left, it will be the sum of all bit values or 255:

	MSB							LSB
BIT	D7	D6	D5	D4	D3	D2	D1	D0
NUMBER	1	1	1	1	1	1	1	1
VALUE	128 +	64 +	32 +	16 +	8 +	4 +	2 +	1 = 255.

If we divide the boxes into 2 four-bit numbers, we have converted to the hexadecimal number system (base 16):

1 1 1 1 1 1 1 1

and the number is represented as FF, still equal to the decimal number of 255 because $(15 \times 16) + (15 \times 1) = 255$.

Finally, if we sort into three-bit groups, we obtain:

1 1 1 1 1 1 1 1
3 7 7

where although we are short one bit on the left group, we still have converted the byte to the octal number system (base 8). The decimal value of the number is $(3 \times 64) + (7 \times 8) + (7 \times 1) = 255$.

Each of these three ways of representing numbers has some useful purpose when working with computers. We shall see the importance of binary numbers throughout Chapter 2 and again in Chapter 3 when we describe how a computer performs mathematical operations. Also in Chapter 3 we will use the octal number system to illustrate the logic of the programming instructions of machine language. When we discuss writing programs in BASIC having machine language routines in Chapter 3 and the interfacing experiments in Chapters 4, 5, and 6, base 256 numbers will prove useful. We will not have occasion to use the hexadecimal (usually abbreviated HEX) system, but should point out that it is the most common system in use in the literature because it saves space when printing lists of byte values. The HEX values use only two digits per byte, whereas decimal and octal use three digits, and, of course, binary uses eight digits.

NUMBERS AND CODES

Binary numbers are more than just numbers to a computer, they also serve as codes. A *code* is just one set of symbols used to represent another set of symbols. The simplest binary code is the Binary Coded Decimal (BCD) code. Here we use a set of bits to

represent the decimal numerals. Because there are ten numerals, we need four bits in the BCD code. In fact, BCD is identical to the hexadecimal number system except the BCD code does not use the highest six HEX codes (numerals A through F).

Another code uses seven bits to represent all the upper and lower case letters, numbers, symbols, and punctuation marks used in ordinary writing. This code is the American Standard Code for Information Interchange (ASCII; pronounced As'-key). Actually, because seven bits allows 128 different code symbols, there are 32 nonprinting codes included in ASCII, such as backspace and carriage return found on a typewriter. ASCII is listed in Appendix A, Chart A.6.

One of the most important codes is the set of eight-bit numbers that forms the instruction code for the microprocessor in the computer. This code is the machine language of the computer. It controls the specific sequence of actions that the microprocessor performs that we consider an operation or instruction.

The concept of coding is very important to computer operation. Of course, a code must always be used in its proper context. It would make no sense to use a byte to represent two (packed) four-bit BCD numbers, say 74 = 0 1 1 1 0 1 0 0, when the code expected is ASCII, where 0 1 1 1 0 1 0 0 = r. That would be like speaking English to a German (nein? or nine?).

□ 2 □

digital electronics

DIGITAL SIGNALS

Digital signals are the signals by which microprocessors communicate with each part of the microcomputer system: the memory, the keyboard, and the display. Digital signals are also used for the purpose of data transfer or control between instruments, between computers, and with the outside world.

Just how are digital signals different from our normal everyday perception of how electronic (or any other) devices communicate with each other, and why is it important to understand exactly the nature of digital signals?

In trying to answer such a question, first let us take a look at how information can be transmitted. Take, for example, a temperature sensor, a common type being the mercury in glass thermometer. As the temperature rises, the mercury expands nearly linearly so that the mercury column rises up a graduated scale indicating the temperature. The thermometer is an instrument which communicates information to us, the temperature of our surroundings, visually by the length of the mercury column. Now consider another common type of temperature transducer such as an electric thermometer, which could use a thermocouple or thermistor as the temperature sensor. With such a measuring instrument it is not possible for us to visualize directly the temperature of the transducer because we cannot see the electric currents which are flowing in the circuits connected to the transducer. Instead, use is made of electronics to transform the electric signals to drive the needle across the scale on a meter. Now again, as the temperature increases, the needle moves gradually across the scale of the meter indicating ever-increasing values as the temperature rises.

The preceding discussion has described the communication of information by means of analog signals. That is, the response of the instrument is a smooth gradual variation of the output indicator—the length of the mercury column or the movement of a meter needle—as the input variable, temperature, changes.

Digital signals are quite different from the above examples, and we can draw on another everyday analogy by considering the action of the temperature or oil pressure indicators in an automobile. These are often small red lights on the instrument panel and should the temperature of the engine coolant become too great or should the oil level become too low the warning lights will turn on. The situation is one of discrete

information where at one instant of time the temperature of the coolant is apparently quite normal and at the next instant it is not. The information from the temperature transducer has been conveyed to us in the following manner:

WARNING LIGHT	COOLANT TEMPERATURE
Not lit	Normal
Lit	Too hot

This instrument warning light provides us with an example of communication of information by means of digital signals: either your coolant temperature is normal or it is not. It is a good example of the on/off technique.

We can now consider how electric signals are transferred from one instrument to another by electronic circuits. In the case of digital electronic signals we use only two states to describe the type of information that is to be transmitted. We could use the presence or absence of a voltage level in the circuit or we could use the flow of current or no flow of current as the indicator of our two digital states. The case of current flow extends easily from our warning light example in which the light will be lit when current flows, and the light will not be lit when current ceases to flow.

For our future use we will take the presence or absence of a voltage level in a circuit to indicate one or the other of the digital states and also define the voltage levels needed to specify these two states. The voltage level corresponding to the digital state of the light being lit will be taken as +5 V. The voltage level corresponding to the digital state of the light not being lit will be taken as 0 V, that is, at earth or ground potential.

Why are digital signals used as the means of communication between certain types of electronic equipment such as microcomputers? The answer to this question is tied up with an understanding of how electronic circuits function. It is relatively easy for electronic circuits to distinguish, with 100% reliability, between the two digital states corresponding to voltage levels of +5 V and 0 V, but it is not easy for electronic circuits to distinguish with 100% certainty between two analog voltage levels such as 3.685 V and 3.681 V. Consequently, digital microcomputers can give this sort of insurance. In summary, digital signals will be represented by two voltage levels, namely +5 V and 0 V. The reader should be aware that other voltage levels can be defined, but, for our purposes, the above representation is sufficient.

INTEGRATED CIRCUITS

The electronic circuits that are constructed to work with digital signals take on various functions as will be described in the later sections. Because these functions involve many duplicated circuits, the construction of such circuits lends itself to integrated fabrication. It is not the purpose of this text to provide an in-depth coverage on how integrated circuits are fabricated. Suffice it to say that integrated circuits (IC) often involve the fabrication, on a single chip of silicon, of a large number of similar circuits such as the common-emitter transistor amplifier and other electronic circuits and include coupling capacitors, bias resistors, and other components necessary to have

the circuit perform properly. These chips are packaged in Dual In-Line plastic (or ceramic) packages (DIL or DIP) with typically from 14 to 40 pins in two parallel rows. There are a few DIPs that have only eight pins but these are usually integrated analog circuits referred to as *linear ICs*.

Microprocessor chips often have tens of thousands of transistors on a single chip making up a very complex circuit. Even so, were you able to probe about inside the circuit while it was operating, the only two voltages you would observe would be +5 V and 0 V. In actual practice, the ideal voltages of +5 and 0 V for the two digital states are approximations only with one digital state corresponding to voltage levels above about 3.5 V and the other digital state corresponding to voltage levels below 1.5 V.

Some other terms are often used to describe the particular digital integrated circuits discussed in the preceding paragraphs. One term in particular is TTL, which stands for *Transistor-Transistor Logic* which reflects the fact that many of the digital integrated circuits use a pair of transistors as the basic active component. These circuits are standardized with series numbers denoted by the 7400 series of digital ICs. They are fast in operation with propagation times on the order of 10 nanoseconds (ten billionths of a second between input and output) for the simpler elements and use comparatively large amounts of current having a rating of 16 mA per output and 1.6 mA per input. Another series in the TTL family is the Low-power Shottky (LS) series denoted by the 74LS00 series of numbers. These ICs are about as fast as the regular 7400 series but have one-half the current drive for outputs and one-fourth the current requirements for inputs as the regular 7400 series.

The experiments described in this book will use almost exclusively LS integrated circuits because of the lower amount of power consumed and hence ensure that most experiments can be carried out using only the Timex/Sinclair power supply: one of the prime objectives for a low-cost interface unit.

Other classes of digital integrated circuits include CMOS (*complementary metal oxide semiconductor*), RTL (*resistor-transistor logic*), and ECL (*emitter-coupled logic*). Apart from CMOS, which also draw small amounts of current, we will not have occasion or need to use these other classes. It should be mentioned that CMOS are capable of operating at higher voltages than TTL.

There are several hundred specific integrated circuits in the TTL family of digital chips having Series Numbers 74XXX in the regular family and 74LSXXX in the Low-power Schottky family where the XXX represents a two- or three-digit number. It makes digital circuit design much easier when you realize that most of these chips fall into one of about 12 classes. Each class performs one type of function. The following list gives the functional names of the most common circuit classes:

1 Buffers/inverters
2 Gates
3 Flip-flops/latches
4 Shift registers
5 Decoders/demultiplexers
6 Data selectors/multiplexers
7 Encoders

8 Counters
9 Monostables
10 Digital comparators
11 Arithmetic logic units
12 Memory registers

Within each class, there are many specific ICs, each having a unique series number and performing its function in a particular way. We shall see that each series number circuit meets three sets of specifications. The first is the so-called *pin-out diagram*, which is a schematic figure of the integrated circuit and shows the assignment of each pin to a specific function as input, output, or power. The second is a *truth table*, which gives the state of each output connection for all permutations of logic states at the input connections. (Remember that there are only two possible logic states for inputs and outputs.) The third set of specifications is *timing diagrams*. These provide information on the time relations between changes in the logic states of input and output connections for the integrated circuit. These specifications are available from the chip manufacturer's literature, such as the *Texas Instruments TTL Data Book for Design Engineers*.

We shall devote the rest of this chapter to a discussion of several of the classes of TTL LS circuits, which we will need to use and understand.

BUFFERS AND INVERTERS

A *buffer* means exactly what it says: a go-between, a softener of heavy blows, etc. In microcomputing, buffers are particular types of integrated circuits used to isolate your experimental circuits from the intricate electronic operations of the microcomputer you are connected to (interfaced). In this way mistakes you make in your circuit connections are unlikely to damage ("burn up" is the expression often used) your precious microcomputer. So if the interface has been designed correctly using buffer integrated circuits then you can experiment quite happily in the knowledge that you can't normally harm your micro, even if you do apply power the wrong way around to some of your integrated circuits, which we have all done at one time or another.

The buffer integrated circuit is essentially made up of a transistor amplifier. The transistor is a three-terminal device, which has an input lead called the *base* and output and power supply leads connected to the other terminals called the *collector* and *emitter*. The transistor action ensures electrical isolation of output signals from input signals and is also capable of amplifying an input signal in current strength so that a number of other transistor-type integrated circuits can be attached to it electrically without making unfair power demands on the original signal.

A second type of buffer to consider is the NOT buffer or more simply the *Inverter*. Like the noninverting buffer, the inverter only has one input and one output and a very brief truth table. It is shown in Figure 2.1 together with the truth table. Note the inversion circle on the output lead. This inversion circle will appear on many digital circuit diagrams. The triangular symbol is that for an electronic amplifier or

INPUT	OUTPUT
A	Q
0	1
1	0

Figure 2.1 Inverter Diagram and Truth Table.

noninverting buffer; the inversion circle on the output indicates that when the amplifier has a logic 1 output the inverter itself will have a logic 0 output and vice versa. The inverter is fabricated in a Hex (six independent) Inverter integrated circuit type 74LS04.

In summary, buffers can then act as current amplifiers and also as voltage inverters (phase inverters), that is a digital voltage of +5-V input to a particular type of buffer will result in a 0-V output from the buffer, that is the opposite digital state. Such a buffer is called an inverter.

GATES

A knowledge of digital gates is essential to being able to understand clearly how interfacing and experiments can be accomplished using a microcomputer. We will spend a little time reviewing what a gate is and outline numerous examples of the use of gates in the remainder of the book. You have already seen that digital signals are of two values only, +5 V and 0 V. These two voltage levels can be expressed as two logic states 1 and 0 where we will choose what is known as the *positive logic equivalence*, that is:

+5 V is equivalent to the logic state 1;
and, 0 V is equivalent to the logic state 0.

Gates are electronic circuits which allow certain combinations of the logic states input to the gate to produce particular output logic states.

As an example, let us consider the logic AND gate. This gate will have two inputs, which could be labeled A and B, and an output labeled Q. The electronic symbol is shown in Figure 2.2.

Figure 2.2 Diagram of an AND Gate.

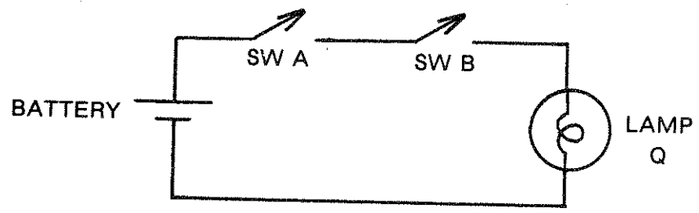


Figure 2.3 Switches in Series.

When different combinations of digital signals are applied to the inputs A and B, the output Q takes on only certain resultant digital states. To visualize this relationship, consider the simple circuit of switches in series shown in Figure 2.3, which represents an electrical analogy of the AND gate.

We will consider that an open switch is equivalent to the logic 0 state (no current flowing) and vice versa. For the resultant output of this circuit we have a small lamp Q, which can either be lit or not lit depending on whether voltage is applied to it or not (voltage will be applied to the lamp when current flows).

It should be clear that if switch A is open (0 state) and switch B is closed (the opposite 1 state) then no voltage will be applied to Q so it will remain unlit (logic 0 state). And again if switch A is closed (logic 1 state) and switch B is open (logic 0 state) then Q will still remain unlit. Only if switch A is closed AND switch B is closed will the combination result in lamp Q being lit. This can most easily be described using a truth table which tabulates the relationships between the inputs A and B and the output of the gate Q. The truth table is:

AND GATE INPUTS		OUTPUT	LOGIC STATEMENT
A	B	Q	A × B = Q
0	0	0	
1	0	0	
0	1	0	
1	1	1 = Unique State	

The last line gives the truth of the AND statement, that is, the output state is true (logic 1 state) only if both the logic states of A AND B are true. The AND gate is fabricated in a quad (four independent) Two-Input And Gate integrated circuit type 74LS08, where you could use a voltmeter or logic probe to test the truth table of each gate.

Another example is the logic OR gate, shown in Figure 2.4. The action of this circuit is explained in the electrical circuit shown in Figure 2.5.

In this circuit the two switches are in parallel so that if either switch A OR switch B is

Figure 2.4 Diagram of an OR Gate.

in the logic 1 state (closed), then the lamp Q also will be in the logic 1 state (lit). Expressed in a truth table the relationships between the logic states would be:

OR GATE INPUTS		OUTPUT	LOGIC STATEMENT
A	B	Q	A + B = Q
0	0	0 = Unique State	
1	0	1	
0	1	1	
1	1	1	

The logic relations of the OR gate are quite different from those of the AND gate. Use will be made of both functions later. The OR gate is fabricated in a Quad Two-Input OR Gate integrated circuit type 74LS32.

The inverter can be combined with both the AND and OR gates to produce NAND and NOR functions. The electronic symbols for these gates are shown in Figure 2.6*a* (NAND) and Figure 2.6*b* (NOR) together with their truth tables.

Note that for corresponding inputs, the output of the NAND gate is opposite (inverted) to that of the AND gate, and the output of the NOR gate is inverted to that of the OR gate. Inversion is also referred to as *negation*. The Quad Two-Input NAND gate is fabricated in an integrated circuit type 74LS00 and the Quad Two-Input NOR gate is fabricated in an integrated circuit type 74LS02.

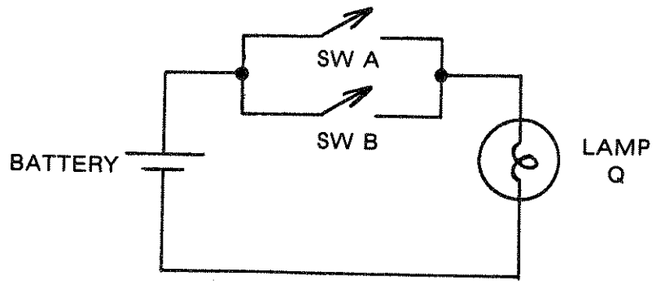


Figure 2.5 Switches in Parallel.

Another useful gate is the EXCLUSIVE OR gate whose circuit symbol and truth table appear in Figure 2.7.

Examples of the applications of the above gates will be given in the next section. It should also be noted that the number of inputs to any gate is not restricted to two only, you can have three-input, four-input, and eight-input gates, but the unique state of each truth table remains the same.

GATING AND DECODING

When microcomputers are used to control or interact with circuits to which they are interfaced it becomes necessary to gate particular control signals together to activate other integrated circuits at specific moments throughout the running of the microcomputer program.

By *gating* we mean exactly the same as opening and closing a fence gate to allow or prevent the passage of a person. In the electronic gate the action will be to prevent data from passing from an input to an output or to allow the passage of such data. In other words our gate will act as a control over the passage of data or "Enable" the passage of data as indicated in Figure 2.8. Because the inputs to a gate are equivalent, the input

NAND (NOT AND) Gate

INPUTS		OUTPUT	
A	B	Q	
0	0	1	
0	1	1	
1	0	1	
1	1	0	= Unique State

Figure 2.6a NAND (NOT AND) Gate and NOR (NOT OR) Gate.

NOR (NOT OR) Gate

INPUTS		OUTPUT	
A	B	Q	
0	0	1	= Unique State
0	1	0	
1	0	0	
1	1	0	

Figure 2.6b

that is called the data and that which is called the control is arbitrary and depends only on the user's point of view.

Often two different control signals will be logically combined by a gate and a unique output signal will only exist when the control signal applied to the one input enables the control signal applied to the other input. As has already been discussed, the microprocessor uses control signals to access memory locations, read the keyboard, output data to the video screen, and so on. Because the microcomputer can only handle one task at a time, the same control signals should not be allowed to activate two devices at once, hence it is necessary to gate certain control signals together to provide unique combinations of signals for use by the various circuits connected all the time to the microprocessor.

To give you an idea of how this works imagine that you have 16 different machines in your home which are to be controlled by the microcomputer. The microcomputer will decide when each machine should be turned on or off and how long each machine should stay on or off. Obviously we would not want the home heater running at the same time that the air conditioner is operating, so the microcomputer must be provided with a unique identification code (address) for each device.

This can be accomplished through a channel selector integrated circuit such as the four-channel to sixteen-channel 74LS154 decoder/demultiplexer shown in Figure 2.9.

You will note that there are four address inputs A, B, C, and D and 16 output channels. There are two gating control inputs, G1 and G2. Whenever both are held low, logic 0 state, the IC will be enabled. But we have 16 output channels so how can the microprocessor tell the decoder chip which channel should be selected?

INPUTS		OUTPUT	LOGIC STATEMENT
A	B	Q	A<+>B = Q
0	0	0	When both inputs are the same, then the output is a logic 0. No single unique state.
1	0	1	
0	1	1	
1	1	0	

Figure 2.7 EXCLUSIVE OR Gate.

This is accomplished through the four address inputs A, B, C, and D to which different binary codes can be applied. By starting with all address inputs LOW (0 V), channel 0 will be selected as shown below.

INPUTS				CHANNEL													
D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
:	:	:		:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

As the binary code applied to the address inputs is sequenced through 0000 to 1111, each channel in turn will be selected, and the output at that channel will go to the active logic 0 state. All other channels will be in their inactive logic 1 state. If either or both of the gate control inputs are in the logic 1 state then all 16 output channels will be in their inactive logic 1 states.

The 74LS154 functions as a decoder by substituting the set of four input signals for a set of 16 output signals (i.e., one set of symbols for another set of symbols). Of course, the new code is just a combination of 1 zero and 15 ones but it serves to select and activate one of 16 lines. If we wanted to transmit a sequence of data bits to any one of the 16 output channels we could feed that data to one of the gate inputs, G1 or G2. Then, assuming G1 was the data input line and G2 was enabled, when the address inputs held the channel number, every logic 0 data bit and logic 1 data bit would be output at the selected channel in sequence. This use of directing a stream of data to one of many channels is called *demultiplexing*.

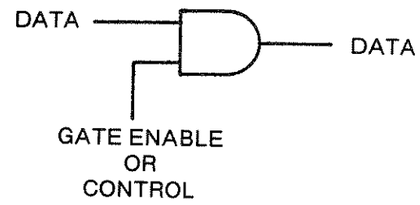


Figure 2.8 Gating Input for Control.

LATCHES AND REGISTERS

A *latch integrated circuit* "latches on" to a data signal and holds the logic state of the data at its output. A latch requires an enable control input, in addition to its data input, to signal the chip when to latch the data. This control input is either a clock or a gate input depending on how the specific latch circuit operates. The latch IC is important because it extends the lifetime of a data signal, which itself may be of very short duration. The output of a latch retains the state of a previous data input until a subsequent control pulse input causes it to reread the present logic state of the data input line. In this respect, a latch is a one-bit memory. Such latches are called *D-type*

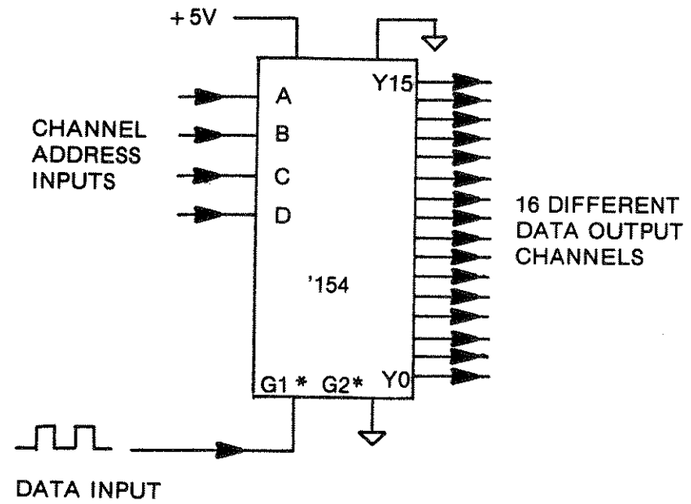
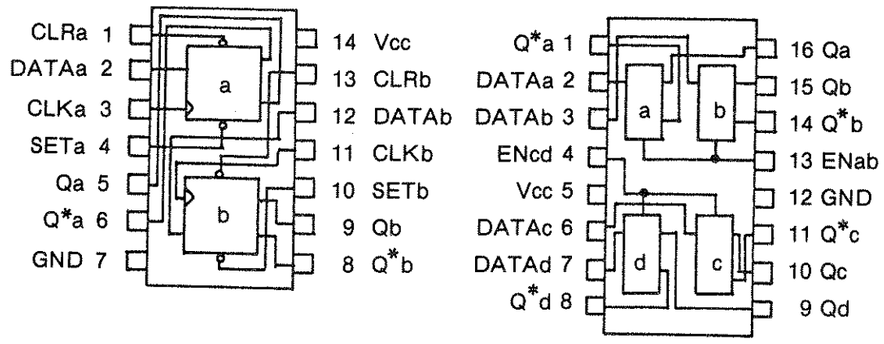


Figure 2.9 74S154 Schematic.

latches where the D originally referred to delay but more currently is used to mean data. D-type latches often are designed with two outputs, which are complementary and are labeled Q and Q*. Q* is always the inverse (negation or complement) of Q; that is, when Q is 0, Q* is 1 and vice versa. Many D-type latches also are designed with two additional control inputs labeled Preset and Clear. Logic 0 signals on either of these inputs take priority over the enable input and force the Q output to a logic 1 and logic 0 respectively.



----------INPUTS----------				OUTPUTS	
SET	CLR	CLK	DATA	Q	Q*
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	1?	1?
1	1	_/‾	1	1	0
1	1	_/‾	0	0	1
1	1	0	X	Qo	Qo*

---- INPUTS -------		OUTPUTS	
EN	DATA	Q	Q*
0	X	Qo	Q*o
1	0	0	1
1	1	1	0

Figure 2.10 Data Latch Pin-outs and Truth Tables.

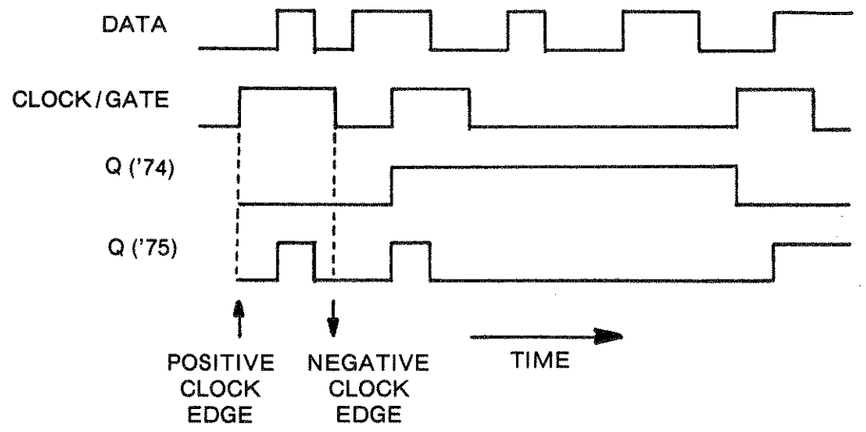


Figure 2.11 Data Latch Timing Diagrams.

The 74LS74 and 74LS75 D-type latches are representative of this type of circuit. Figure 2.10 gives the pin-outs and truth tables of both. The obvious differences between the two ICs are that the 74LS74 is a Dual D-type Latch having Preset and Clear and complementary outputs, whereas the 74LS75 is a Quad D-type Latch having only complementary outputs. More significantly, these ICs illustrate the difference between Clock and Gate data input controls. The 74LS74's Clock latch control is positive-edge triggered. This means that as the clock signal changes from a logic 0 to a logic 1, the data at the D input is latched at the Q output. The 74LS75's Gate enable, on the other hand, is active in the logic 1 state. This means that if D changes when the G input is high (logic 1), then Q will "follow" and change also. Q is not latched until the negative edge of G (i.e. when G changes from a logic 1 to a logic 0 and the latch is disabled). Figure 2.11 illustrates the difference in the two latches by showing how the Q of each responds to a busy data line. Typically, latches whose control input is a Clock input are also referred to as *flip-flops*.

There are several other types of flip-flops and latches in the TTL series which we will not consider because of their limited usefulness in computer interfacing. To mention two by name, there are Reset-Set or R-S flip-flops and also J-K flip-flops, neither of which has a Data input, but both are used in combining various control signals to create their output signal. There are also many other series numbers which incorporate latches that behave like those in the 74LS74 and 74LS75 ICs. When several latches are controlled by a common (single) enable signal, they are referred to as *registers*. We shall use the 74LS373 Octal D-type Latch as an eight-bit register in Chapter 4.

The difference between gating logic and clocked logic is an important distinction. Many of the other classes of integrated circuits use one or the other of these for their enabling technique. Generally when the enabling control input is labeled G (or EN), the circuit is enabled as long as the control input is in its active logic state; when it is

labeled CK, the circuit carries out its operation on the rising (positive) or falling (negative) edge of the signal transition. Truth tables for clocked devices will have entries for positive or negative edge-triggered control inputs represented by up-arrows or step-ups or by down-arrows or step-downs (reading from left to right) respectively.

COUNTERS

A *counter integrated circuit* counts pulses arriving at its input. Because this is a digital counter, it would require four output lines to store a count up to 15 (decimal). The outputs are typically labeled A, B, C, and D, where A is the least significant bit and D is the most significant bit. Thus when an output is a logic 1, A is worth 1, B is worth 2, C is worth 4, and D is worth 8. Obviously, all four bits must be read to know the value of the count. The major elements in the circuitry of a counter are flip-flops. The most common 74XX and 74LSXX counters are the '90, '92, and '93 which count to base 10, 12, and 16 respectively. Actually, each of these ICs consists of two counters; one is a binary or "divide-by-two" counter, while the other is a 5, 6, and 8 counter respectively. Because there are two counters on each IC, there are two inputs. The A output is the output of the binary counter; when it is connected to the input of the second counter in the circuit (labeled Input B), then all four outputs D–A hold the four-bit count. The 74LS90 is a decimal counter, which means it counts from 0000 to 1001 (9 in decimal). Like the odometer in an automobile, the next count after 9 is 0. Figure 2.12 illustrates the timing diagram of the 74LS90's outputs when the A input receives a train of clock pulses and output A is connected to the B input. If we read the logic states of the D–A outputs vertically, we find the four-bit binary-coded decimal (BCD) code. There are two important features you should observe about this timing diagram. First, count the number of pulses at the A input and the number of pulses at the A output. You should get ten and five respectively. Thus the output has divided the input by two: this is why the first counter is called a "divide-by-two." Now compare the number of pulses at input B with the number of pulses at output D: the second counter in the 74LS90 is a divide-by-five. Of course, with the two internal counters connected in series (cascaded), the counter is a divide-by-ten.

The second observation we can make about the timing diagram in Figure 2.12 is that the inputs are Clock inputs, which are active on the negative (falling) edge. It is not until the input pulse falls from logic 1 to logic 0 that the counter actually advances the logic states of the outputs. If we have two 74LS90 ICs, we can cascade them by connecting the D output of the first to the A input of the second. Now, everytime the first "rolls-over" and the 9 returns to 0, the second will increment its count. In this manner we could count from 0 to 99. Each additional '90 added by cascading gives

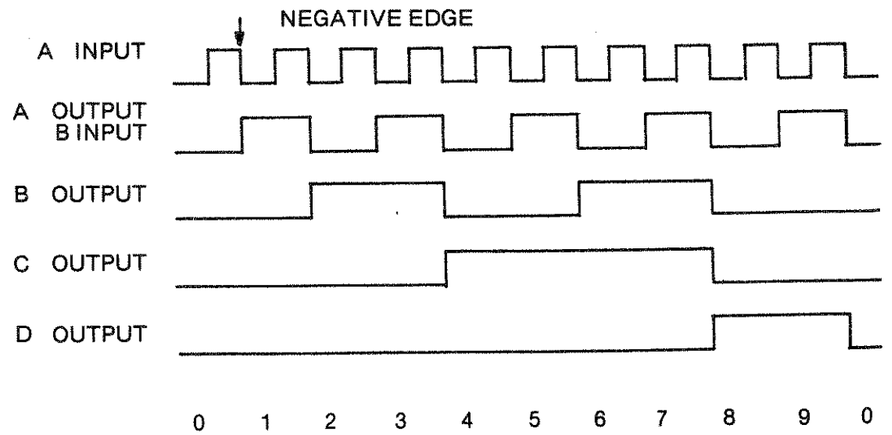


Figure 2.12 Decimal Counter Diagram.

another decimal digit. This is possible because it is the falling edge of the D output which "clocks" the count of the next counter.

Unlike the 74LS90 whose D output returns to 0 after the count of 9, the 74LS93 is a full four-bit counter. Its D output remains in the logic 0 state for counts 0 through 7 and then goes into the logic 1 state for counts 8 through 15. The pin configurations of the '90 and '93 are shown in Figure 2.13.

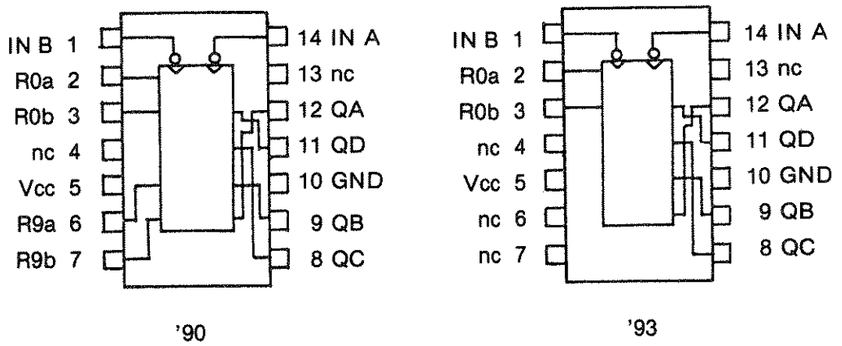


Figure 2.13 Pin-outs of '90 and '93 Counters.

The three counter ICs described have additional control inputs, R0(1) and R0(2), plus an additional pair on the '90, R9(1) and R9(2). Various logic states placed on these control lines will reset the counter outputs to 0 or 9. The truth tables for these controls are:

	R0(1)	R0(2)	R9(1)	R9(2)	A	B	C	D
'90:	1	1	0	X	0	0	0	0
	1	1	X	0	0	0	0	0
	X	X	1	1	1	0	0	1
'92, '93:	1	1	not applicable		0	0	0	0

where X is a "don't care" or irrelevant state. For all three ICs, for any other combination of logic states at the reset inputs, the input pulses will be counted.

TIMERS

Like counting, one of the more widespread applications of digital electronics is timing. A *clock pulse* is defined as a transition from one logic state to the other and back to the first. A positive clock pulse is 0 V to +5 V to 0 V. A negative clock pulse is +5 V to 0 V to +5 V. Clock pulses can be generated electronically with Monostable ICs or manually with a switch. The IC is called *monostable* because its output is stable in only one state. When it is triggered it goes into its unstable logic state, stays there for a brief (determined) time, t, and then falls back into its stable logic state. In the process, of course, it has generated a single clock pulse. There are several TTL Monostable ICs such as the 74121, 74LS122, and 74LS123. The duration of the clock pulse when the IC is triggered (by another clock pulse at a control input pin) is determined by the values of an external resistor and capacitor connected to two pins on the IC. For example, the 74121 can be adjusted to put out a pulse ranging between 40 nanoseconds and 28 seconds. The duration, t in seconds, of the 'LS122 and 'LS123 clock pulses can be calculated from the equation:

$$t = 0.45 * R * C$$

where R is in ohms and C is in farads (C must be greater than 1000 pF for the equation to be valid). The pin configuration of the 74121 is shown in Figure 2.14a.

There is another IC which, although not a TTL chip, can be operated between +5 V and Ground and is, therefore, TTL compatible. This IC is the eight-pin 555 Timer or the 14-pin 556 Dual 555-type Timer shown in Figure 2.14*b*. The 555 can be wired to work either as a monostable or as an astable multivibrator. *Astable* means that neither logic state of the output is stable. Therefore when it jumps into a logic state of 1 it stays for a period, $t(1)$, then jumps into a logic state of 0, but it is not stable here either, so after another period, $t(0)$, it jumps back to a logic state of 1 and repeats the process all over again. It therefore continues to vibrate back and forth between logic states. As it

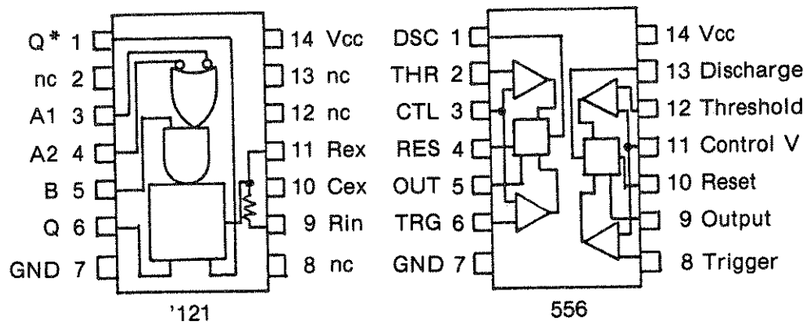


Figure 2.14 74121 and 556 Pin-outs.

does so, it creates a train of clock pulses. Thus, as we have seen, the astable multivibrator is a square wave generator.

When the 555 is wired as a monostable, it requires a resistor, R(a), and a capacitor, C. The duration in seconds of the logic 1 state, t, is calculated by the formula:

$$t = ln\ 3 * R(a) * C$$
$$= 1.1 * R(a) * C \text{ seconds}$$

when R is in ohms and C is in farads. Alternately, if R is in kiloohms and C is in microfarads, then t is measured in milliseconds.

When the 555 is wired as an astable multivibrator, it requires two resistors, R(a) and R(b), and one capacitor, C. The duration of the logic 1 state, $t(1)$, is calculated from:

$$t(1) = ln\ 2 * (R(a) + R(b)) * C$$
$$\text{Milliseconds} = 0.7 * \text{kiloohms} * \text{microfarads}$$

and the duration of the logic 0 state, $t(0)$, is:

$$t(0) = ln\ 2 * R(b) * C$$

with the same units as previously. The total period of one complete clock pulse is:

$$T = t(0) + t(1),$$

and the frequency (f) of the square wave in hertz (cycles per second) is:

$$f = 1/T.$$

If T is in milliseconds, then f is in kilohertz. We shall have use for both of these circuits in our counting experiments.

THREE-STATE BUFFERS

Although an output signal of an IC can be connected to several inputs, two or more outputs *cannot* be connected to one input. If two outputs were connected to one input and one was in a logic 1 state and the other in a logic 0 state, the +5 V of the logic 1 connected to the 0 V of the logic 0 would be a short-circuit and probably burn out one of the integrated circuits. We shall see that the microcomputer needs to have many IC output signals share a common signal lead (called a *bus line*) into the microprocessor. To solve this problem of connecting outputs together, integrated circuits have been specially designed that function like switches. These ICs are called *three-state buffers*. The three states correspond to the usual logic 1 and logic 0 states when the buffer is enabled (when the switch is closed) and to a high impedance state when the switch is open.

You will have little difficulty understanding the concept of three states once you realize that the logic 0 state (0 V) is a real condition and a legitimate signal which is different from a "no signal" or disconnected state. A three-state device permits the output signal of an IC to be isolated from the output connection of the device in the same manner as if the output lead were physically disconnected. Of course, the output lead is not physically disconnected but electronically disconnected by means of an extra control signal applied to the three-state device. Whereas the ordinary buffer has one input and one output, the three-state buffer has an additional enabling control input line that "throws the switch." The logic state of the enable gate may be either 1 or 0 depending on the specific series number of the IC.

The 74LS125 and 74LS126 are Quad Bus Buffer Gates. The '125 is enabled by a logic 0 and the '126 is enabled by a logic 1. The pin-outs of the two ICs are shown in Figure 2.15. Many ICs of other classes are available with built-in three-state buffers. One example is the 74LS373 Octal Latch mentioned earlier whose eight outputs are three-state and enabled by a single control input.

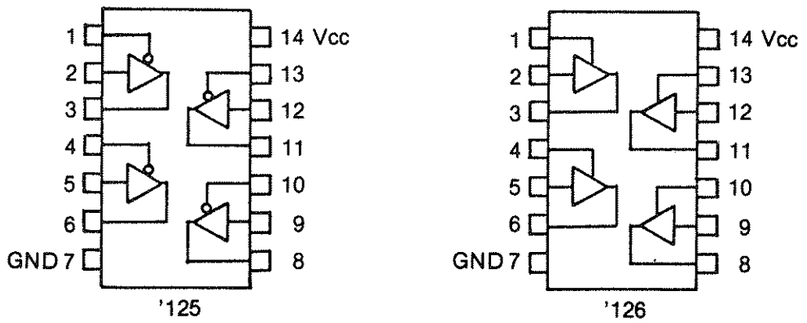


Figure 2.15 Three-State Buffer Pin-outs.

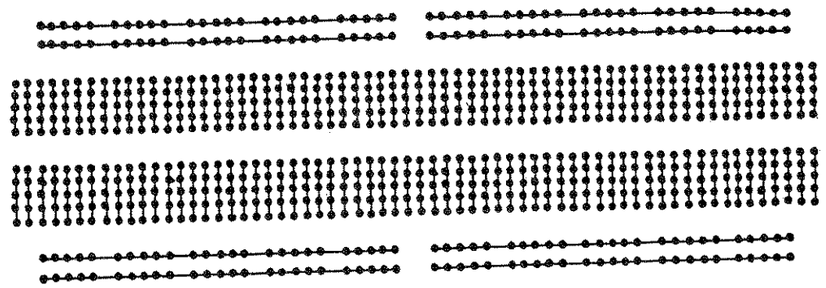


Figure 2.16 Layout of Breadboard.

HARDWARE AND TOOLS

As we already mentioned in Chapter 1, the interfacing experiments require a breadboard socket for wiring the circuits. In addition to the 64 rows of five common tie points on either side of the center channel, there are four rows in two sections each of 25 common tie points that run lengthwise along the outer edge of the breadboard. These rows are used as power rails. Always wire up the socket board so that the outside horizontal rails both at the top and bottom of the board are wired up for +5 V and the inside horizontal rails for 0 V as shown below. The connections of the tie points are shown in Figure 2.16. Note that jumpers on the power rails must be made at the center of the board and at one of the ends.

By using standard DIP integrated circuits, the solderless breadboarding socket allows rapid wiring of the experiments. Insertion of the chips is not difficult as long as all the pins are straight. Use a little force and rock the chip across the board backwards and forwards to seat them in the spring-loaded holes of the socket board. The chips then insert easily in a horizontal direction symmetrically across the center channel of the board. Four spring-loaded sockets will then be available in a vertical direction to each pin of the chip. Use a small screwdriver inserted under the integrated circuit chips to lever upwards gently when removing them from your socket board. Failure to do this usually results in bent pins on the chips or punctured fingers.

Circuitry in all our experiments will be very straightforward with the use of connecting lengths of single stranded plastic covered, 22 B & S gauge wire (0.06 mm wire diameter). Use of at least ten colors of wire is recommended to facilitate finding incorrect connections when all the wire connections in your circuit diagrams are color coded. Adherence to a color code for each experiment will assist you to find and correct wiring errors. Pay special attention to using red colored wire ONLY for +5-V connections and if possible black wire ONLY for earth or 0-V connections. In this way it is easy to check circuits for chips inserted the wrong way around or power incorrectly connected to the chips. The other eight colors can be used for the bus lines and various control lines.

When stripping the single stranded colored wire, only leave about 5 mm (0.2 inch) of wire exposed at either end. Too much exposed wire could lead to short circuits when many wires are used in an experiment. A suggested color code could be:

Red	+5 V Power
Black	0 V Ground
Brown/Orange/Grey/White	Control lines
Yellow/Green/Blue/Violet	Bus lines

Cost savings can be achieved by purchasing large reels of different colored wires rather than buying prepackaged wires cut to specific lengths. The following tools are required:

1 Long nosed pliers
2 Wire cutters
3 Wire strippers
4 Small screwdriver

A small-tipped, 15-watt soldering iron will also be useful if you intend assembling the printed circuit Interface Board yourself, but we envisage very little soldering for the experiments. Other hardware involved should be relatively inexpensive parts, which can be purchased from your local electronics store.

It is hoped that a kit of integrated circuit chips and other components will be made available with the Interface Board to be used with this book. You may have many of the chips we recommend lying around your workbench at the present time. We would like to emphasize that only LS chips should be used with these experiments as use of the regular 7400 series of TTL chips will lead to too much drain on the Timex/Sinclair power supply and loss of quality in the video picture.

If you intend to wire wrap any of your circuits for a permanent application then you will need to purchase a wire wrap tool, wire wrap wire, and wire wrap sockets and connectors for your circuit.

BREADBOARDING

The layout of a typical solderless breadboard has already been discussed in this chapter. The experiments using the FDZX1 Interface Board with the microcomputer can be performed easily by following the guidelines below.

Gather all the necessary components and integrated circuit chips as listed under *Components* and as noted from the *Schematic* for the experiment. Next insert your integrated circuit chips into the socket board adjacent and as close as possible to the cables from the Interface board. The orientation of the chips should be with pin 1 nearest the left front corner of the breadboard. If you have your chip oriented in the correct fashion then the identification key (notch or dimple in the plastic) will be on the left side of the chip—otherwise the chip is turned around. Pin 1 is invariably marked with a dot or notch on the integrated circuit chip. In this orientation, for a 14-pin chip: pin 1 is in the lower left; pin 7, the lower right; pin 8, top right; and pin 14, top left corner of the chip. The +5-V power jumper from the socket board power rail to the +5-V cable pin should be disconnected.

First, connect all GND pins of your integrated circuit chips to the 0-V rail, then all the V(cc) pins (+5 V) to the 5-V rail of your socket board. Now, complete the rest of wiring of the circuit schematic. It is good practice to have the schematic in front of you at all times while wiring the circuit. Most often, the wires will jumper between the vertical series of four holes at each of the pins of the integrated circuit chips. The schematics have all connections to the integrated circuit chips numbered with the appropriate pin number of the chip. An added tip would be to color code your wire connections and tick off each wire connected on the schematic as it is inserted into the socket board. All interfacing signals can be jumpered in a similar manner from the cable socket pins on the breadboard.

Any resistors or other components should be inserted into the socket board in as nearly a similar position as indicated in the schematic. For example, don't put a resistor used in the input circuitry (left end of your schematic) in the socket board near the right end of your circuit; this sort of wiring leads to considerable crossing of wires and the construction of what is loosely referred to as a "birds nest"!

Your schematic indicates wires that join and wires that cross without an electrical connection as shown in Figure 2.17*a* and 2.17*b* respectively. Once you are satisfied that all connections have been made, check your circuit connections, working from one end of the socket board towards the other. By color coding and cutting your wires to length, neat circuit wiring can be accomplished on the socket board, which greatly enhances your understanding of the circuit layout and assists you in fault finding. It also reduces the amount of radio frequency interference, which can degrade your video display significantly.

Note that the solderless breadboard allows for up to four individual electrical connections to any pin of an integrated circuit on the socket board. This is usually more than sufficient.

We hope that you will not be intimidated by the apparent complexity of the electronic circuits used in the experiments and will be able to construct the circuits easily when you follow the above guidelines, carefully and logically.

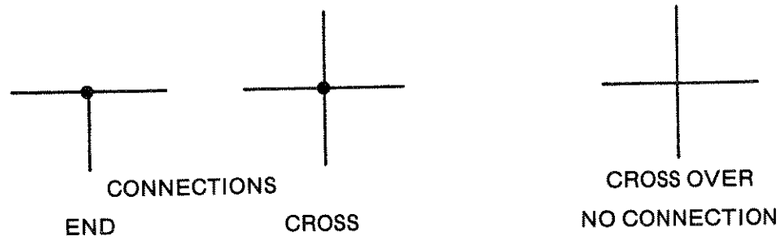


Figure 2.17 Wire Connections and Crossings.

Once you have successfully wired a few interface experiments, beginning with the short simple ones in Chapter 2, then the larger circuits in Chapters 5 and 6 can be constructed with confidence. Good luck, and always remember that you, not the microcomputer, are the master.

CAUTION. Only plug or unplug your interface board with the computer switched off.

It is always good practice if you ever wish to connect or disconnect the FDZX1 Interface Board from your Timex/Sinclair microcomputer to switch off the whole unit while carrying out such an operation. This prevents the possibility of damage to your microcomputer or Interface Board from transient voltage pulses generated whenever circuit connections are made or broken.

EXPERIMENT 2.1
TRUTH TABLES OF COMMON INTEGRATED CIRCUIT CHIPS

COMPONENTS
1 * 74LS00 quad two-input NAND gate
1 * 74LS02 quad two-input NOR gate
1 * 74LS08 quad two-input AND gate
1 * 74LS32 quad two-input OR gate
2 * Jumper leads or logic switches
2 * 3.3-Kohm resistors
1 * LED and 470-ohm resistor or lamp monitor

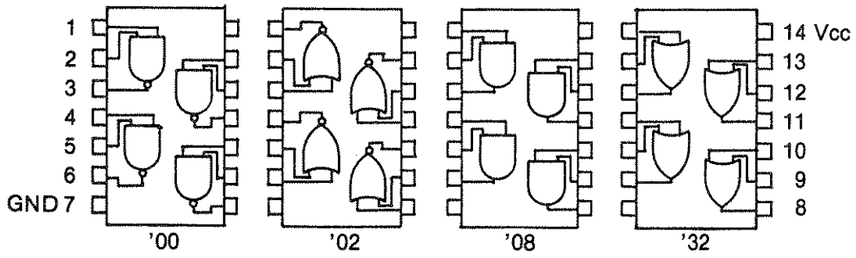
DISCUSSION As explained in this chapter knowledge of the truth tables of certain integrated digital logic chips assists greatly in understanding how logic circuits function. To determine the truth tables of the chips use will be made of logic switches and a lamp monitor. To keep costs down jumper wires can be used as the 0-V or 5-V logic switches and an LED in series with a 470-ohm resistor can be used as the lamp monitor.

PROCEDURE

STEP 1 With the computer unplugged, mount the interface Board to the computer. Position the breadboard socket on the computer case between the keyboard and the Interface Board, and insert the cable sockets from the Interface Board at the right end of the breadboard.

STEP 2 Lamp Monitors and Logic Switches will be used throughout many of the experiments in this book. You may wish to refer to Steps 1 and 2 in Experiments 4.4 and 4.5 for a description of assembling sets of eight of each. Alternatively, small printed circuit boards are commercially available which plug into the breadboard and thereby save room on the breadboard. Further references to lamp monitors will mean LEDs with current-limiting 470-ohm resistors, and references to logic switches will mean the equivalent of power rail jumpers.

STEP 3 Wire up one of the gate chips to the +5-V and 0-V lines on your socket board (with the power off). These four chips all have +5 V going to pin 14 and 0 V going to pin 7. If you mount



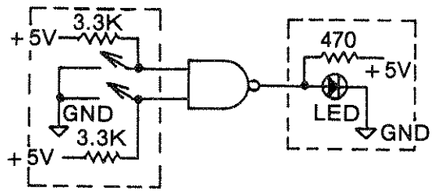


Figure 2.18 Experiment 2.1 Schematic.

your chips on your board so that pin 1 is in the lower left corner, then pin 14 is the top left corner of the chip. Pin 1 will have an indentation nearby to indicate that it is pin 1. Pin 7 then is on the lower right corner of the chip.

STEP 4 Once the IC is connected to the power rails, plug in the computer and jumper the +5-V and 0-V rails to the power pins on the left cable socket. Next, using jumpers to the power rails (or using a set of logic switches), connect the gate inputs on the chip (four gates in each) according to the table in Step 5. Do not leave any of the inputs or outputs to a gate disconnected. If you want 0 V on an input, then jumper that input down to 0-V rail. If you want +5 V on an input, then jumper it up to the +5-V rail. In this way you do not leave the inputs to float to a potential of their own choosing, making your truth tables look wrong.

STEP 5 Complete the truth table for each IC using the lamp monitors to display the gate outputs:

			——— OUTPUTS ———			
	INPUT A	INPUT	'08 AND	'00 NAND	'32 OR	'02 NOR
Gate 1	0	0				
Gate 2	0	1				
Gate 3	1	0				
Gate 4	1	1				

Note that the pin-out of the 74LS02 differs from the other three ICs. You can wire the 74LS00 and then substitute the 74LS08 and 74LS32 directly, but not the 74LS02.

EXPERIMENT 2.2

TRUTH TABLE OF 74LS20 FOUR-INPUT NAND GATE

COMPONENTS 1 * 74LS20 dual four-input NAND gate
4 * Logic switches
4 * 3.3-Kohm resistors
1 * Lamp monitor

DISCUSSION Digital logic gates are not limited to just two inputs as will be shown in this experiment. It is possible to have a number of inputs to any gate but the most common numbers are two, three, four, and eight. It will also be instructive in this experiment to determine what happens to a gate input that is left disconnected.

PROCEDURE

STEP 1 With the power jumper from the +5-V cable pin disconnected from the power rail, remove all other chips from your socket board, and connect your 74LS20 according to the schematic shown in Figure 2.19.

STEP 2 Verify that the truth table for the four-input NAND gate is as below, by jumpering the inputs in sequence to 1s and 0s.

INPUT D	INPUT C	INPUT B	INPUT A	OUTPUT
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
:	:	:	:	
1	1	1	0	1
1	1	1	1	0

STEP 3 Now remove the four jumper leads connected to the inputs so that each input is allowed to float to its own potential. What output logic state do you observe? You should have observed that the output state was a logic 0. What does that indicate about the logic states of the inputs that have been left unconnected? It indicates that when the inputs to the chips are left disconnected they float high; that is, they take up the Vcc potential of +5 V.

The general rule for the TTL integrated circuit chips is that unconnected inputs float high and assume a logic 1 state.

FURTHER DISCUSSION It is as well to note that the 74LS00 series of chips can be joined together in series with the output of one chip driving the input or inputs of succeeding chips. When such connections are used, it is important for the experimenter to be aware that the electronics of the devices (chips) have current limitations. The maximum a 74LS00 chip output can drive is five

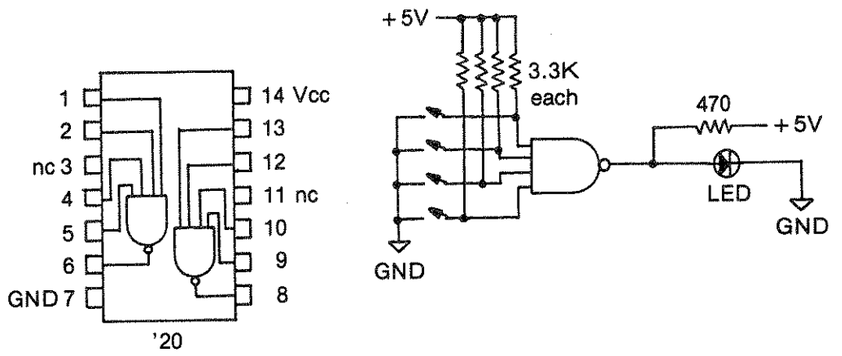


Figure 2.19 Experiment 2.2 Schematic.

regular TTL (7400) inputs or 20 inputs of 74LS00 series chips. This ability to drive other chips is referred to as the *fan out*. The typical maximum current drive limits are:

7400 FAMILY:	REGULAR	LS
Inputs	−1.6 mA	−0.4 mA
Outputs	16 mA	8 mA
Fan out*	10	20

*within family

Because LS chips use less current than regular TTL chips, we have used them almost exclusively in these experiments with the Timex/Sinclair interface so that the power pack is not overloaded.

Should you wish to use TTL chips in any of the experiments you must be aware of the extra current loading on the Timex/Sinclair power pack and the fact that mixing LS and TTL chips can lead to fan out problems (that is, insufficient current drive available for one chip to drive two or more inputs of succeeding chips).

EXPERIMENT 2.3

DEBOUNCED PULSER

COMPONENTS 1 * 74LS00 quad two-input NAND gate
1 * SPDT (single pole double throw) switch
2 * 1-Kohm resistor
1 * Lamp monitor

DISCUSSION We have seen that individual clock pulses can be generated electronically with monostable ICs. They can also be created manually with a switch. A problem is encountered in digital electronics when switches are used to activate clock inputs on integrated circuits such as

counters. Experimenters often wonder why the count displayed by their digital counter rarely agrees with the number of times the switch was activated.

This observation is very important because it highlights the fact that switches are mechanical devices and that their signals need to be conditioned so that digital circuits will respond correctly to a key closure. The problem arises due to the mechanical nature of a switch that uses springs to push the stationary and moving contacts of a switch mechanism into contact. The contacts tend to bounce apart when the lever of the switch is thrown from one position to the other. This bouncing of the contacts causes voltage pulses, the pulses being directly related to the number of times the contacts bounce together. So even if you only move the lever of the switch once, many pulses are produced in a short time, at least for several milliseconds.

A digital counter connected to such a switch will count each and every individual pulse produced by the bouncing contacts and display a count that is anything but related to the number of times the switch lever was moved. To overcome this problem the switch (or any mechanical key closure) has to be "debounced." One method is to use an R-S (Reset-Set) latch circuit made from two NAND gates. Another method, often used with microcomputer keyboards, is to write a time delay into software to wait for the bouncing process to terminate. This experiment will demonstrate how you can construct a hardware debouncer circuit.

By referring to the schematic of the experiment, Figure 2.20, it can be seen that the output of each of the pair of gates is returned to one of the inputs of the other gate. The switch is connected across the two remaining inputs, pins 1 and 5. If we assume that the situation is as shown in the schematic we can use our knowledge of the two-input NAND gate to determine the states of the outputs Q and Q*.

Because the switch is touching contact A, then input 1 of G2 will have a 0 applied to its input. If any of the inputs of a NAND gate are 0, the output must be a logic 1 state. This logic 1 state is applied to pin 4 of gate G2 at the same time a logic 1 is being applied to pin 5 from the 1-Kohm pull-up resistor. Thus the output of G2 is a logic 0 showing that Q and Q* are truly opposite digital states.

Now if the switch is moved over to contact B and momentarily touches the contact, then the input to gate G2 on pin 5 will be a logic 0. Therefore, the output of G2 is put into a high or logic 1 state. This logic 1 state is passed back to the input on pin 2 of G1, which together with the logic 1 state on pin 1 from the pull-up resistor causes the output of G1 to go low to a logic 0 state, the reverse of the original situation.

Now, should the switch bounce off contact B and put a logic 1 on pin 5 again there will be no further change in the output of G2 because the other input, pin 4, is at a logic 0 state and we only require one input to be a logic 0 state for the output to be a logic 1.

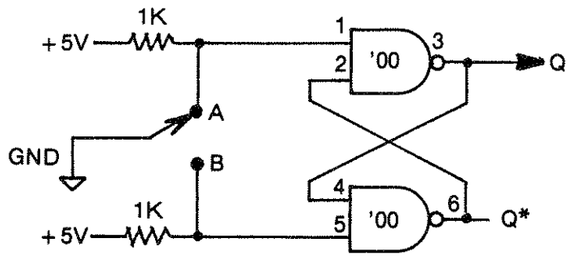


Figure 2.20 Experiment 2.3 Schematic.

So no matter how many times the switch (or key) bounces on the contacts only one change of state will occur at the outputs, that is, only one pulse is produced each time the switch is activated. This is exactly the situation we require in digital electronics to perfect a digital counter.

PROCEDURE

STEP 1 Wire the circuit as shown allowing reasonable lengths of lead to the switch so that it can be operated easily.

STEP 2 Using the Lamp Monitor, probe the pins of the 74LS00 to check the operation of the latch circuit and complete the following table:

SWITCH POSITION	PIN 1	PIN 5	PINS 2 AND 6	PINS 3 AND 4
UP				
Mid-position				
DOWN				
Mid-position				
UP				

This table should verify the description given in the discussion. Obtaining the mid-position of your switch may be a little tricky depending on the nature of the switch.

STEP 3 Leave this circuit wired on the breadboarding socket for the next experiments.

EXPERIMENT 2.4
DIGITAL COUNTER CIRCUITS

COMPONENTS 1 * 74LS90 decade counter
1 * 74LS93 binary counter
5 * Lamp monitors
1 * Debounced switch or pulser

DISCUSSION The ability to count pulses or events is a very useful feature of all digital circuits. There is, however, a need to display the count from a digital circuit, and this is readily accomplished using either Lamp Monitors or numeral displays. The HP 5802 hexadecimal latch display is the simplest to wire, but it is quite expensive. Seven-segment displays are common but require decoder/driver chips, such as the 74LS47, to convert the four-bit number code to the seven-segment display code.

The 74LS90 decade counter chip is an integrated circuit containing four flip-flops. The circuit has the ability to count in binary-coded decimal (BCD) and will count to 9 before resetting to 0 and continuing its count. The outputs of the four flip-flops are available at the pins 8, 9, 11, and 12 and are labeled C, B, D, and A, respectively. To permit the chip to count in decimal, output A at pin 12 must be wired to Input B at pin 1. Pulses from a debounced switch or pulser are input at pin 14, Input A. Other pins available on the chip cause the count to be reset to 0, R0 at pins 2 and 3, and to be reset to 9, R9 at pins 6 and 7. Refer to truth table at end of Chapter 2.

The 74LS93 binary counter chip is identical in pin-out to the 74LS90 counter chip but its outputs are arranged to count the full four-bit binary (i.e. hexadecimal 0 to F). The 74LS93 only has the reset to 0 controls at pins 2 and 3 and not the Reset to 9: pins 6 and 7 have no internal connections.

Using LEDs to display the output condition of the counter where a lit LED corresponds to the logic 1 state and an unlit LED corresponds to the logic 0 state, the appearance of the LEDs will be as follows:

LIGHT EMITTING DIODES				COUNT
D	C	B	A	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

PROCEDURE

STEP 1 Wire the circuit as shown in the schematic, Figure 2.21, (with the +5-V rail disconnected!) together with the debouncer circuit from Experiment 2.3.

STEP 2 By activating your debounced switch you should cause your counter to advance by one count each time. If this does not happen, thoroughly check all your wiring connections.

STEP 3 Each time you throw the switch you create a logic transition. We have seen that the count increments on a negative edge (transition from +5 V to 0 V). Note the direction you move your switch to obtain this transition edge. Does this agree with the output of the debounced circuit lamp monitor?

STEP 4 Now disconnect pin 2 from 0 V and connect it to +5 V. You will note that your counter has reset to 0. If you replace pin 2 back to 0 V and repeat the sequence with pin 7, you will see the counter reset to 9. What are the logic states of the unconnected pins 3, R0(2), and 6, R9(1)?

STEP 5 Now disconnect from the +5-V supply, remove the 74LS90 decimal counter, and replace it with a 74LS93 binary counter in exactly the same position. Reconnect the +5-V power rail, and repeat Steps 2 to 4 again. The count should now progress up to 15 or F on your display. The reset 9 input should have no effect.

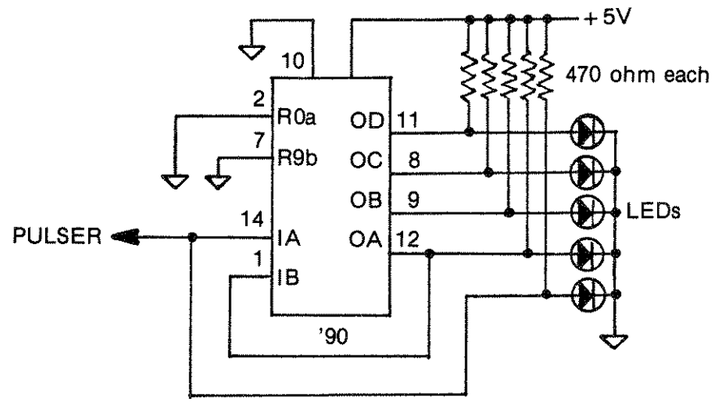


Figure 2.21 Experiment 2.4 Schematic.

STEP 6 Can you suggest a circuit modification to convert the 'LS93 to a decimal counter? First note that you have two free NAND gates on the 74LS00. If you gate outputs B and D of the 'LS93 through the first NAND gate and invert this NAND gate output with the second NAND gate

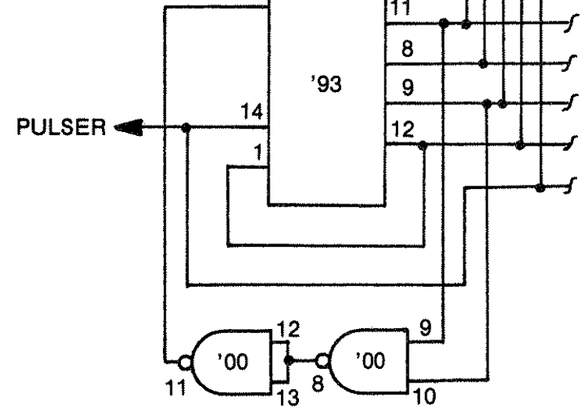


Figure 2.22 AND from NAND Invert.

and then connect this output to R0(1) at pin 2 of the 'LS93, what happens? Why? Your circuit should look like the circuit shown in Figure 2.22.

STEP 7 By selecting any two outputs of the counter and ANDing (NAND and Invert) them to R0, you can make the following modulus counters: 2 (with outputs A and B), 5 (A and C), 6 (B and C), 9 (A and D), 10 (B and D), and 12 (C and D).

STEP 8 You can verify the excessive bounce of a mechanical switch by substituting the undebounced switch. Leave the common lead of the switch connected to the 0-V rail and, after removing the jumper from pin 14 of the counter, insert one of the other switch leads into pin 14. There is no way to predict the number of bounces. If it exceeds 16 you could not know with the present circuit. Can you suggest a way to find out if the number is less than 160 (16 * 10) with the components on hand? How about cascading the D output of the 'LS90 to the A input of the 'LS93? You will need eight lamp monitors.

STEP 9 Save your circuit for the next experiment.

EXPERIMENT 2.5
GATING A COUNTER

COMPONENTS
1 * 74LS93 4-bit Binary Counter
1 * 74LS00 Quad Two-Input NAND Gate
1 * 556 Dual 555-type Timer
4 * Lamp Monitors
1 * Debounced switch or pulser
2 * 0.01 μF Capacitors
2 * 0.10 μF Capacitors
2 * 5.1-Kohm Resistors
1 * each 2.2-, 3.3-, 7.5-Kohm Resistors
2 * 10-Kohm Resistors

DISCUSSION In this experiment we will show how a gate can be used to stop and start a counter. The use of such a system can be extended by using pulses of controlled length from a monostable applied to the gate to turn the counter on and off. In this way very high frequency clock signals can be counted by the counter.

The schematic for this experiment, Figure 2.23, explains the action of the circuit. The controlling gate is an AND gate (NAND and Invert) whose truth table should be reviewed during this explanation. To one input of the AND gate is connected the incoming square wave clock frequency that is to be counted, to the other input is connected the monostable pulse. As you remember, both inputs of the AND gate must be high (in a logic 1 state) before the output goes high. The pulses from the clock are alternately high and low and while the monostable pulse is kept in the low state (logic 0) the output (pin 8) of the AND gate will remain low, so in effect no pulses from the clock will pass through to the counter.

When the monostable pulse goes into the logic 1 state, the AND gate output will go high every time the clock input goes high thus passing the clock pulse through to the counter. By connecting a pulse of known time length on to AND input pin 12, the gate can be opened for a set period and the number of clock pulses counted for that time interval. In this manner, a digital counter can be made to operate as a frequency meter.

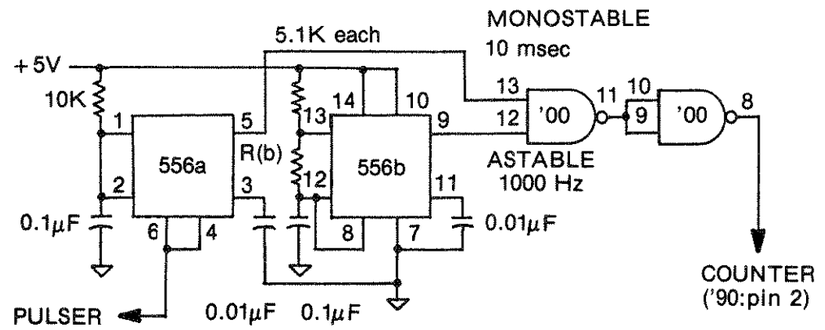


Figure 2.23 Experiment 2.5 Schematic.

The Monostable and the square wave frequency generator (astable multivibrator) will be built from the two 555 timers on the 556 IC. The pulse length for the monostable will be approximately 10 milliseconds with R(a) = 10 Kohm and C = 0.1 microfarad. The frequency of the astable will be approximately 1000 Hz when R(a) and R(b) are 5 Kohm and C = 0.1 microfarad.

PROCEDURE

STEP 1 Remove the +5-V jumper from the cable socket and alter the circuit from Experiment 2.4 as shown in the schematic, Figure 2.23. Do not forget to connect +5 V and 0 V to the 556 chip.

STEP 2. Apply power to the circuit. The monostable trigger is active on the negative edge of the signal from the debounced switch. Set the switch in the logic 1 state. Each time you trigger the monostable, you will have to reset the switch.

STEP 3 Check whether the astable is operating. You can do this by temporarily jumpering the output of the astable to the input A of the counter. Be sure to first disconnect the other lead to the counter input. If the astable is running, all lamp monitors should appear to be on. Due to the rapid counting rate, our eyes are unable to distinguish individual flashes of the LEDs, so they appear to be lit continuously.

STEP 4 Now replace the output of the astable to the gate input of the 'LS00 and the input to the counter to the output of the second (inverting) NAND gate. Lift the lead from pin 2 of the counter out of the 0-V rail to reset the counter to 0 and then replace it.

STEP 5 Trigger the monostable, and observe the count on the lamp monitors. It should be about 10. Without resetting the counter to 0, repeat and record the count about ten times. We obtained the following sequence of counts: 0, 11, 6, 1, 12, 7, 2, 13, 9, 4.

STEP 6 Calculate the difference between successive pairs of your counts. Remember that when the second number is smaller than the first to add 16 to the second number before taking the difference. Our results were: 11, 11, 11, 11, 11, 11, 11, 12, 11. Can you explain why one of our

counts was 12? Remember that we are actually counting negative edges, and the sample taken during the monostable enabling of the gate could occur just before the first negative edge.

STEP 7 You can change the frequency of the astable by changing the R(b) resistor value. We got the following counts (in parentheses) for the following resistances: 10 K (6 or 7), 7.5 K (8 or 9), 5.1 K (mostly 11), 3.3 K (14 only), 2.2 K (2 only—this really had to be 18). The corresponding frequencies are (assuming the monostable was 10 msec): 6500, 8500, 1100, 1400, 1800 Hz, respectively. Of course, the uncertainty is at least 10%.

STEP 8 If we had cascaded the 'LS90 to the 'LS93 as suggested in the last experiment our result for the 2.2 K value of R(b) in Step 7 wouldn't have to be a guess.

SUMMARY Many examples of just how useful gates can be in controlling events will be given later on in this book. This has been a simple experiment to illustrate the principle of a gate.

EXPERIMENT 2.6
DECODING

COMPONENTS 1 * 74LS138 three-to-eight line decoder
6 * Logic switches
8 * Lamp monitors

DISCUSSION *Decoding* is the means by which microcomputers can send information to specific devices. Several bus lines of the microcomputer (called the *Address Bus*) are used to output a binary number (called the *Port Address* or *Device Code*). This number is like the output of a counter and has to be converted into a single enabling clock pulse for the device whose code corresponds to that number. Decoder integrated circuits like the 74LS154 described in the text or the 74LS138 used in this experiment are used to generate unique clock pulses from the address bus.

To see how a decoder functions we will work through an experiment to demonstrate how the device numbers are produced at the eight outputs of the decoder from the various digital codes applied to the inputs of the decoder. The digital codes applied to the inputs are representative of the address information put out by the microprocessor. Each of the eight output lines could then be used to enable one of eight devices.

PROCEDURE

STEP 1 Wire the circuit according to the schematic, Figure 2.24. If you do not have a set of logic switches, use jumper leads to +5 V and 0 V for the digital inputs. The logic state of the outputs will be displayed by the LEDs: logic 1 when on, logic 0 when off.

STEP 2 Apply power and with the three inputs A, B, and C set to logic 0, adjust the three gate switches to obtain a logic 0 (lamp off) state for channel 0 (pin 15). What are the settings on the

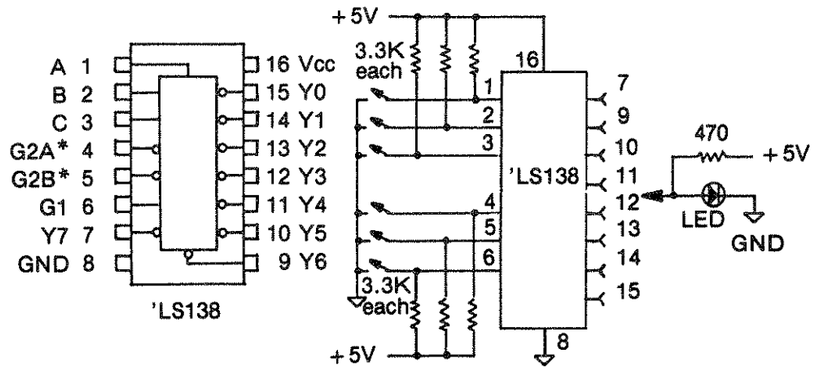


Figure 2.24 Experiment 2.6 Schematic.

gate switches? G1 should be logic 1, and G2A* and G2B* should be logic 0. Any other combination of the gate switches should have all lamps on.

STEP 3 Now complete the following truth table for all the combinations of digital input code.

C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
INPUTS			——— Y OUTPUTS ———							
			0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	1	1	1	1
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

STEP 4 Could you say with certainty that for each combination of digital input code there was only one output line activated? Your answer should be yes. Are the outputs active high or active low?

STEP 5 If the digital inputs were connected to address lines A5, A4, and A3, complete the following table, using the binary weights of the lines, which shows the various decimal numbers required to activate a particular output line when A0 and A1 are always at logic 1 and A2 is always at logic 0.

	ADDRESS LINE INPUTS						OUTPUT LINE 74LS138							
Weight:	32	16	8	4	2	1								
	A5	A4	A3	A2	A1	A0	0	1	2	3	4	5	6	7
	0	0	0	0	1	1	3							
	0	0	1	0	1	1		11						
				0	1	1			19					
				0	1	1								
				0	1	1								
				0	1	1								
				0	1	1								

We will use this table in Chapter 4.

STEP 6 You should, by studying the previous table, be able to deduce (at least, partly) how the 74LS138 decoder chip on the buffered Interface Board produces the unique device select codes as shown on your ribbon cable connectors.

□ 3 □

microcomputer fundamentals

A microcomputer consists of four major components and the support logic circuitry needed to coordinate them. The four components include (1) a microprocessor, (2) a certain amount of memory registers ranging typically between 9 KB and 64 KB (1 KB = 1024 eight-bit registers), (3) an input device, which is usually a keyboard, and (4) an output device, which is typically a video monitor or television set.

A *microprocessor* is a very large scale integrated (VLSI) digital circuit consisting of an arithmetic logic unit (ALU), several registers, and the subsidiary decoding, timing, and control circuitry. Most microprocessors are 40-pin DIP ICs. There are several different microprocessors in use currently, some of which differ significantly from the others in the way they operate. There is one group, however, that functions similarly, is known as the 80 family, and consists of the following microprocessors: 8080, 8085, Z80, and NSC800. To say that the 80 family microprocessors function similarly does not mean that they are identical but that they have comparable internal registers and have a large portion of their instruction set in common. Thus the Z80 microprocessor has an extensive instruction set consisting of 698 distinct operations. The 8080 has 244 operation codes, all of which are included in the Z80's set. We shall refer to these as the 8080 subset when we discuss machine language programming in a later section.

THE MICROCOMPUTER BUSES

All microprocessors can be schematically represented as consisting of three sets of connections exclusive of the power supply connections. Each set is a number of parallel wires (or lines) called a *bus*. The three buses are known as the Data Bus, the Address Bus, and the Control Bus. In its simplest definition, a bus is a common signal pathway. This means that each bus line serves to carry information (digital signals) between the microprocessor and all other components of the microcomputer. Figure 3.1 illustrates the components of a microcomputer and the connections of the three buses. Notice in Figure 3.1 that the Data Bus is drawn with arrow heads pointing in

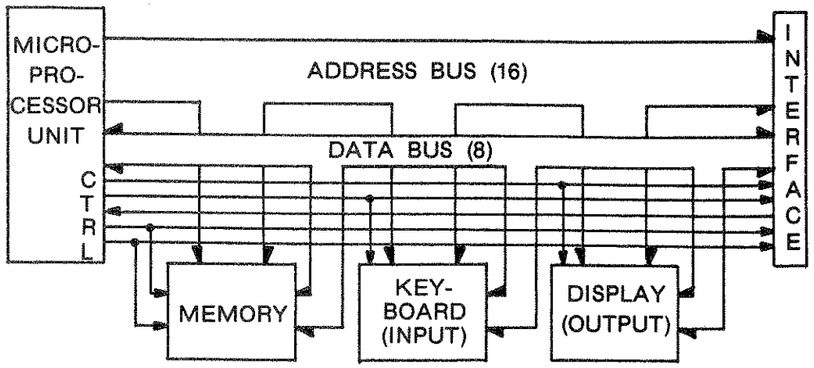


Figure 3.1 Components of a Microcomputer.

both directions. This emphasizes the fact that the Data Bus is bidirectional; data can be transmitted either from the microprocessor to one of the other components or vice versa. The Address Bus is unidirectional and transfers information from the microprocessor to the other components. Also note that the Control Bus is represented as five individual lines having two pairs going out from the microprocessor and one bus line coming into the microprocessor. This is highly symbolic because there are more than just five control lines on the Control Bus. It is drawn thus to emphasize on the one hand the greater individuality of the Control Bus lines while serving, on the other hand, as a reminder that while some of the control lines transfer signals out from the microprocessor, there are some that transfer signals into the microprocessor. For all the various types of microprocessors referred to previously (including those of the 80 family), it is the Control Bus lines that show the greatest variation among microprocessors.

All so-called eight-bit microprocessors have an eight parallel line Data Bus and are thus capable of transferring eight bits of data simultaneously. These lines are labeled D0, D1, . . . , D7. Note that this means that the transmission of a data byte (eight bits) makes possible 256 different "words" or codes. The Address Bus for the eight-bit microcomputers consists of 16 parallel address lines providing for the simultaneous transmission of 16 bits of information representing 65,536 (256 × 256) codes. We shall see that in certain instances it is convenient to consider the Address Bus in two parts, each of which forms one byte and which we will denote as the Low Address Bus (with the eight lines labeled A0 through A7) and the High Address Bus (the lines labeled A8 through A15).

For a given operation of the microprocessor each of the buses serves to provide information that answers one of the questions What?, Where?, When?, and How? the action of the microprocessor's operation is to take place. The Data Bus carries *what* information is to be transferred (either to or from the microprocessor). The Address Bus carries the information of *where* the data byte is to be transferred, that is, to or

from which address. One of the Control Bus lines will signal *when* the data byte is to be transferred and, depending on which one of the control lines was active, determines *how* the transfer is to take place.

Because our interest is in interfacing the Timex/Sinclair models, we shall concern ourselves only with the buses available to us. These lines are physically located at the right rear of these computers and are available as pads, or small tin-plated strips, on the computer's printed circuit (PC) board. The pads are arranged in parallel rows of 23 (28 for the Spectrum and 32 for the TS2068) on either side of the PC board. One pair of the pads has been cut out to form à keyway slot in the board. This slot serves as a keyway guide to prevent misalignment when a PC edge connector is inserted on the board. Figure 3.2 shows the positions of the pads and the signals assigned to each pad as viewed from the rear of the computer. Pad 1 is to the left side as you face the keyboard of the computer. Although the pad numbers differ on the three versions, the relative positions of all the signals with respect to the keyway slot are the same on the B&W models and the TS2000. Only the signals that are different from the TS2000 are labeled on the other two versions. Pads denoted with a dash have no defined signal. The boxes shown on the TS2000 connector are the signals actually used in the Interface Buffer circuit described in Chapter 4. Note that these signals are all identical and that all experiments are possible on any of the five Sinclair and Timex models. The block labeled Interface corresponds to the connection referred to in Figure 3.1.

The easiest of the signals to identify at the interface connector are the eight data bus lines, D7–D0, and the 16 address bus lines, A15–A0. There are four power connections, two of which are labeled 0 V (electrical ground), a third is the unregulated DC supply voltage labeled +9 V, and the fourth is the regulated system supply voltage of +5 V. The remaining 16 (or more) lines make up the Control Bus of the Timex/Sinclair models. We shall have more to say about these signals after we take a closer look at the Z80 microprocessor which is used in the Timex/Sinclair computer. But we should draw attention to four of the control lines that are particularly important to our ensuing discussion.

These four are all output control lines, meaning they originate at the microprocessor and are transmitted to the other components of the microcomputer. Their abbreviations and labels are:

1 MREQ*, Memory Request;
2 IORQ*, Input/Output Request;
3 RD*, Read;
4 WR*, Write.

As control signals, we note first of all that all four are active low. We shall use the asterisk to indicate Control signals whose inactive, or quiescent, state is a logic 1 (+5 V) and whose active state is a logic 0 (0 V). The first two distinguish between the two types of data transfer unique to the 80 family microprocessors. Whereas all microprocessors must be able to request data transfer with the memory registers, the 80 family microprocessors have the additional capability to use the Low Address Bus to request data transfer with an additional 256 input devices and 256 output devices. This is not to imply that microprocessors other than the 80 family members cannot

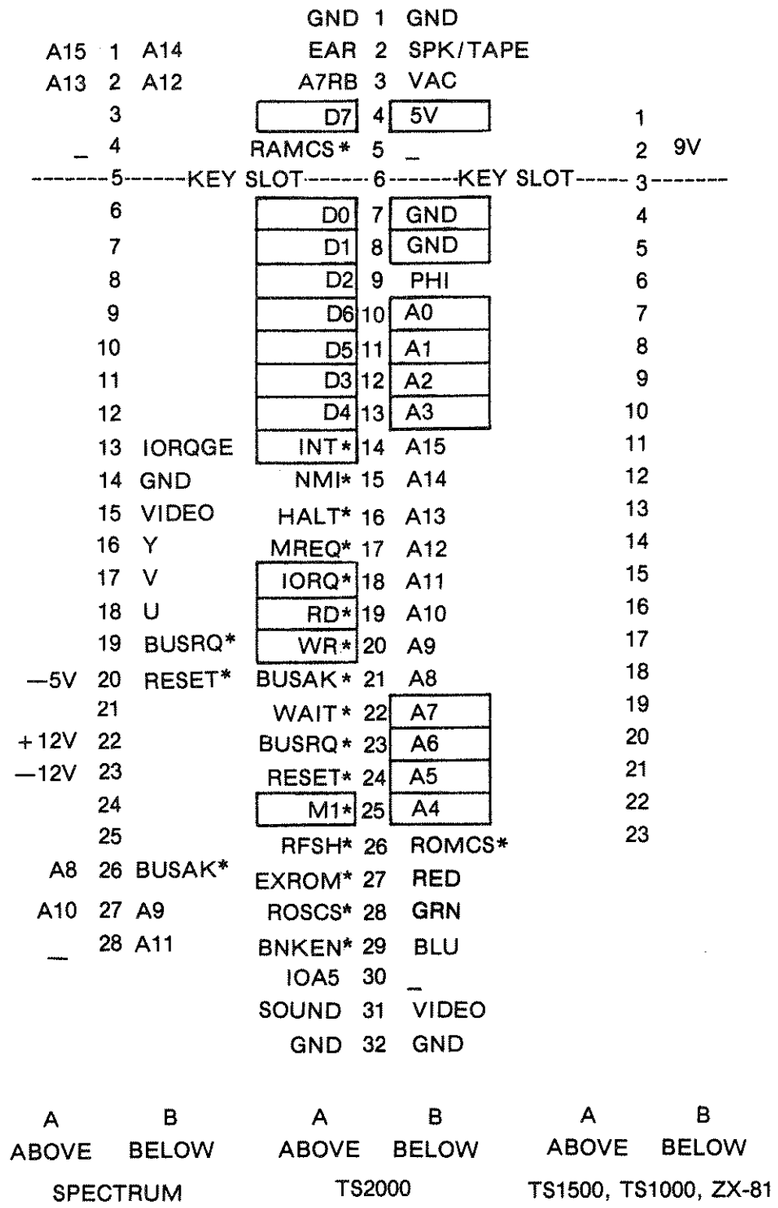


Figure 3.2 Interface Edge Connectors of TS Models.

transfer data to input and output devices, it means that they must make those devices appear to be memory locations and use the full 16-bit Address Bus for addressing them.

The remaining two control lines, RD* and WR*, determine the direction in which the transfer is to take place. Because the reference point for transfer is always the microprocessor, the Read control is used when data is to be read, or input, into the microprocessor. Similarly, the Write control is used when data is to be written, or output, from the microprocessor. It should be apparent that two of these control signals (pulses) must occur simultaneously in order to determine the *how* of the operation. In fact, from the interfacer's point of view, it is more convenient to combine them using some simple logic gates into the four alternate control signals:

1 MR*, Memory Read;
2 MW*, Memory Write;
3 IN*, Device Input;
4 OUT*, Device Output.

Each of these now uniquely provide both the type of transfer (memory or I/O device) and the direction of the transfer. This combination can be made very simply with the use of one integrated circuit. Because we wish our new control lines to be active low, we recall from the discussion on gates in Chapter 2 that the unique output of the OR gate is low only when both inputs are low. The 74LS32 IC consists of 4 two-input OR gates and very nicely satisfies our requirements. The corresponding logic is illustrated in Figure 3.3. We shall return to this concept more fully in our discussion of Device Select Pulses in Chapter 4.

MICROPROCESSOR ARCHITECTURE

We now turn our attention to the logic structure, or so-called architecture, of the Z80 microprocessor. Figure 3.4 illustrates this architecture and we shall have occasion to refer to it periodically as we proceed. The description of the architecture of a

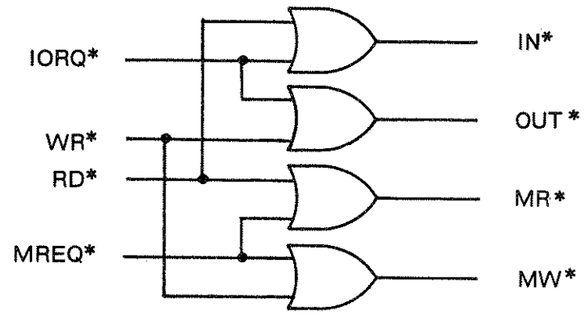


Figure 3.3 Control Logic.

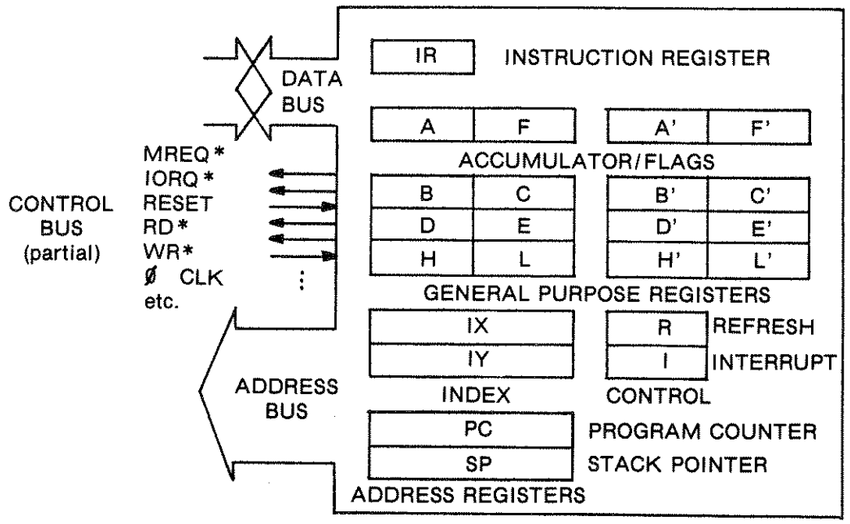


Figure 3.4 Z80 Architecture.

microprocessor amounts to a description not of the network of electronic circuits within the integrated circuit but of the number and kinds of internal registers used to store and manipulate the information stored in them. Examination of Figure 3.4 shows a set of various short and long boxes representing one-byte (eight-bit) or two-byte (16-bit) registers, respectively. The figure also illustrates how the registers relate to the Data and Address Buses. The Control Bus, on the other hand, does not tie in directly to these registers but originates from other support circuitry within the microprocessor. We see that in addition to the four control lines already discussed, two additional control lines labeled Reset and ϕ (Phi) have been included to aid in our discussion.

Each register is an internal memory register. There are additional internal registers in the microprocessor but they are used by the microprocessor for internal manipulation and cannot be accessed or manipulated through programming. All of the registers shown in Figure 3.4 can be manipulated with programming instructions. There are two sets of six general purpose registers, which serve as storage registers of program variables although only one set is accessible at any one time. With certain instructions, the general-purpose registers can function in the pairs BC, DE, and HL as 16-bit registers with the first-mentioned register of each pair serving as the more significant byte. Of particular importance, the HL register pair can be used as an address to index (point to) any memory register in external memory. The IX and IY registers are also index registers similar to the HL pair but more versatile. They become useful in programming techniques more advanced than are necessary for our purposes. The Refresh and Interrupt registers are special-purpose registers whose use

is beyond our needs. We now focus our attention on the remaining special-purpose registers, which are of considerable importance in understanding how microprocessors work.

The A register is most commonly referred to as the *Accumulator* for the very good reason that it is the only register in which arithmetic and logical operations can be carried out with the general-purpose registers by the microprocessor. The results of these operations are saved (accumulated) in this register. The only "true" arithmetic that the microprocessor can do, besides counting, is addition and subtraction between one of the general-purpose registers and the Accumulator. You might recall, however, that multiplication and division really only amount to counted additions and subtractions, respectively. The logical operations that the microprocessor performs are all done bit-by-bit and consist of negation of A (that is, bitwise inversion, also called *complement*, where each logic 1 is converted to a logic 0 and vice versa); and ANDing, ORing, and Exclusive ORing between A and one of the general-purpose registers, just as was described with two-input logic gates in Chapter 2. As indicated in Figure 3.4, most of the transfers of programmed data in and out of the microprocessor use the Accumulator as the destination or source register.

The F, or Flags, Register is closely connected with what happens in the Accumulator. Although it is an eight-bit register, only six of the eight bits are active, the other two are dummies. The six active bits are used to signify (signal) certain information about the data byte which resulted from the last (most previous) math instruction (arithmetic or logic). The three most important flag bits will be described, the remaining three (Negate, Parity/Overflow, and Half Carry) are of no use for our purposes

F7, the most significant bit of the Flag register, is the *sign flag:* when the result of the last math operation resulted in the most significant bit of the result being in a logic 1 state then F7 = 1. Why this is referred to as the Sign Flag requires explanation. We all understand that the sum of two numbers of opposite sign but equal magnitude (+X and −X) equals zero. Therefore, we can define the negative of a number as that number which yields zero when added to the number in question. Thus, X + (−X) = X − X = 0. Now, we know that eight-bit numbers can be added, for example

```
  0 1 0 1 0 1 0 1
+ 0 1 1 1 0 1 0 0
  ---------------
  1 1 0 0 1 0 0 1.
```

What eight-bit numbers can be added to yield a sum of zero? Consider the sum:

```
    0 0 0 0 0 0 0 1        1
  + 1 1 1 1 1 1 1 1      255
    ---------------      ----
1   0 0 0 0 0 0 0 0       256
                        −256   Carry
                        ----
                           0   Result.
```

Notice a ninth bit (Carry) was generated. Obviously, if 00000001 = 1 then 11111111 = −1! In fact, for the decimal range of eight-bit numbers from 0 to 255, we can consider 0 to 127 as positive numbers and 128 to 255 as negative numbers when it suits our purpose. In binary notation, the positive numbers have D7 (the leftmost or most significant bit of the eight-bit byte) = 0 while negative numbers have D7 = 1. There is a simple trick for finding the negative of a binary number called the "two's complement of a number": add 1 to the inverse (complement) of the number. For example, the complement of 00000001 is 11111110; if we add 1 (00000001) to this result we obtain −1 or 11111111. We shall have occasion to check the sign flag when we program the Z80.

The second important flag is the *ZERO Flag* on bit F6. If the result of a math operation was eight bits of 0s, then the Zero Flag is set (raised?) by becoming a logic 1. Take care to note that if the Zero Flag is zero (F6 = 0), then the math result was *not* zero!

We have already encountered the third important flag, which is the *CARRY Flag* on bit F0. The Carry Flag serves as a ninth bit for all arithmetic operations (not logical operations because they are done bit-by-bit). Actually, the Carry Flag or Carry bit also serves as a BORROW bit when a subtraction is performed. We have already seen an example of the Carry as the ninth bit created in an addition when we added 1 to −1. Let's examine the Carry Flag when we subtract one byte from another. For example,

```
    0 1 1 0 0 1 0 1        101     Minuend
  − 0 1 1 1 0 0 1 0      − 114     Subtrahend
  -----------------      -----
1   1 1 1 1 0 0 1 1        −13
                         + 256     Carry
                         -----
                           243     Result
```

But subtraction is just adding the negative of the subtrahend by taking its two's complement, then

```
  0   0 1 1 0 0 1 0 1        101
+ 1   1 0 0 0 1 1 1 0        142
  -------------------      -----
  1   1 1 1 1 0 0 1 1        243.
```

Notice that just as every positive number may have any number of leading zeros (to its left), every negative number must have that many leading ones.

Dealing in binary arithmetic takes some getting use to. Most personal computers are programmed to translate bytes into decimal numbers when they are displayed to the user. It may make you more comfortable also to deal in decimal numbers. If you do, then when it comes to knowing whether the Carry Flag is set to 1 (or cleared to 0) from either a carry-on addition, or a borrow-on subtraction, you have to see whether the decimal result is greater than 255 or less than 0 (i.e., is negative). Remember that 255 is the largest value a byte can have. Whenever you add two decimal bytes and the

number is greater than 255, subtract 256 from the number and set the Carry Flag to 1. Whenever you subtract two numbers and obtain a negative result, you must add 256 to the result and set the Carry (borrow) Flag to 1. This has been illustrated to the right of the binary arithmetic in the above examples.

Before describing the remaining three registers, we have to consider how the microprocessor manages to run. The Control Line labeled ϕ (Phi) carries a continuous train of digital clock pulses (i.e., it is a square wave). Each rising and falling edge of this train is used electronically by the microprocessor to trigger the next appropriate event. As a point of reference, the clock for the Z80A microprocessor can operate up to frequencies of 4 MHz. (4 million pulses per second.) The frequency in the Timex/Sinclair is actually 3.25 MHz. This means that each square wave is 0.308 μsec or 308 nanoseconds long.

The other control line we showed in Figure 3.4 was the Reset input line. When the Reset control is made a logic 0 by momentarily grounding it with a pushbutton switch or, as in the Timex/Sinclair, when the power is turned on to the computer, the 16-bit Program Counter register is reset to zero.

The operation of a microcomputer starts by fetching a byte of machine code from external memory at the memory address held in the Program Counter. The 16 bits of the PC are placed on the Address Bus and the Control Lines MREQ* and RD* are activated (momentarily made logic 0). The addressed memory register is enabled and puts the contents of its eight-bit register on the Data Bus. The microprocessor reads the byte on the Data Bus and latches it into the Instruction Register. The Program Counter is incremented by the next appropriate ϕ (Phi) clock pulse so that it once again is pointing to the next program byte in memory and the microprocessor proceeds to electronically decode the instruction byte into the proper sequence of actions appropriate to the instruction code.

It is convenient to group the train of clock pulses according to the action of the microprocessor in executing an instruction. Each group is called a *machine cycle*, and, in the Z80, the shortest is four clock pulses in duration. Depending on how complicated the instruction is, several machine cycles may be required to execute the instruction. The set of machine cycles is called the *instruction cycle*. Each unique instruction is always the same number of machine cycles and, therefore, the same number of clock cycles when it is executed. For example, if a particular instruction takes 18 clock cycles to be completed, then you can calculate that it will always take $18 \times t$ (where t is 308 nanoseconds on the Timex/Sinclair) or 5.54 μsec to perform that instruction.

The first machine cycle of each instruction cycle is always a Fetch operation in which the instruction code is loaded from memory into the Instruction register. When the microprocessor starts a new instruction it activates another Control line called M1 during the first machine cycle. We will have occasion to refer to the M1 Control signal in Chapter 4.

There remains only one register to complete our description of the architecture of the Z80 microprocessor. It is the 16-bit Stack Pointer register. There are many instances when the microprocessor needs more storage space than its internal registers can hold at any given time. This problem is solved by using external memory to store any number of bytes in a kind of scratch pad called the *Stack*. This storage area is

appropriately called a stack because for however many bytes are stored on it, the Stack Pointer always holds the address of the last entry only. In other words, the Stack operates like a push-down list of numbers. The last-in entry must be the first-out. The Stack starts at a higher memory address and as bytes are appended, they are placed at consecutively lower addresses. As bytes are recalled from the Stack, the Stack Pointer automatically increments. Each stack operation consists in loading (PUSHing) or removing (POPing) two data bytes. Thus, the Stack Pointer is always incremented or decremented by two. The two bytes that are PUSHed or POPped may be either of the register pairs: BC, DE, HL, IX, IY, or the Program Counter, PC. The more significant byte (B, D, H, or Hi Address) is always stored first (at the higher stack memory address).

MACHINE AND ASSEMBLY LANGUAGE

We have already mentioned that the Z80 microprocessor has a set of 244 of its 698 instructions, which we referred to as the 8080 subset. Because each instruction must be decoded by the Instruction register as one byte, and a byte can be any number between 0 and 255, it is apparent that there can only be a maximum of 256 one-byte instructions or operation codes (opcodes). If the 8080 subset uses 244 of these 256, then there are 12 unused bytes in the 8080 subset. For the Z80 to have 698 instructions must mean that most of the extra (Augmented) Z80 instructions consist of more than one byte. This is exactly the case. Four of the unused 12 opcodes are used as prefixes; that is, when one of these four bytes is received by the Instruction Register, it signals the microprocessor to fetch the next memory byte and decode it as an Augmented Z80 opcode. The decimal values of the prefixes for the augmented Z80 opcodes are 203, 221, 237, and 253. With these four of the 256 code values accounted for, we have only to try to understand 252 others! Actually, we will not say much more about the augmented Z80 instructions but confine most of our attention to the 8080 subset.

Because it is tedious to work with the opcodes as numbers, a set of abbreviations, which serve as memory aids (mnemonics) for the machine language programmer, was developed by the microprocessor manufacturer. These mnemonics describe each type of operation performed. It is much easier to write a program in mnemonics and then assemble it to the list of numbers that have to be entered into the computer memory. The set of mnemonics is called the Assembler Language of the particular microprocessor.

Six charts, included in Appendix A, can be cut out and glued onto cards to form a sliding chart for assembling machine code. We shall describe the Z80 machine language with reference to these charts. You may want to put the chart together before proceeding; however, that won't be necessary for our discussion.

The entire instruction set can be grouped into seven classes of operations. These are:

1	Math (arithmetic and logic)	8080 subset
2	Register transfer	8080 subset plus prefixes 221 and 253
3	Stack reference	8080 subset
4	Program branch	8080 subset
5	Miscellaneous	8080 subset
6	Relative branch	Z80 augmented set
7	Bit reference	Z80 augmented set (prefix 203)
8	Block reference	Z80 augmented set (prefix 237)

The first five classes are included in Chart A.1 in the Appendix. The Math ops are listed in the three boxes in the upper half of the two righthand columns, and the Transfer ops are listed in the three boxes in the lower right half. The second column from the left are the Stack reference ops grouped in four boxes while the top three boxes in the leftmost column are the program branch ops. The Miscellaneous ops are listed in the box in the lower left corner.

The remaining three classes are included in Chart A.4. The small box in the upper left corner contains the relative branch ops and the two large boxes list the operations for the Bit and Block reference ops.

To give you some idea of the actual organization and logic of the eight-bit instruction codes, we have to refer to our earlier discussion of the octal number base. We can quickly cover half of the 8080 subset of instructions with the register Transfer operations and the Math operations.

A register transfer operation consists of loading the contents of one register (the source) into another register (the destination). Recall that there are eight registers available for these one-byte transfers, namely, A, B, C, D, E, (HL), H, and L, where (HL) is any external memory register whose address is stored in the HL register pair at the time. For convenience, we also refer to (HL) as register M. Because there are eight registers, they can be individually coded with three bits as: B = 0, C = 1, D = 2, E = 3, H = 4, L = 5, M = 6, and A = 7. By dividing the eight bits of the instruction code into three-bit octal groups (only two bits for the left group) as

X X	X X X	X X X
Op Code type	Destination register	Souce register

and defining the Op Code type as 01, we can see the logic of the machine language operation codes. For example, the opcode for LD A,C (read as "load A from C") will be

0 1 1 1 1 0 0 1

or 171 (Octal) which, on conversion to a decimal value, is

$$(1 \times 64) + (7 \times 8) + (1 \times 1) = 121.$$

Find this value on the Decimal Assembler. If you have put the Assembler together, locate the expression "LD A," in the lower right Transfer box on Chart A.1 and pull the slide out until the C column lines up. You should now be able to read: LD A,C 121. If you are reading from the charts, read the C column on Chart A.2 on the tenth row up from the bottom of the chart.

All 64 "LD register, register" opcodes are similarly encoded except LD M,M which is not encoded in this manner. Because it doesn't do anything useful, the LD M,M code, 166 in octal, is used for the HALT instruction. Verify its decimal value ($1 \times 64 + 6 \times 8 + 6 \times 1$) by finding it in the lower right box of Chart A.1. We have now covered about one-fourth of the 8080 subset.

We can cover another one-fourth by understanding how the octal-based machine code for the 8 one-byte Math operations are encoded. Recall that these are all performed with the Accumulator register and one of the other eight registers. Notice that there are eight of these operations, so again they can be coded with a three-bit octal digit. These are:

MNEMONIC	CODE	OPERATION
ADD A,r	0	LET A = A + r
ADC A,r	1	LET A = A + r + Carry
SUB r	2	LET A = A − r
SBC A,r	3	LET A = A − r − Carry
AND r	4	LET A = A AND r bitwise
XOR r	5	LET A = A XOR r bitwise
OR r	6	LET A = A OR r bitwise
CP r	7	LET A = A but F = result of A − r

where *r* is any one of the eight registers. The Carry bit (flag) being added or subtracted for codes 1 or 3 respectively in the list is treated as the least significant bit with seven leading zeros. The logical operations of codes 4–6 are bit-by-bit operations just like we described in Chapter 2. The Compare operation, CP r, does not alter the A register's value but does set the flags in the F register according to the result of subtracting r from A.

Because the Accumulator register is always involved in the Math ops, the operation code needs only to specify which Math op code and what other register is to be used. The Octal partition is:

X X	X X X	X X X
Op code type	Math operation	Register code

where the Op Type code for Math ops is 10. For example, the op code for ADC A,E (read as "Add E with Carry to A") would be

1 0 0 0 1 0 1 1

or 213 (octal) and 139 (decimal).

There is similar order for the rest of the machine code but for codes beginning with octal digits 0 and 3 are arranged somewhat differently. We leave the remainder of octal decoding of the 8080 subset to the interested reader. The point of this discussion, besides introducing over half of the machine code in the easiest way, is to illustrate just

how the Instruction Register of the microprocessor can analyze the eight bits of an instruction. By reading the first two bits (octal 1 or 2), it can decide how to interpret the remaining six bits as register and/or Math op codes in two sets of three bits.

We now turn our attention to the other types of machine language instructions available. You may have noticed that in addition to the eight columns for registers A, B, C, D, E, M, H, and L, there are three other columns in the Transfer and Math columns on Chart A.2 which are labeled N, X, and Y. The references to N (number) are called *immediate references* where the number N is the byte immediately following the instruction in the program listing. This second byte of the instruction is called an *operand* and allows the constant N to be incorporated directly into the program. Notice that it is a constant and not a variable. Once it is incorporated into the program, it cannot be changed without changing the program. For example, suppose at some point in your program (say at memory location MEM = 16540) you knew that register A held some number from which you wanted to subtract the number 8. You would use the instruction SUB N, <8>, and your program list would include:

ADDRESS	LABEL	MNEMONIC	CODE
16539	MEM−1	:	:
16540	MEM	SUB N	214
16541	MEM+1	<8>	8
16542	MEM+2	:	:

These instructions are two-byte instructions where the first byte is the op code and the second byte is the operand.

If you look over the other boxes on Chart A.1 you will also find some instructions which include a double N or NN reference. If you guessed that these are three-byte instructions consisting of an op code followed by two operands, you were right. There are two ways the NN reference is used. In the *direct form,* you can load any register pair with a two-byte number. The register pairs that you can load immediately include BC, DE, HL, as well as the Stack Pointer, SP, and the Index registers, IX and IY. Your program list, again at some memory address, MEM, would read:

MEM−1	:	:
MEM	LD HL,NN	33
MEM+1	<N(Lo)>	130
MEM+2	<N(Hi)>	64
MEM+3	:	:

where the first operand byte (130 in the example) is loaded into the low register (C, E, L, etc.) and the second operand byte (the one at the higher memory address, the number 64 in the example) is loaded into the high register. The second way that the NN byte is referenced is in parentheses as, for example, LD A,(NN). This is the *indirect form.* In this form, (NN) refers to the contents (one byte) at the memory address NN. Whenever you encounter one or two Ns in the Assembler Language it reminds you that the instructions have one- or two-byte operands which must be included in the program list immediately following the opcode.

We noted earlier that there are also Math and Transfer instructions, which refer to columns labeled X and Y. These are just abbreviations for the Index registers IX and IY. All of the instructions that reference the Index registers belong to the Z80 Augmented set and not the 8080 subset. This means that the opcode is preceded in the program list with a prefix byte. You might recall that the IX and IY registers can perform every operation that the HL register pair performs. If you examine these instructions, you will note that they have the same opcode as the corresponding HL instruction and that they differ only in having a prefix. The other difference between HL and the Index registers is that in indirect references, (HL) = M, (IX) = X, and (IY) = Y, a relative displacement from the indexed memory address must be specified. This is summarized in the third box in the second column on the right in Chart A.1. On the slide table, Chart A.2, the prefix is denoted by a *p* and the displacement by a *d*. Thus, these are three-byte instructions of the form, for example, for LD (IX+d),A:

MEM−1	:	:
MEM	IX Prefix	221
MEM+1	LD (HL),A	119
MEM+2	Displacement	−128 to +127
MEM+3	:	:

Note that the displacement is given in two's complement notation as we discussed earlier. Thus, not only can we reference the memory location pointed to by the IX or IY register, but any memory location spanned by the displacement value without changing the value of the Index register. If you specifically wanted to reference the address held in the Index register, you would have to give a displacement of zero.

The remaining Math ops include the 16-bit ADDs and both 8- and 16-bit increment (add 1 to the current value) and Decrement (subtract 1) instructions. A particularly useful set of Math instructions cover the Rotate operations. The 8080 subset includes four of these, which operate on the Accumulator. The Z80 Augmented set has seven of these Shift/Rotate instructions, which can operate on any of the ten registers: A, B, C, D, E, H, L, M, X, and Y. These augmented instructions use the prefix 203 and include the four Rotate ops of the 8080 subset. Each of the seven types is most easily described by the diagram in the upper right box on Chart A.4. The data bits of the register are shifted one bit position to the right or left, with some of the instructions including a ninth bit (the Carry flag) in the operation. Note the redundancy between the first four in this box on the A column and the box on Chart A.1 without the prefix. The remaining four Math ops include the decimal adjust accumulator (DAA), complement the accumulator (CPL), set (to logic 1) the Carry flag (SCF), and complement (change its logic state) the Carry flag (CCF). (Except for DAA, these are self-explanatory. The DAA is a binary-coded decimal operation of little immediate interest.)

The Stack operations are all two-byte Transfer operations whose descriptions are straightforward. The EX mnemonic stands for "Exchange" and corresponds to a swap between the registers specified (direct or indirect). The PUSH and POP operations were discussed previously and are of significant use for saving and restoring values of the registers with the Stack.

The major branching instructions include Jumps (like GOTO in BASIC), Calls (like GOSUB in BASIC), and Returns (just as in BASIC at the end of a subroutine). These instructions are either unconditional (JP NN, CALL NN, RET) or conditional. The decision for the conditional branches are made on the Flag register bits with two choices (1 or 0 for the particular Flag bit). These are shown in the left table in Chart A.2 and the corresponding box on Chart A.1. The two-byte operand for the Jumps and Calls are the specific memory address (Lo address byte first, Hi address byte second). The RET are one-byte instructions because the return address of a Call is automatically PUSHed onto the Stack when the Call is executed and automatically POPped off the Stack when the Return instruction is executed.

The Restart instructions, RST X, are one-byte Call subroutine instructions having fixed address destinations. These instructions automatically branch to High Address 0 and Low Address 0X0 (Octal base!) where X is the RST number 0 through 7. These instructions are of little utility to the user of a Timex/Sinclair because their destinations are in the ROM using dedicated subroutines.

We have already seen that the 8080 subset consists of 244 opcodes and that of the 12 unused codes (out of the 256 possible ones), the Z80 Augmented set used four for prefixes. The remaining eight are also used in the Z80 Augmented set but without prefixes. These are shown in the upper left corner of Chart A.4. Six of the eight are branch instructions having a very important distinction from those just described in the 8080 subset. These six branch opcodes are relative jumps. Whereas it was necessary to given an absolute (two-byte, NN) memory address for the destination of the 8080 jump opcodes, JP; the relative jumps, JR, use a one-byte displacement operand from the current value of the Program Counter. Because the Program Counter will point to the next opcode in the program after it has fetched the operand byte from memory, that memory address becomes the reference or zero point for the relative jump. We have seen that a one-byte number can, when convenient, be treated as a positive or negative number. In calculating the displacement this technique is used for the relative jumps. For example, a jump of ten locations forward in the program is +10 while a jump of ten locations backward will be 256 − 10 = 246. Therefore, from the PC reference point, a relative jump can be executed up to +127 memory locations forward and −128 = 256 − 128 = 128 locations backward.

The advantages of the relative jump compared to the absolute jump are twofold. It saves program space by using fewer bytes, and more importantly, the routine that contains a relative jump is independent of absolute location in the program and can be relocated without having to respecify the destination address.

Of the six relative jump opcodes, one is unconditional and four are Flag conditional with branching occurring on Nonzero, Zero, Noncarry, or Carry. The sixth, DJNZ, is also conditional but instead of depending on a Flag bit, it is made on the value held in the B register. Each time a DJNZ instruction is executed, the B register is decremented, and the decision to branch is based on whether B is Nonzero. If B is Nonzero then the jump is made, otherwise the next opcode to be executed is the one following the DJNZ. This is a particularly useful instruction and is comparable to the BASIC FOR . . . NEXT command for repeating a sequence of instructions a number of times (equal to B) when the displacement operand is negative, before proceeding.

The remaining two unprefixed instructions of the Z80 Augmented set are register

transfer instructions. EX AF,AF' exchanges the current accumulator and flags registers with the alternate pair, while EXX exchanges the current six general-purpose registers with their alternate counterparts. The values stored are preserved and can be recovered on the next exchange instruction.

The six so-called Miscellaneous instructions of the 8080 subset are shown in the lower left box of Chart A.1. The NOP instruction is a do-nothing or "no operation" useful for allocating space in memory or using up time in execution. The EI, DI, and HALT instructions are of significance for Interrupt servicing where external devices can pulse the Interrupt Control line and cause the microprocessor to branch out of the program it is currently executing. EI (Enable the Interrupt) and DI (Disable the Interrupt), permit or prevent the Interrupt control line from activating the microprocessor. HALT stops the program with recovery only possible by either pulsing the Reset Control line, or from a pulse on the Interrupt Control line after the HALT instruction has been executed, and this only provided that the Interrupt had been enabled (EI) previous to the HALT. This latter technique is used excessively by the Timex/Sinclair B&W models in SLOW mode.

The last instructions are the most essential ones for the Timex/Sinclair interfacer. Although many versions of BASIC on 80 family microcomputers include commands INP and OUT, the Timex/Sinclair does not. We must use the IN A,(N) and OUT (N),A machine instructions in BASIC USR (User) routines if we wish to communicate with external devices. These are two-byte instructions whose second byte (operand) is the device code. We shall have more to say about these instructions in Chapter 4.

Now that we have surveyed the range of machine language instructions, we shall conclude our discussion of the Z80 microprocessor by referring to the Table included on Chart A.3. As we noted previously, each machine instruction takes a definite amount of time to execute. This time, *t*, is measured in number of clock cycles, where the actual duration of each clock cycle depends on the speed at which the microprocessor operates (308 nanoseconds per clock cycle on the Timex/Sinclair B&W models operating at a frequency of 3.25 MHz). There will be occasions when we need to know specifically how long a portion of a program takes to execute. By referring to the clock cycles table, the time can be computed by summing the *ts* for each instruction in the routine. All Z80 instructions are given in the table. The three columns in the left box of the table cover single register reference, indirect register pair reference, and direct register pair reference instructions. The various branch instructions and the remaining instructions are given in the right box of the table. Note that for the conditional branches, the number of clock cycles depends on whether the decision is or is not met.

EXPERIMENT 3.1

THE BASIC USR FUNCTION AND MACHINE CODE STORAGE

DISCUSSION The USR function allows machine language routines to be executed from a BASIC program. In the Timex/Sinclair, this function calls a machine language subroutine. The format of the USR function can be put into a LET command; for example, the format we will use is:


```
### LET L = USR M
```

where ### is the BASIC program line number. Although any variable can be used, we have used L because it is the same key as LET, =, and USR. The argument of the function, M, is the starting (destination) address in memory of the machine language subroutine.

After the USR function has been executed, the value assigned to the variable L equals the 16-bit value held in the microprocessor's BC register pair on return to the BASIC program. If the BASIC program recalculates L into two bytes by the following lines:

```
150 LET B = INT(L/256)
160 LET C = L - 256*B
```

we can recover the contents of registers B and C. Recall that the decimal value of a 16-bit number is obtained by the equation:

$$L = 256*(MSBy) + LSBy$$

where L is the 16-bit decimal value, MSBy is the more significant decimal byte (register B in our case), and LSBy is the less significant byte (register C).

We noted in Chapter 1 that different addresses have to be used for the B&W and Color models. This is primarily because the only simple way to SAVE and LOAD machine routines on the B&W models is by storing them within a BASIC program. The particular address M = 16514 is unique in the B&W computers (ZX81, TS1000, and TS1500) because it is the first memory location available "inside" the BASIC program. The B&W operating system always starts storing a BASIC program at memory location 16509. Five bytes are used to hold the line number (two bytes), the length of the line (two bytes), and a command code (one byte). If we make the first line of the BASIC program a REM statement followed by as many bytes (or character spaces, because each character uses one byte) as are needed to store the machine language subroutine(s), the first character will be at memory address 16514(B&W). The space occupied by a REM statement is ignored by the BASIC interpreter when the program is RUN. Editing the rest of the BASIC program does not change the address locations of the code and the entire program including the machine language routine(s) can be SAVEd on cassette. Machine code can be easily SAVEd and LOADed with the Color models. Therefore, it is easier to store the machine language routines at the top of memory where the operating system cannot disturb it. To keep our instructions simple, we will store our machine code at the same addresses in the TS2000 and the Spectrum. Because top of memory for the 16K Spectrum is 32767, we have selected address 32130 as the starting address of our routines for the Color models. The advantage to using this address is more apparent if the separate high and low bytes of the B&W and Color addresses are compared:

MODEL	ADDRESS	LOW ADDRESS BYTE	HIGH ADDRESS BYTE
B&W	16514	130	64
Color	32130	130	125

Thus only the high address byte will be different in our machine code listings when an absolute address is referenced.

BASIC PROGRAM

```
  1 REM 123456789 123456789 1234567890        (for B&W models)
  1 CLEAR 32129                               (for Color models)
 10 LET Z = 0
 20 PRINT " NUMBER OF BYTES? "
 30 INPUT N
 40 LET M = 16514                             (for B&W models)
 40 LET M = 32130                             (for Color models)
 50 PRINT " ENTER CODE: "
 60 FOR I = M TO M+N-1
 70 IF Z <> 0 THEN GOTO 100
 80 INPUT B
 90 POKE I,B
100 PRINT I; " = " ;PEEK I
110 NEXT I
120 PAUSE 33333
130 LET Z = 1
140 LET L = USR M
150 LET B = INT (L/256)
160 LET C = L - 256*B
170 PRINT " B =  " ;B, " C = '';C
```

PROCEDURE

STEP 1 ENTER the BASIC program. Throughout the rest of the experiments, differences for the B&W and Color models will be indicated with the same line number listed twice as shown in this listing. DO NOT enter the statements "for . . . models" into the BASIC line. The number of bytes that need to be allocated in the REM statement are shown in groups of ten (counting the space) with all digits shown in the last group before <ENTER>. The CLEAR # command for the Color models protects the memory above that address from the operating system.

STEP 2 Our first project is to verify that the USR function returns the B and C register values. To do this, we need a simple machine language subroutine to Load Immediate BC with a pair of numbers and return to BASIC. The instruction we need is: LD BC,NN. Our subroutine will be:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD BC,NN	1
16515 / 32131	<N>	201
16516 / 32132	<N>	1
16517 / 32133	RET	201

How many bytes long is the subroutine? Your answer should be 4.

STEP 3 RUN the BASIC program. Respond to "NUMBER OF BYTES?" with 4 and ENTER.

STEP 4 Now respond to "ENTER CODE:" by ENTERing each of the four numbers of the machine language subroutine in succession. The program will print your entries on the screen so you can check your entries against the list. If you make a mistake, press BREAK and reRUN.

STEP 5 After your program is entered correctly, press any key (other than BREAK) to get out of the PAUSE 33333 command at line 120. What do you observe for the values of B and C printed on the screen? You should have B = 1 and C = 201.

STEP 6 What is the distinction between the 1s at addresses 16514/32130 and 16516/32132 and the 201s at addresses 16515/32131 and 16517/32133? Which are opcodes and which are operands?

STEP 7 ReRUN the program but choose different numbers for the values at addresses 16515/32131 and 16516/32132.

STEP 8 LIST the program. If you are using a B&W model, what do you observe in the REM statement? Look up the codes for the first four characters in the Appendix of your User's Manual. SAVE the BASIC program on cassette for the rest of the experiments in Chapter 3.

SUMMARY In addition to verifying that the USR function returns the value of the B and C registers, we have seen that a machine language program works like a subroutine and returns to the BASIC program on a RET instruction. The experiment also illustrates the Immediate Load register Transfer machine language instruction.

EXPERIMENT 3.2

MACHINE LANGUAGE ARITHMETIC AND LOGIC OPERATIONS

DISCUSSION We have said that all arithmetic and logic operations are performed on the Accumulator with any one of the other registers (including the Accumulator itself). Moreover, the various bits in the Flag register are altered as a result of these operations. Because we have seen that we can obtain the contents of registers B and C from the BASIC USR function, we need a scheme for transferring the A and F bytes to B and C so that we can observe the results of the math ops. The AF register pair can be placed on the Stack with a PUSH AF instruction. Once on the Stack, these bytes can be placed in BC with the POP BC instruction.

The ten Arithmetic and Logic operations found in the upper right box of Chart A.1 will be investigated. In particular, the Flag register bits will be examined and related to the operation executed. The Flag bits are:

F7	F6	F5	F4	F3	F2	F1	F0
MINUS	ZERO	NEGATE	X	HALF CARRY	X	EVEN PARITY/ OVERFLOW	CARRY

where X means the value of the bit is unassigned and indeterminate.

PROCEDURE

STEP 1 LOAD the BASIC program from Experiment 3.1 if it is not already in memory.

STEP 2 To display the binary values of B and C so that the individual bits can be examined, add the following lines to your BASIC program:

```
200 DIM B(8)
210 LET B$=" "
220 DIM C(8)
230 LET C$=" "
240 FOR J=8 TO 1 STEP -1
250 LET B(J)=INT (B/2**(J-1))
260 LET B=B-B(J)*(2**(J-1))
270 LET B$=B$+STR$ (B(J))
280 LET C(J)=INT (C/2**(J-1))
290 LET C=C-C(J)*(2**(J-1))
300 LET C$=C$+STR$ (C(J))
310 NEXT J
320 PRINT B$,C$
330 PRINT TAB 16; "MZ-----C "
```

This routine will take about 30 seconds to run. SAVE the BASIC program on cassette for use in subsequent experiments.

STEP 3 The first instruction we shall examine is the XOR A, which performs an exclusive OR on A with A. The subroutine we need is:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD A,N	62
16515 / 32131	<N>	255
16516 / 32132	XOR A	175
16517 / 32133	PUSH AF	245
16518 / 32134	POP BC	193
16519 / 32135	RET	201

RUN your program and ENTER the six codes of the subroutine as in Experiment 3.1. After you have compared your list for correctness, remember to press any key (except BREAK) to continue past line 120.

STEP 4 Make up a chart with headings:

	<16515/32131>	<B>	<C7>	<C6>	<C0>
OP	A(INITIAL)	A(FINAL)	MINUS	ZERO	CARRY
XOR	255	0	0	1	0
	127				

Your results should have been as shown in the first line where A(INITIAL) is the byte in the subroutine at address 16515/32131, A(FINAL) is the value of B printed on the screen, and the Flag bits are from the binary digits of C printed on the screen and underlined by M, Z, and C, respectively.

STEP 5 You can substitute other numbers into A by changing the operand on line 16515/32131. Rather than reloading the subroutine each time just to change one number, a new number can be POKEd directly by typing (without a line number)

```
POKE 16515,127 <ENTER>        (B&W)
POKE 32131,127 <ENTER>        (Color)
```

then rerun the program by ENTERing:

```
GOTO 60
```

After doing this, fill out your chart.

STEP 6 Repeat Step 5 several times, each time change the number to be loaded into A. What can you conclude about XOR A? You should observe that it always CLEARS the A register and sets the Flags for a value of zero.

STEP 7 Now change the operation in line 16516/32132 from XOR to ADD A,A by

```
POKE 16516,135 <ENTER>        (B&W)
POKE 32132,135 <ENTER>        (Color)
```

and

```
GOTO 60 <ENTER>
```

Try several numbers again as in Step 5.

STEP 8 Repeat Step 7 using other opcodes for the Arithmetic and Logic ops until you are satisfied you understand the results of each. Continue recording your results in your chart so you can review them as you progress. In particular, of what special value is the operation OR A? You should find that the flags are properly set for the number in the A register without altering the number. This is especially useful because the Transfer ops *do not* alter the flag bits.

STEP 9 To carry out these ops with two different registers, we need a new subroutine to load two registers. RUN your BASIC program, and enter the eight codes for the following subroutine:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD BC,NN	1
16515 / 32131	<N>	127
16516 / 32132	<N>	128

16517 / 32133	LD A,B	120
16518 / 32134	ADD A,C	129
16519 / 32135	PUSH AF	245
16520 / 32136	POP BC	193
16521 / 32137	RET	201

Write down your results in a chart having the headings:

	<16516/32132>	<16515/32131>	<B>	<C7>	<C6>	<C0>
OP	A(INITIAL)	C(INITIAL)	A(FINAL)	MINUS	ZERO	CARRY
ADD	128	127	255	1	0	0

You should have the results shown.

STEP 10 You can POKE different values for C and B into locations 16515/32131 and 16516/32132 and also change the operations in location 16518/32134 but remember to use the operations for the A,C registers.

EXPERIMENT 3.3
MACHINE LANGUAGE ROTATE OPERATIONS

DISCUSSION There are essentially two types of rotate operations: the eight-bit rotate in which the Carry bit is in parallel with either D7 or D0 (depending on direction of rotation), and the nine-bit rotate in which the Carry bit is in series with the eight bits of the register. The Carry bit is the only Flag affected by the Rotate and Shift instructions. As was pointed out earlier, the four rotates on Chart A.1 belong to the 8080 subset and are also repeated in the upper right box of Chart A.4 (A column) when they are used in the Z80 Augmented set and require the Prefix 230. The only difference between them is the time of execution. If you look on the Timing Table (Chart A.5) you will find that the prefixed Rotates take 8*t* and the 8080 subset Rotates take 4*t*.

PROCEDURE

STEP 1 Load the Basic program from Experiment 3.2 if it is not already in memory.

STEP 2 We can examine the 8080 rotates using almost the same subroutine we first used in Experiment 3.1 by loading an immediate operand into A and ORing it to set the Flags based on its value. This is because the Transfer ops do *not* affect the Flags but the Math ops do. RUN the BASIC program and load the following 7 codes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD A,N	62
16515 / 32131	<N>	129

16516 / 32132	OR A	183
16517 / 32133	RLCA	7
16518 / 32134	PUSH AF	245
16519 / 32135	POP BC	193
16520 / 32136	RET	201

STEP 3 Make up a table to record your observations using the headings:

	<16515/32131>	<B>	<C7>	<C6>	<C0>
OP	A(INITIAL)	A(FINAL)	MINUS	ZERO	CARRY
RCLA	129=10000001	00000011	1	0	1

Your results should be identical to the first line shown.

STEP 4 Change the rotate instruction to RRCA by

```
POKE 16517,15 <ENTER>        (B&W)
POKE 32133,15 <ENTER>        (Color)
GOTO 60 <ENTER>
```

and record your results.

STEP 5 Repeat Step 4 with the RLA and RRA operations POKEd into location 16517/32133. Make special note on the results of the Carry flag compared to the RLCA and RRCA ops. You might also want to change the initial A value in 16515/32131 to 128 and 1 to verify your results.

STEP 6 The Z80 Augmented Rotates can be studied with a subroutine that returns the B register and the Flags register. Note carefully the Transfer ops because these Rotates are *not* in the Accumulator but directly in the register itself (B for our case). RUN the BASIC program and enter the following 9 opcodes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE	
B&W / Color			
16514 / 32130	LD B,N	6	
16515 / 32131	<N>	129	
16516 / 32132	XOR A	175	(Clears A and F)
16517 / 32133	Prefix	203	
16518 / 32134	RLC B	0	
16519 / 32135	LD A,B	120	
16520 / 32136	PUSH AF	245	
16521 / 32137	POP BC	193	
16522 / 32138	RET	201	

STEP 7 Complete your table from Step 3 making enough changes in operations at 16518/32134 and values at 16515/32131 to complete your understanding. Remember to work from the B column of Chart A.5.

EXPERIMENT 3.4
INDIRECT LOAD MACHINE LANGUAGE INSTRUCTIONS

DISCUSSION The LOAD instruction transfers the contents of one register (source) to another register (destination). We have seen that there are two kinds of LOAD instructions: direct and indirect. Besides the microprocessor's registers, the source register can be any memory register. When the memory register is part of the program list, we have called the transfer an *immediate load* because the source register is the operand of the opcode. These transfer instructions are called *direct loads* because the source register is identified directly by name as either r(A, . . . ,L) or N. The *indirect load* is a transfer instruction where the source register is identified by the 16-bit register pair that holds (points to) the address of the source register. The mnemonics for indirect load are written with parentheses around the 16-bit address, such as LD A,(BC) and LD A,(NN) where the second load is an immediate indirect transfer and (NN) is a two-byte operand of the opcode holding the address whose contents (one byte) is to be loaded into register A.

One of the extra capabilities of the HL register pair compared to BC and DE is a two-byte transfer. The operations LD HL,(NN) and LD (NN),HL are immediate indirect transfers where L is loaded from, or loaded into, respectively, memory location NN. The microprocessor then increments NN to NN+1 and transfers the contents of that location from or to register H. There are six other opcodes of the form LD (HL),r and LD r,(HL). The latter are given as column M on the lower right table in Chart A.2.

PROCEDURE

STEP 1 Load the BASIC program from Experiment 3.1. If program 3.3 is already in the computer either delete lines 200–330 or ignore the additional printout.

STEP 2 For our indirect loads, we need to use memory locations that are protected from the BASIC interpreter. In the B&W models, these are the unused character spaces in the REM statement on line 1 of the BASIC program. Because 30 locations have been reserved, the highest memory register available is 16543. In the Color models, there are at least 638 locations CLEARed, therefore the unused bytes following the machine code up to 32767 are available.

STEP 3 If you are using a B&W model, then enter the following BASIC command:

```
PRINT PEEK 16543                    (B&W)
```

You should obtain 28(B&W) which is the character code for zero (the last character in the REM statement). Now enter:

```
          POKE 16542,37+128         (B&W)
and       POKE 16543,28+128
```

LIST your program. The last two characters in the REM statement should still appear as 9 and 0 but in inverse video.

If you are using a Color model, enter:

```
POKE 32158,47                       (Color)
```


and
then

```
POKE 32159,48
PRINT PEEK 32158; PEEK 32159
```

The numerals 9 and 0 should appear on your display.

STEP 4 We can write a subroutine to load these values into L and H using the LD HL,(NN) instruction. Of course, we will need to then transfer H and L to B and C if we are to have them returned by the USR function. For the B&W computer models, the address NN=16542 must be rewritten as two bytes or a base 256 number, equal to 64(MSBy) and 158(LSBy) because 64*256+158=16542. For the color computer models, the address NN=32158 equals 125(MSBy) and 158(LSBy).

STEP 5 RUN the BASIC program, and enter the following six codes.

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD HL,(NN)	42
16515 / 32131	<Lo N>	158
16516 /	<Hi N>	64
/ 32132	<Hi N>	125
16517 / 32133	PUSH HL	229
16518 / 32134	POP BC	193
16519 / 32135	RET	201

B and C should return with the values we POKEd in Step 3.

STEP 6 The rest of the INDIRECT transfers are one byte, or single register, loads. To use the (HL) or M opcodes, we will keep using location 16542/32158. RUN the BASIC program, and ENTER the following seven codes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD HL,NN	33
16515 / 32131	<Lo N>	158
16516 /	<Hi N>	64
/ 32132	<Hi N>	125
16517 / 32133	LD B,(HL)	70
16518 / 32134	INC HL	35
16519 / 32135	LD C,(HL)	78
16520 / 32136	RET	201

B and C should return with the values you found in Step 5 except reversed.

STEP 7 To transfer data from one of the microprocessor's registers to a memory register, both the destination and address and the contents of the byte to be transferred have to be loaded directly in the subroutine. RUN the BASIC program, and ENTER the following seven codes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD BC,NN	1
16515 / 32131	<Lo N>	158
16516 /	<Hi N>	64
/ 32132	<Hi N>	125
16517 / 32133	LD A,(BC)	10
16518 / 32134	INC BC	3
16519 / 32135	LD (BC),A	2
16520 / 32136	RET	201

What values for B and C should be printed on the screen? This will be address 16543/32159. If you LIST the B&W program, the two last characters in the REM statement should now both be the number 9 in inverse video.

EXPERIMENT 3.5
ABSOLUTE BRANCH INSTRUCTIONS

DISCUSSION The branch opcodes of the 8080 subset include unconditional and conditional Jumps, Calls, and Returns. Because the Timex/Sinclair USR function implements the unconditional CALL, we have in essence been performing the three-byte CALL instruction by providing the 16-bit destination address as the argument of the USR function in BASIC. Of course, all the subroutines have also used the unconditional Return opcode as well. We shall examine the absolute JUMP instructions with the understanding that subroutine Calls and Returns operate in a similar manner.

PROCEDURE

STEP 1 Load the BASIC program used in Experiment 3.2.

STEP 2 RUN the program, and enter the following 16-byte subroutine:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD A,N	62
16515 / 32131	<N>	1

16516 / 32132	ADD A,N	198
16517 / 32133	<N>	128
16518 / 32134	JP NZ,NN	194
16519 / 32135	Lo Addr.	140
16520 /	Hi Addr.	64
/ 32136	Hi Addr.	125
16521 / 32137	PUSH AF	245
16522 / 32138	POP BC	193
16523 / 32139	RET	201
16524 / 32140	PUSH AF	245
16525 / 32141	LD A,N	62
16526 / 32142	<N>	85
16527 / 32143	POP BC	193
16528 / 32144	LD B,A	71
16529 / 32145	RET	201

Note that the routine will return A=85 (alternate 0s and 1s in binary) if it makes the jump, otherwise the value will be A(Initial)+128. The Flags will have the same values irrespective of whether or not the jump was made and will reflect the conditions at the time of the jump decision; that is, on the sum of A(Initial)+128. Make a table such as the following to record your results. The first entry should be as shown:

	<16515/32131>	<B>	<C>		
OP	A(INITIAL)	A(FINAL)	M	Z	C
JP NZ,	1	85	1	0	0

STEP 3 Using the same branch operation:

```
                POKE 16515,255        (B&W)
                POKE 32131,255        (Color)
and             GOTO 60
then            POKE 16515,128        (B&W)
                POKE 32131,128        (Color)
and             GOTO 60
```

Record your results for each case.

STEP 4 Now change the branch operation of JP Z,NN by

```
POKE 16518,202        (B&W)
POKE 32134,202        (Color)
```

and repeat the three values for A(Initial) as in Steps 2 and 3.

STEP 5 Repeat Step 4 using JP NC,NN (210); followed by JP C,NN (218), JP P,NN (242); and JP M,NN (250). When you have finished, your table should have all the information to verify the following summary table:

BRANCH DECISION FOR A = N + 128:

		N = 1	255	128
		A = 129	127	0
OPS:	JP NZ	Yes	Yes	No
	JP Z	No	No	Yes
	JP NC	Yes	No	No
	JP C	No	Yes	Yes
	JP M	Yes	No	No
	JP P	No	Yes	Yes

STEP 6 In the table given above, you will see that the NC/C and M/P results are identical. The only way they could be made to differ on an ADD instruction is by obtaining a result in A(Final) which would set the Zero flag and not set the Carry flag. This condition is not possible for an ADD operation if A(Initial) is nonzero. If you wanted to see a distinction between JP C,NN and JP P,NN, how would you modify the subroutine? Try replacing ADD A,N with SUB A,N and repeating the two jump instructions for the three different values of A(Initial) used previously.

EXPERIMENT 3.6
RELATIVE BRANCH INSTRUCTIONS

DISCUSSION If you examine the relative jumps for the Z80 Augmented instructions, you will note that there are four unconditional jumps based only on two flags: Zero and Carry. Because they use a one-byte operand, the relative jumps (including the unconditional jump, JR d, and the decrementing jump, DJNZ) can only jump forward up to 127 locations and up to 128 locations backwards. Recall that the one-byte operand is interpreted as a two's complement number and that the displacement is relative to the Program Counter. At the time of the branch decision, the Program Counter equals the address of the byte following the operand (or two more than the jump opcode).

PROCEDURE

STEP 1 Load the BASIC program used in Experiment 3.5.

STEP 2 We can write a subroutine similar to the one in Experiment 3.5 but use the conditional relative jump instruction instead of the corresponding absolute jump. Because this will be a

forward jump to a higher memory address, it will illustrate a positive operand. To illustrate a jump backwards, we will substitute the unconditional relative jump for the second RET and jump back to the first return instruction. RUN the BASIC program and load the following 16 codes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD A,N	62
16515 / 32131	<N>	1
16516 / 32132	ADD A,N	198
16517 / 32133	<N>	128
16518 / 32134	JR NZ,d	32
16519 / 32135	<d>	3
16520 / 32136	PUSH AF	245
16521 / 32137	POP BC	193
16522 / 32138	RET	201
16523 / 32139	PUSH AF	245
16524 / 32140	LD A,N	62
16525 / 32141	<N>	85
16526 / 32142	POP BC	193
16527 / 32143	LD B,A	71
16528 / 32144	JR d	24
16529 / 32145	<d>	248

STEP 3 Before pressing any key to get past the PAUSE command, verify the displacement operands at addresses 16519/32135 and 16529/32145. In the first case, because address 16520/32136 is the zero displacement and we wish to jump to 16523/32139, the operand value should be 3. In the second case, 16530/32146 is zero reference, 16529/32145 is −1 or 255, 16528/32144 is −2 or 254, then the RET at 16522/32138 is −8 or 248. Note that the sum of the absolute values of the negative number and its two's complement is always 256.

STEP 4 Your results for this subroutine should be the same as those obtained in Experiment 3.5.

STEP 5 As we noted earlier in this chapter, the DJNZ instruction uses register B as a countdown register. The most typical applications of this instruction are similar to the NEXT command in BASIC, where a preceding LD B,N instruction would correspond to the BASIC command: FOR V = N to 0 STEP −1. We want to verify that B is zero on return from the subroutine and that the loop was executed B(Initial) times. LOAD the BASIC program from Experiment 3.1, or delete line 330 from the BASIC program used in the last experiment because register C will not be loaded from the F register.

STEP 6 RUN the BASIC program and ENTER the following 7 codes:

ADDRESS	INSTRUCTION MNEMONIC	DECIMAL CODE
B&W / Color		
16514 / 32130	LD BC,NN	1
16515 / 32131	<C>	0
16516 / 32132	<B>	255
16517 / 32133	INC C	12
16518 / 32134	DJNZ d	16
16519 / 32135	<d>	253
16520 / 32136	RET	201

Your results should be B=0 and C=255.

STEP 7 Calculate the time delay involved in the DJNZ loop by finding the number of clock cycles each instruction in the loop uses from Chart A.3. The INC C instruction takes 4*t* and the DJNZ d instruction takes 8*t* for all but the last time it is executed. Therefore, a total of 12*t* times 255 clock cycles are used in the loop. For a *t* of 300 nanoseconds, the time delay is 12*255*0.3 or 920 microseconds, which is approximately 1 millisecond (0.001 seconds).

SUMMARY These experiments have illustrated most of the types of machine language instructions of the 8080 subset and some of those from the Z80 Augmented set. The BASIC USR function is a very useful command and permits the use of machine language subroutines to achieve executions about 1000 times faster than the BASIC interpreter. Other subroutines can be written to demonstrate those instructions that have not been covered. It should also be noted that as many subroutines as desired can be called from a BASIC program simply by changing the argument (destination address) of the USR function.

□ 4 □

input and output ports

DEVICE SELECT PULSES

At the end of Chapter 3 we saw that there are two machine codes that are used to transfer a byte between the A register and an external device. The opcode mnemonics are IN A,(N) and OUT (N),A. These are two-byte instructions whose one-byte operand, (N), is an indirect reference to the device address. The device address is an eight-bit code which appears on the Low Address bus, A7-A0, during the execution of the instruction. Because the device code is eight bits, it is possible to have a maximum of 256 input device addresses and 256 output device addresses. Another name for a device is *port* and another name for a device address is *port address*. A peripheral device, such as an instrument for measurement or control, or a support device, such as a printer, may require more than one port in order to operate. For example, the printer is an output device, but there may be conditions that the computer must know before it transfers the code of a character to be printed such as: Is the printer on? Is the printer out of paper? Is it busy still printing the previous character? Notice that each of these conditions is binary (yes or no) and therefore each needs only one bit to inform the computer of its condition. This type of I/O interaction is referred to as *handshaking*. Up to eight conditions could form one byte, which would be transferred INto the computer before it tried to OUTput the next character to be printed. Thus, a printer as a peripheral device might have an input port as well as an output port.

We saw in Chapter 2 that eight bits, such as a device (port) address, can be decoded so that only one output channel of a decoder is uniquely activated. This allows the conversion of eight simultaneous signals in parallel on the Address Bus into a single pulse which is active ONLY when a specific (1 out of 256) eight-bit code is present. We shall refer to this single pulse as a *Device Code*. We can illustrate a one-channel decoder with the 74LS30 eight-Input NAND gate shown in Figure 4.1. In this figure, four inverters of a 74LS04 IC are used in conjunction with a 74LS30 eight-input NAND gate to decode the low address bus. Because the unique state of a NAND gate has a 0 output only when all inputs are in a logic 1 state, then the Device Address 43 (decimal) will be the only possible combination of 1s and 0s on the low address bus that will cause the output of Device Code 43*. Note that the asterisk indicates that the device

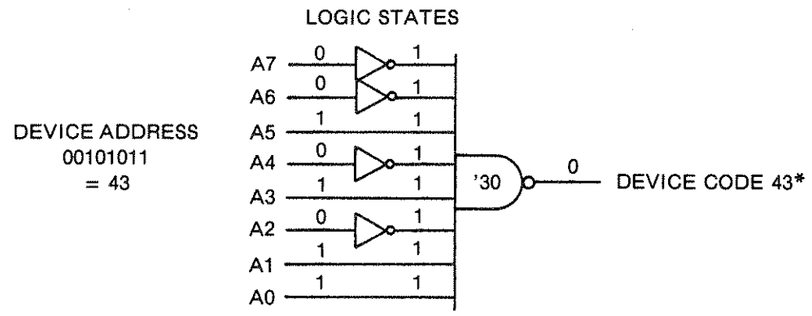


Figure 4.1 One-Channel Decoder.

code is active low (a logic 0). For all 255 other addresses on lines A7–A0, the output of the NAND gate will remain at a logic 1.

This scheme is perfectly adequate if an interface design requires only one device code. If our interface design needs to create more than one port address then we can choose one of the multichannel decoders such as the 8-channel 74LS138 studied in Experiment 2.6 or the 16-channel 74154 described in Chapter 2. Whatever the decoding scheme used, it answers the question of *where*? concerning the data transfer, but leaves unanswered the questions of *when*? and *how*? Recall that earlier in Chapter 3 we saw that there are control signals generated by the Z80 microprocessor which can be combined with OR gates to generate the control I/O signals IN* and OUT*. These I/O control signals answer the questions of *when* and *how*. If we are to obtain a single signal which will activate a particular device, then a logical combination of the device code and the I/O control pulse must be produced. This final result is the Device Select Pulse (DSP). Because the IN* and OUT* control pulses are active low and most decoders also produce an active low channel output, the most common method of creating the Device Select Pulse, using the unique state of a gate, is to OR (for an active low DSP) or NOR (for an active high DSP) the device code with the I/O control pulse. The choice made depends on the logic requirements of the device itself. Each device code can be combined with both I/O control pulses as shown in Figure 4.2. In the figure, different gates are shown to illustrate opposite active states for the Device Select Pulses: IN 43* (active low) and OUT 43 (active high). Note that it is impossible for the I/O control pulses to occur at the same time because the control lines RD* and WR* cannot be generated during the same machine cycle of an instruction.

The concept of Device Select Pulse generation is one of the most important concepts of interfacing. We can summarize with the following statements:

1 The machine language instructions IN A,(N) and OUT (N),A transfer a data byte from and to, respectively, a port having a one-byte address, N.

2 To form the Device Select Pulse that activates a particular port, the eight bits of the port address occurring on the Low Address bus must be decoded to form a unique Device Code Pulse.
3 Depending on the direction of transfer, the Device Code must be logically combined with either IN* or OUT* control pulses to form the Device Select Pulse.

INPUT PORTS

An input port is a peripheral device, which, when activated by its Device Select Pulse, transfers a byte onto the Data Bus. The microprocessor accepts the data byte and loads it into the Accumulator register. Implied in this process is the condition that the output lines of the input port in question and those of all other input ports are connected in parallel to the data bus. We saw in Chapter 2 that the outputs of digital devices cannot be connected together unless they are connected through three-state buffers. Also implied in the data transfer from an input port is that there is valid data available at the port when the microprocessor executes the IN A,(N) opcode. Typically, the peripheral device and the microcomputer act independently of each other—each carrying out its own functions without regard to the other—until the moment of transfer. This type of operation is referred to as *asynchronous*. In order to preserve the data byte generated by the input port, a set of eight latches must be incorporated in the circuitry of the input port. The three main components of an input port are, therefore, the data source or generator, a data register which may be gated (enabled) by the data generator, and a three-state buffer. Figure 4.3 shows the block diagram of a generalized input port.

The data source is a general device that generates a data byte; it may be as simple as a set of eight switches each of which selects a logic 0 or 1 state. More advanced data generators will be described in Chapters 5 and 6. The register is not necessarily separate from the data generator. The point of including it as a separate component of an input port is to emphasize that the data must be held long enough to be acquired by the microcomputer. (Of course, a set of switches serves as its own register.) The Ready

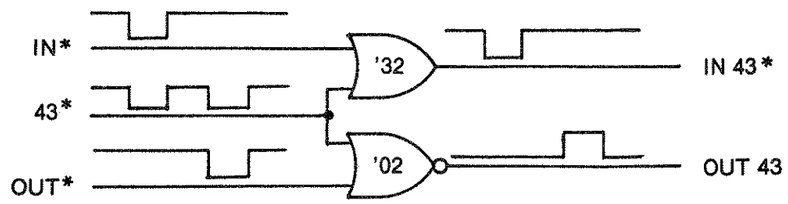


Figure 4.2 Device Select Pulses.

(RDY) output of the data generator is called a *Flag*. Data generators usually have such a control line to be used to indicate the presence of valid data. It may be used as indicated in Figure 4.3, or it might be used as a separate input port (one bit) for the computer to determine the presence of valid data. The three-state buffer is an essential element on any (and all!) input port. If it is already incorporated into the data generator, an additional three-state buffer is not necessary. This would be determined by the specifications of the particular data generator. Figure 4.3 also shows schematically how the Device Select Pulse is created. The connections to the Address, Data, and Control Buses of the computer shown are typical for every input port. Further refinements to an input port, in addition to the previously mentioned second input port for the Ready flag, might include an output port to clear the register and/or trigger the data generator to start another generator cycle.

OUTPUT PORTS

As opposed to input ports, output ports are passive. There is no need to isolate output ports from the data bus with three-state buffers because they do not attempt to load the data bus with information; that is, they are receivers not transmitters. The important point with output ports is that each requires a register to "latch" and hold the data byte intended for it. (Recall that the data on the Data Bus is valid only for a few microseconds at best.) The need for a Device Select Pulse as the gate enable control signal to the output port's latch should be apparent. Figure 4.4 shows the block diagram of a generalized output port. As in the case of the input port, the output port is

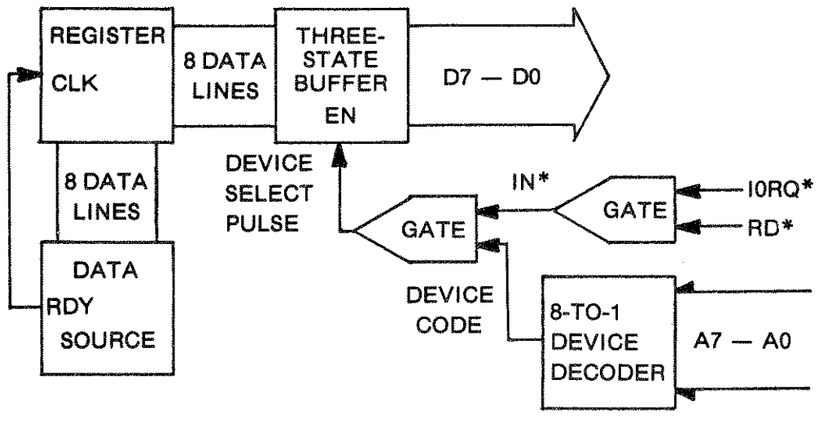


Figure 4.3 General Input Port.

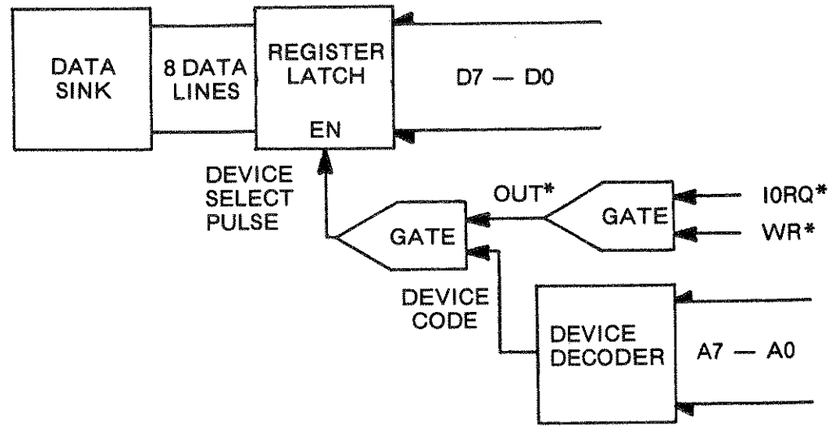


Figure 4.4 General Ouput Port.

activated by a unique Device Select Pulse created from the Device Code and a Control Pulse. The control pulse, OUT*, is generated in turn from the microprocessor signals IORQ* and WR* on execution of the instruction OUT (N),A. The data byte transferred from the computer's Accumulator register to the device whose address code, N, is the operand of the instruction in the machine language program. Once the data byte being transferred is latched into the device's register, the outputs of the register will hold the byte until the next OUT instruction to this device address is executed.

The sequence of events for output (or input) can be put into perspective by examining the timing diagram of the process. This is shown in Figure 4.5. The first six lines in Figure 4.5 are the various control signals from the Z80 microprocessor. The Clock input, ϕ (Phi), is the fundamental timing control and operates at a frequency of 3.25 MHz in the Timex/Sinclair. M1 is the control output that signals the start of a new instruction: this is always the fetch operation, which places the program counter, PC, on the address bus to read the next opcode from memory. The memory read, MREQ* and RD*, puts the opcode on the Data Bus. When the opcode is the OUT (N),A instruction, the next operation is a second memory read to obtain the operand from memory. The third operation in sequence is to place the operand value, which is the device address code, on the low Address Bus and the value stored in the A register on the Data Bus. After one clock cycle these buses are stable, then the control signals IORQ* and WR* are activated. The entire process takes 11 clock cycles in accordance with the t value given in Chart A3. In this type of timing diagram, the individual Address and Data Bus lines are grouped together and show only when their respective signals change. The values of the bus lines for each change are indicated by the labels

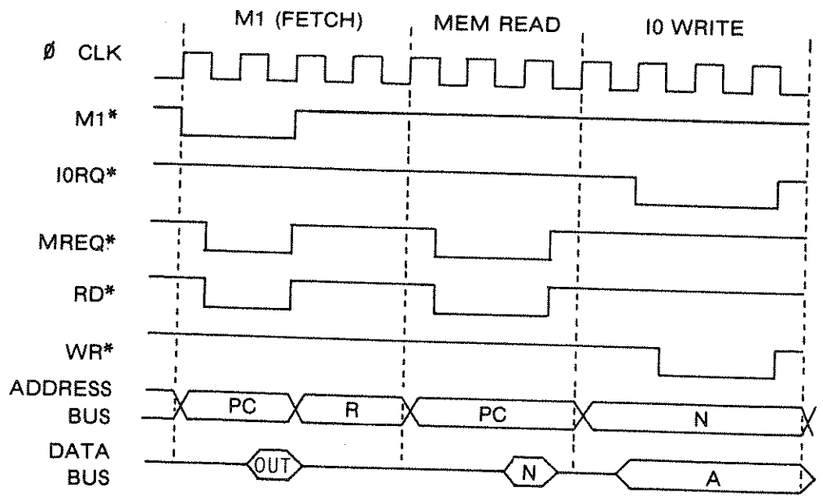


Figure 4.5 Output Timing Diagram.

shown, such as PC and N on the address bus and OUT, N, and A on the data bus. The timing diagram for an IN A,(N) instruction would be similar except the RD* control line would be activated rather than the WR* line. In either case, the actual transfer takes place in the time interval when the OUT* or IN* pulse is low.

THE T/S INTERFACE CIRCUIT

We are now in position to understand the requirements for constructing an interface circuit for a Z80 microcomputer such as the Timex/Sinclair. To perform Input/Output we need the eight data bus lines, D7-D0, and the eight lines of the low address bus, A7-A0. The Z80 control signals needed are input/output request (IORQ*), read (RD*), and write (WR*), which will be gated to create IN* and OUT*. Because the drive capability (fan out) of these signals is limited to less than 2 milliamps, it should be apparent that the first consideration should be to buffer them to increase their current drive. In bringing these buffered signals out of the computer so that they may be readily available for experimental interfacing, it would be convenient to also have available a selection of Device Codes. With these available, it would not be necessary to decode the address bus each time we wished to construct an input or output port.

The circuit for a Buffered I/O Interface is shown in Figure 4.6. Five integrated circuits are used. The 74LS244 is a three-state Octal Buffer used to increase the drive of

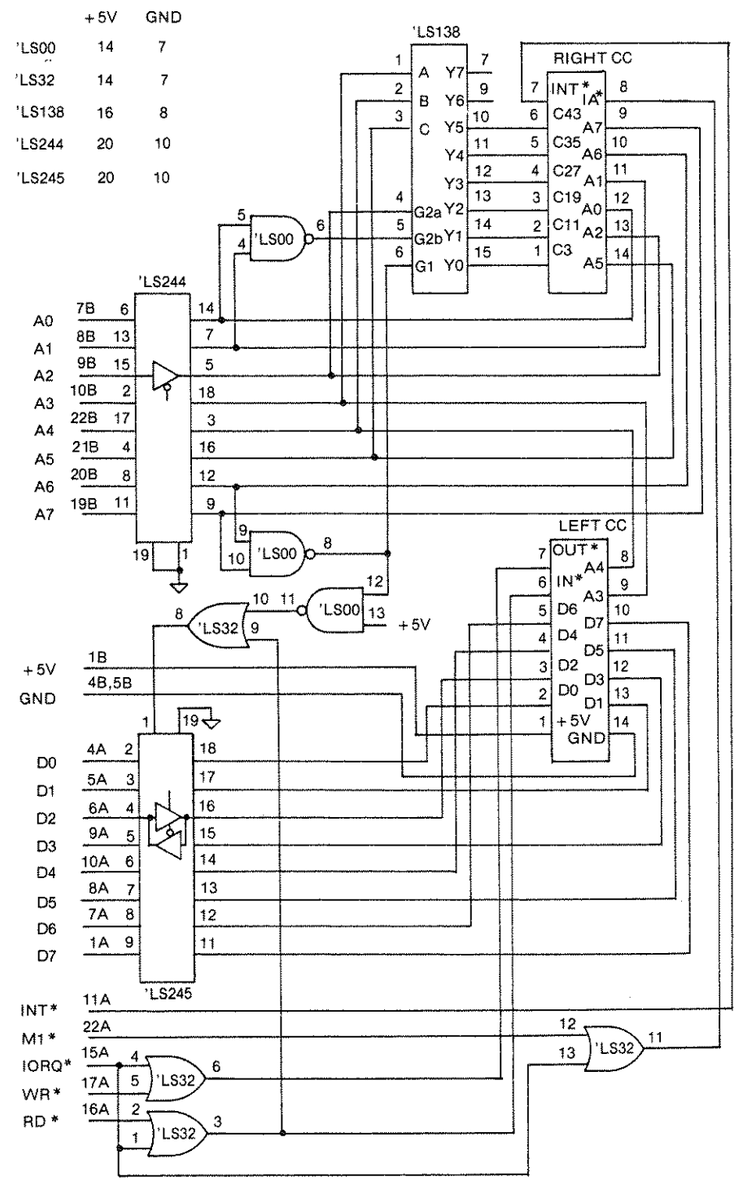


Figure 4.6 I/O Interface Circuit.

the eight low address lines. Because its three-state capability is not required, the enable inputs at pins 1 and 19 are permanently enabled by grounding to a logic 0 state. The 74LS245 is an Octal Bidirectional Bus Driver IC. As indicated, each bus line is equipped with two three-state buffers in opposition. The eight buffers in one direction are enabled by logic 1 while the eight buffers in the opposite direction are enabled by a logic 0. The direction control is located on pin 1. This IC is completely disabled for either direction by a logic 1 on pin 19. Because no advantage is gained by disabling the IC, pin 19 is permanently grounded to a logic 0. The problem faced with interfacing to the Bidirectional Data Bus is when to change its direction. As we noted with output ports, data that is output is passive and causes no conflicts with other components of the computer. It is when the Data Bus is to be "turned around" and input into the computer from the outside that care must be taken. The problem that has to be solved is how to avoid conflicts due to constraints imposed by the computer's hardware and software. In particular, the Timex/Sinclair uses several device codes for input and output in controlling the video, cassette recorder, and the special Sinclair Logic IC. The real problem, however, is that the device codes are only decoded using address lines A0, A1, and A2. Therefore, any device code of the binary form XXXXXABC will activate the Sinclair port that is coded for ABC no matter what the value of XXXXX! This type of decoding is called *relative* as opposed to the absolute decoding we have described previously. The decimal device codes listed in the operating system program in ROM are 251, 253, 254, and 255. Their corresponding octal values are 373, 375, 376, and 377, respectively. The advantage of listing their octal values is to show that only the three bits of the least significant octal digit vary. As a consequence, for example, all device codes 006, 016, . . . , 366 will also activate the device whose code is 376. Because the Timex/Sinclair software always outputs a 3 for the most significant octal digit on the low Address Bus for its internal devices, our solution is to turn the Data Bus buffers around for input only when the IN A,(N) is executed for devices not having an octal 3XX address code; in other words, only devices having N = 0XX, 1XX, or 2XX octal codes can change the direction of the Data Bus buffers of the 74LS245. These addresses in decimal will all be less than 192. The NAND gate with inputs of A6 and A7 will have a logic 0 output only when device addresses of 192 or greater occur on the low Address Bus. The second NAND gate acting as an inverter to the first NAND gate keeps the OR gate output at logic 1, which in turn keeps the Data Bus in the output direction. For values less than 192 on the low Address Bus, the Data Bus buffer OR gate will be a logic 0 only when the IN* control signal is also a logic 0. Therefore, only IN A,(N) with N less than 192 will reverse the Data Bus buffer.

Two OR gates are used in the Interface circuit to create the IN* and OUT* control pulses from IORQ* and RD* and WR* as was previously discussed. The fourth OR gate of the 74LS32 IC is used to create a response control pulse from the microprocessor when it receives an interrupt request. We shall defer discussion of interrupt operations to Chapter 7 except to note that the Z80 uses the IORQ* line during an M1 (the first machine cycle of an instruction) as the indication it has received an interrupt request. Recall that an M1 cycle is always a read from memory (MREQ) operation (opcode fetch) and would never generate a IORQ pulse otherwise. The

Interrupt request control is an input line to the microprocessor and does not need buffering. It is taken directly from the computer's edge connector to the Interface's cable connector.

The remaining circuitry of the Interface is not necessary but, as we mentioned earlier, is desirable. This is the Device Code generator implemented with a 74LS138 three-to-eight Line Decoder. Although the 74LS138 has only three data inputs, it also has three gating inputs all of which must be in their active logic state for the IC to decode its data inputs. One gate, G1, is active high, and the other two gates, G2A* and G2B*, are active low. By using the outputs of two NAND gates to combine two pairs of address lines, A7 with A6 and A1 with A0, we can obtain six address inputs for the 74LS138. The NAND gate output that combines A7 with A6 is the same as described for the Data Bus direction control. In this way, the 74LS138 can never decode addresses above 191 because they could never operate as input ports because of the direction control on the Data Bus buffer. The NAND gate that combines address lines A1 and A0 is connected to gate G2B*. For G2B* to be enabled by the NAND gate, it follows that A1 and A0 must both be in a logic 1 state. Address line A2 is connected to gate G2A*. The remaining address lines, A3, A4, and A5, are used as the data input to activate one of eight channels at the outputs of the IC. The active channel output goes to a logic 0 when selected with the rest of the channel outputs at a logic 1. The decoding therefore yields octal device codes 0N3, 1N3, and 2N3 at each channel where N is the three-bit value (0 to 7) of address lines A3–A5. This means that this decoding is also relative. These values in decimal are shown in Table 4.1.

Channels 6 and 7 are not shown in Table 4.1 because only the first six channels are brought out to the cable connector. Note that each row differs from the next row by 8 (corresponding to the middle octal digit) and that each column differs by 64 (corresponding to the most significant octal digit). The least significant octal digit is 3 in all cases. The value of 3 was chosen because the only other values possible, using one NAND gate to combine two of the three lowest address bits to gate G2B* and the third bit to gate G2A*, were 5 and 6. Either of these choices would conflict with the internal device codes, whereas 3 is the Sinclair Printer device code (for both output and input for handshaking) and cannot cause conflict. The I/O Interface has been "piggy-backed" onto the Sinclair Printer interface and both operated without any problems.

TABLE 4.1 I/O INTERFACE DECIMAL DEVICE CODES

CHANNEL	DEVICE CODES		
(A5,A4,A3)	(A7,A6): 0	1	2
0	3	67	131
1	11	75	139
2	19	83	147
3	27	91	155
4	35	99	163
5	43	107	171

The I/O Interface circuit can be constructed on a 3″ × 3½″ predrilled board having holes drilled on 0.10″ centers in both directions and using wirewrap sockets to mount the ICs and cables. A wirewrap PC edge connector is also mounted but must be cut from a larger (more than 23 pairs of contacts) unit to allow for open ends to fit onto the computer board. The list of components for the Interface board and the experiments is given in Appendix B. The authors found the wirewrapped unit adequate; however, a fair amount of interference caused more video picture noise than desirable. A printed circuit board that eliminates this problem is available. Addresses of suppliers for the board and other components are given in Appendix C.

EXPERIMENT 4.1
PULSE STRETCHING AND BUS ACTIVITY

COMPONENTS 1 * 74121 Monostable
1 * Lamp monitor
1 * 22-Kohm resistor
1 * 1.0-μF capacitor

DISCUSSION The 74LS138 three-to-eight Line Decoder in the I/O Interface circuit relatively decodes the eight bits of the low Address Bus at all times. We saw in Table 4.1 that each channel is activated by three different addresses. It is important to understand that the decoder does not distinguish between memory references and I/O device references. Because the low Address Bus lines are always changing as the computer program sequences through memory, every address from the program counter that matches the decoder channels will activate that channel output. Because each low address is accessed by the program on the average of one time in 256, then we would expect to see practically continuous activity on each channel. However, each pulse is too short to be seen by an LED probe unless its duration can be "stretched" to a period long enough to see: roughly from the approximate 1-microsecond pulse to about 10 milliseconds, or 10,000 times.

A monostable integrated circuit such as the 74121 can be used to monitor pulse activity. Using the pulse to be observed as a trigger, the monostable IC generates a pulse of duration determined by an external resistor and capacitor connected to two pins of the IC. The product of units of resistance in ohms and capacitance in farads is a time constant in units of seconds. The digital circuit is called a Monostable because its output is stable only in one logic state; that is, when it is triggered, its output Q goes into an unstable state of logic 1 for a determined period and then returns to its stable logic 0 state. Monostable ICs typically have complementary outputs Q and Q*.

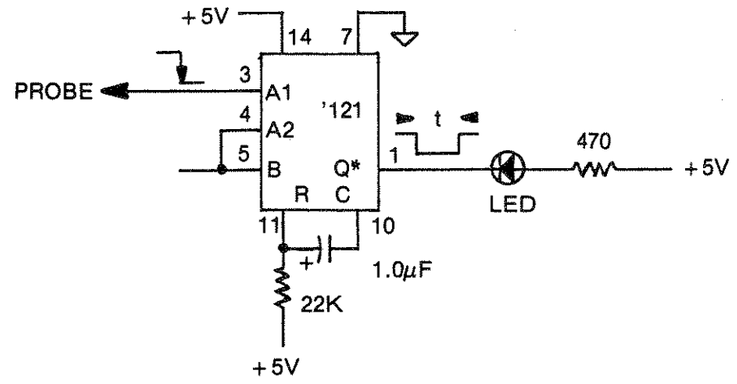


Figure 4.7 Experiment 4.1 Schematic.

PROCEDURE

STEP 1 Wire the 74121 circuit as shown in the Schematic. The period for the monostable pulse in milliseconds is 0.7 * R(Kohms) * C(microfarads). Using a 22-Kohm resistor, a capacitor between 0.1 μF and 4.7 μF yields a visible flash on the LED. The 74121 is not made in an LS version; however, there is sufficient power to support the circuit using the regular TTL IC.

STEP 2 Insert the probe from pin 3 into the ground (0 V) rail while observing the LED. Because the unconnected pin 3 floats to a logic 1 before grounding, inserting the probe into the 0-V rail creates a negative edge, which triggers the monostable. You should have observed the LED flash.

STEP 3 Now remove the probe from the 0-V rail. Did the LED flash? Probably. The pulse should be a positive edge as pin 3 floats from 0 to +5 V, and therefore should not trigger the monostable. However, the mechanical bounce as the wire was withdrawn caused a triggering negative edge.

STEP 4 Rather than inserting the pin 3 probe into the 0-V rail, just touch it to one of the 0-V rail sockets. Every time you make contact the LED should flash. Sometimes when you break contact it may not; this will occur whenever the bounce is shorter than the monostable period.

STEP 5 Successively insert the probe into the cable sockets labeled C3, C11, . . . , C43. In each case you should observe that the LED appears to remain on. This indicates constant activity on the address bus.

STEP 6 Probe the IN* and OUT* cable sockets. Remember that the Timex/Sinclair uses I/O devices and therefore these signals are also active.

STEP 7 Leave the monostable circuit wired for Experiment 4.2.

EXPERIMENT 4.2
DEVICE SELECT PULSES

COMPONENTS
1 * 74LS02 Quad Two-Input NOR Gate
1 * 74LS32 Quad Two-Input OR Gate
From Experiment 4.1:
1 * 74121 Monostable
1 * 22-Kohm resistor
1 * 1.0-μF capacitor
1 * Lamp Monitor

DISCUSSION In this experiment we shall examine the uniqueness of a Device Select Pulse. We saw in Experiment 4.1 that the device decoder has constant activity on its channel outputs and that the control signals are also very active. Actually, all we could verify is that pulses on these lines occur at least once every 15 to 30 milliseconds, otherwise we would have observed the LED flicker. To determine whether a channel pulse from the decoder and one of the control pulses (IN* or OUT*) occur simultaneously, we will have to gate the two signals and examine the output of the gate. This output signal will be a Device Select Pulse. We can create a DSP by programming the BASIC USR function and observe the result as a flash on the LED.

PROCEDURE

STEP 1 If you have not already done so, wire the 74121 circuit from Experiment 4.1.

STEP 2 Mount a 74LS02 on the breadboard, and connect pin 14 to +5 V and pin 7 to 0 V.

STEP 3 Complete the NOR connections to the cable pins of OUT* and C3* and to the Monostable as shown in Figure 4.8. The output at pin 4 is the DSP OUT C3. The output at pin 6 after inversion is OUT C3*.

STEP 4 Load the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
20 LET L = USR 16514       (for B&W models)
20 LET L = USR 32130       (for Color models)
```

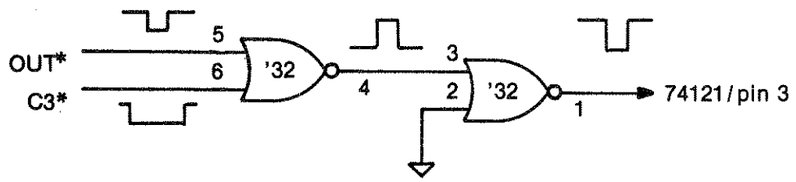


Figure 4.8 DSP "OUT C3".*

STEP 5 Enter the following machine language subroutine:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	211	OUT (N),A
16515 / 32131	3	(N)
16516 / 32132	201	RET

by ENTERing each of the following direct POKE commands:

```
POKE 16514,211          (B&W)
POKE 16515,3
POKE 16516,201
POKE 32130,211          (Color)
POKE 32131,3
POKE 32132,201
```

STEP 6 Observe the LED as you press RUN and ENTER. You should see the LED flash. Repeat a few times.

STEP 7 Now change the device code in the subroutine to 3 + 64 = 67 by ENTERing:

```
POKE 16515,67           (B&W)
POKE 32131,67           (Color)
```

RUN the program again. Did you see the LED flash? You should have because C3* is relatively decoded with a modulus of 64.

STEP 8 Repeat Step 7 only change the device code to 3 + 64 + 64 = 131.

STEP 9 Repeat Step 7 once more, this time using 3 + 64 + 64 + 64 = 195. The LED should not have flashed because the I/O Interface decoder does not decode addresses greater than 191 where both A6 and A7 are logic 1s.

STEP 10 Mount a 74LS32 on the breadboard next to the NOR gate IC. Connect the power pins: +5 V at pin 14 and 0 V at pin 7. Wire the OR and (rewire) the NOR gates according to the schematic shown in Figure 4.9.

STEP 11 Connect pin 1 of the NOR gate (74LS02), OUT 3, to the Monostable probe, pin 3 of the 74121. ENTER the following commands:

```
POKE 16515,3            (B&W)
POKE 32131,3            (Color)
```

and

```
RUN.
```

You should observe the LED flash for the absolute Device Select Pulse OUT 3. Move the Monostable probe to NOR gate pin 1, OUT 67, and RUN. No flash should be observed.

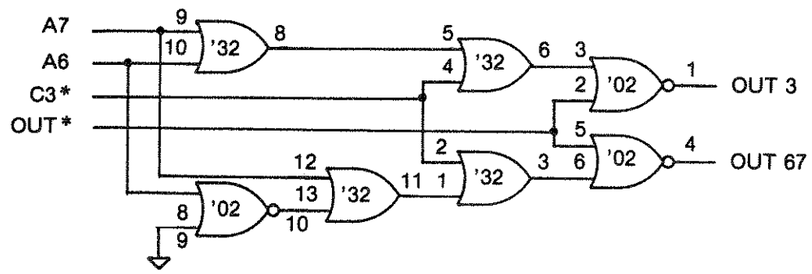


Figure 4.9 Absolute Decoding of Channel 3.

STEP 12 Verify the following Pulse Table by changing device codes in the subroutine and alternately probing pins 1 and 4 of the NOR gates.

DEVICE CODES	NOR GATE OUTPUTS	
<16515/32131>	Pin 1	Pin 4
3	PULSE	None
67	None	PULSE
131	None	None

STEP 13 Exchange the wire connections at A7 and A6 of the cable connector. What DSPs are now available at NOR outputs 1 and 4? Your answer should be OUT 3 and OUT 131 respectively. Can you make another Pulse Table? These last four steps illustrate absolute decoding. In all subsequent experiments, we shall confine our device codes to the six codes below address 64 and thereby avoid port conflict due to the relative decoding of the I/O Interface circuit. If you need more than six port addresses, then by proper absolute decoding as in this experiment, you can have up to 18 available device codes from the I/O Interface.

EXPERIMENT 4.3
DEVICE SELECT PULSES FOR DIGITAL CONTROL

COMPONENTS 1 * 74LS74 Dual D latch
1 * 74LS32 Quad Two-input OR gate
1 * Lamp monitor

DISCUSSION Although most of our interest in interfacing is generally related to data acquisition, the DSP can also function as a control pulse for some external device. Even though the instructions IN A,(N) and OUT (N),A involve transfer of a data byte between a port and the Accumulator, we may choose to ignore the data byte and rely only on the uniqueness of the DSP

to perform some operation. The DSP can be used to activate some digital device such as a relay to turn on or turn off a high voltage lamp or motor or any other ON/OFF device.

PROCEDURE

STEP 1 We can illustrate any ON/OFF device by using a D latch having both PReset and CLear control inputs, such as the 74LS74 IC. Wire the circuit as shown in the schematic, Figure 4.10.

STEP 2 Load the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
20 LET L = USR 16514       (for B&W models)
20 LET L = USR 32130       (for Color models)
30 PRINT "OFF" ;
40 PAUSE 33333
50 LET L = USR 16520       (for B&W models)
50 LET L = USR 32136       (for Color models)
60 PRINT "ON " ;
70 PAUSE 33333
80 GOTO 20
```

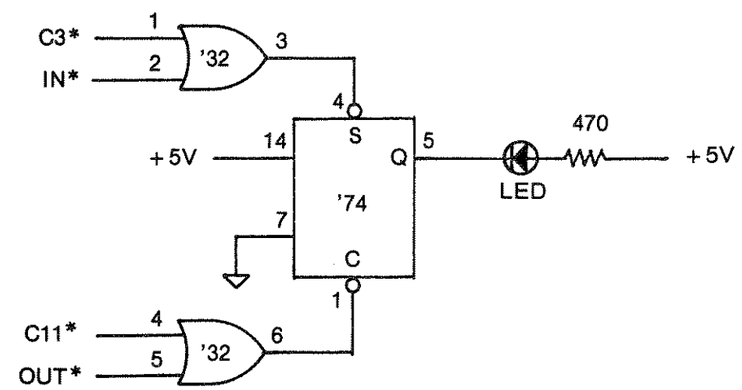


Figure 4.10 Experiment 4.3 Schematic.

STEP 3 Load the following subroutine:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	219	IN A,(N)
16515 / 32131	3	(N)
16516 / 32132	201	RET
:		
16520 / 32136	211	OUT (N),A
16521 / 32137	11	(N)
16522 / 32138	201	RET

by direct POKEs as done previously. Note the memory gap from 16517/32133 to 16519/32135.

STEP 4 When you apply power to the breadboard the LED may or may not turn on. RUN your program. Press any key (except BREAK) to change the state of the 74LS74 output to the LED. Note that both IN* and OUT* can be used as output control pulses (DSPs) to operate the ON/OFF device. A TTL level to high voltage solid state relay device will be discussed in Chapter 6.

STEP 5 Modify the input subroutine with the following instructions:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16516 / 32132	79	LD C,A
16517 / 32133	6	LD B,N
16518 / 32134	0	N
16519 / 32135	201	RET

by direct POKEs.

STEP 6 Add the following line to your BASIC program:

```
35 PRINT L
```

STEP 7 RUN the revised program. The value of L is the value input into the Accumulator. What consistent value is printed after each OFF? It should be 255. Can you explain? Your answer should be that because there is no data at port 3 the value received by the computer is the same as unconnected TTL inputs or all logic 1s, hence the decimal value of 255.

EXPERIMENT 4.4

INPUT PORTS

COMPONENTS 1 * Eight-switch DIP and 16-pin Wirewrap socket
1 * 74LS02 Quad Two-Input NOR Gate
1 * 74LS373 Three-state Octal Latch
8 * 3.3-Kohm resistors

DISCUSSION We have already seen that an input port must be connected to the Data Bus through three-state buffers. We also noted that it might be desirable that the data from the input port be latched so that it is stable when the computer addresses the port. Although there are many TTL ICs that can be used to build an input port, the 74LS373 is a particularly versatile choice. It is a 20-pin IC composed of eight individual data latches with three-state outputs sharing an active high gate latch enable control, EN, and an active low three-state output control, OC*. The truth table is:

OC*	EN	D	Q
0	0	X	Q0
0	1	0	0
0	1	1	1
1	X	X	Z

In the table, X indicates a "don't care" or irrelevant state, Z represents the high impedance state, and Q0 corresponds to the previous value of Q latched before the gate enable control was disabled.

In Experiment 4.5, a second 74LS373 will be implemented as an output port and used with the input port constructed in this experiment. We shall use switches as the data source and therefore do not require the latching capability of the IC. As shown in the schematic, Figure 4.11, the gate enable for latch control is permanently activated by connecting pin 11 to +5 V.

PROCEDURE

STEP 1 To fit an output port on the breadboard in addition to the input port, the components must be placed carefully. Place the first cable socket at the far end of the breadboard. Working towards the other end, leave one row spacing between components: mount the 74LS02 IC, the second cable socket, the 74LS373, and the 16-pin wirewrap socket with the DIP switches plugged into the wirewrap socket.

STEP 2 Bend the leads of the eight 3.3-Kohm resistors perpendicular to the resistor body right next to the ends of the resistor body. Insert the resistors between the +5-V power rail and the DIP switch socket pins on the back side of the breadboard. This will keep the resistors out of the way of the rest of the wires. Run small jumpers from the front side of the DIP switch socket pins to the 0-V rail at the forward edge of the breadboard.

STEP 3 Run +5 V and 0 V to pins 14 and 7 of the 74LS02, and pins 20 and 10 of the 74LS373, respectively. Complete wiring the rest of the circuit shown in the Schematic. Remember to connect pin 11 of the 74LS373 to +5 V and pin 11 of the 74LS02 to 0 V.

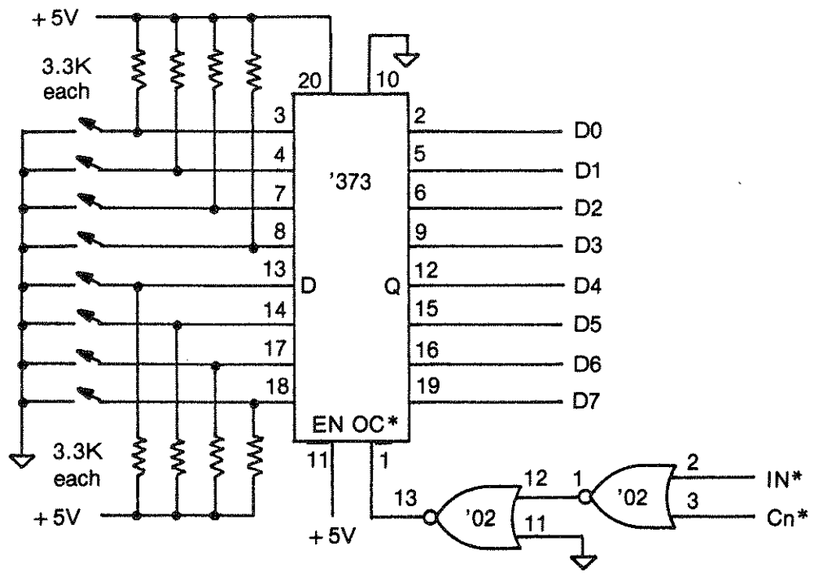


Figure 4.11 Experiment 4.4 Schematic.

STEP 4 Load the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
20 LET L = USR 16514       (for B&W models)
20 LET L = USR 32130       (for Color models)
30 PRINT L
40 PAUSE 33333
50 GOTO 20
```

STEP 5. Using direct POKE commands, enter the following machine language subroutine:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	219	IN A,(N)
16515 / 32131	3	(N)
16516 / 32132	6	LD B,N
16517 / 32133	0	N
16518 / 32134	79	LD C,A
16519 / 32135	201	RET

STEP 6 Set all switches in the same position on the DIP switch. RUN the program. What value was printed on the screen? It should have been either 0 or 255.

STEP 7 Set all switches in their opposite positions. Press any key (except BREAK) to get past line 40. If 0 was printed in Step 6, then you should have 255 printed on the screen now or vice versa.

STEP 8 You can use this input port as an eight-bit binary-to-decimal converter. Try various switch settings remembering to press any key to continue.

STEP 9 Save this circuit for Experiment 4.5.

EXPERIMENT 4.5
OUTPUT PORTS

COMPONENTS
1 * 74LS373 Three-state Octal Latch
8 * LED type MV-50 or T-3/4
1 * 16-pin wirewrap socket
8 * 470-ohm resistors
1 * 74LS02 Quad Two-Input NOR Gate (from Experiment 4.4)

DISCUSSION Refer to Discussion in Experiment 4.4 if you are not familiar with the 74LS373 Octal Latch. We have seen that an Output port is a set of latches activated by an Output Device Select Pulse. We shall use the 74LS373 as the eight latches needed to receive a data byte output by the computer. In this case, we do not need the three-state capability of the 74LS373 because the latch outputs will be used to drive light-emitting diodes. The schematic for the output port is given in Figure 4.12. Because the three-state output control at pin 1 is active low, it is permanently enabled by connecting it to the 0-V rail. The Device Select Pulse for the Gate Enable is made with one NOR gate.

PROCEDURE

STEP 1 The LEDs specified in the Components list are to be inserted directly into opposite pins of a wirewrap socket such that the leftmost LED connects between pins 1 and 16, the rightmost LED connects between pins 8 and 9, etc. The diameters of the plastic case (bubble) for the LEDs specified are less than 0.10 inch in order to fit eight of them side by side without crowding. All LED cathodes should be on one side (pins 1–8). The surest way of determining which lead of the LED is the cathode and which is the anode is to test each one. In the space available on your breadboard, set up the circuit shown in Figure 4.13. Insert each LED between points A and C, if it lights up then the lead at C is the cathode. If the LED does not light, swap leads. If it still doesn't light and you know the rest of your circuit works, then it is defective. Test all eight LEDs as you insert them into the wirewrap socket.

STEP 2 Mount the LED wirewrap socket on the extreme end of your breadboard. There should be enough room left to mount the 74LS373 between the LED socket and the DIP switch

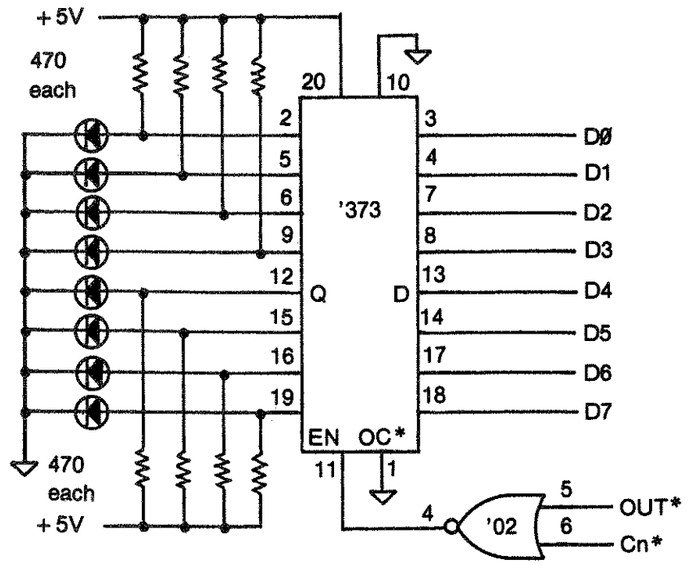


Figure 4.12 Experiment 4.5 Schematic.

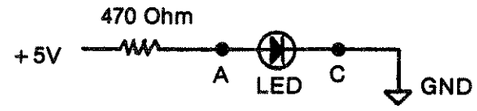


Figure 4.13 LED Test Circuit.

socket of the input port. Insert the eight 470-ohm resistors between the rear edge +5-V power rail and the LED socket pins 9–16 as was done with the DIP switch resistors in Experiment 4.4.

STEP 3 Complete wiring the circuit for the Output Port according to the Schematic, Figure 4.12. As with the switches on the Input Port, all connections to the LEDs are made on the back side of the wirewrap socket except the short ground (0 V) jumpers to the cathodes.

STEP 4 Enter the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
```



```
20 INPUT I
30 POKE 16523,I            (for B&W models)
30 POKE 32139,I            (for Color models)
40 PRINT I; "  " ;
50 LET L = USR 16514       (for B&W models)
50 LET L = USR 32130       (for Color models)
60 GOTO 20
```

STEP 5 Load the following machine language subroutine using direct POKEs:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	58	LD A,(NN)
16515 / 32131	139	(Lo N)
16516 /	64	(Hi N)
/ 32132	125	(Hi N)
16517 / 32133	211	OUT (N),A
16518 / 32134	3	(N)
16519 / 32135	201	RET

STEP 6 To transfer a byte from the BASIC program to the machine language subroutine, we use memory location 16523/32139 as a storage register; in the B&W program, this is the last location in the REM statement. Line 30 POKEs the value of the byte which is input as the variable I into this storage register. The first instruction in the machine language subroutine then "peeks" the same memory location where the two-byte operand of the instruction provides the address: 139 + 64*256 = 16523 for B&W models; and 139 + 125*265 = 32139 for Color models. RUN the BASIC program and INPUT any number between 0 and 255. The LEDs should give the binary equivalent of the decimal value. Thus a value of 170 will give the binary value of 10101010 where the 1s will be lighted LEDs.

STEP 7 We are now ready to combine the input port with the output port. Enter the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
20 LET L = USR 16514       (for B&W models)
20 LET L = USR 32130       (for Color models)
30 PRINT L; "  " ;
40 PAUSE 33333
50 GOTO 20
```

STEP 8 Using direct POKEs, load the following machine language subroutine which reads the value of the switches of the input port, writes the value to the output port, and also returns the value to the BASIC program as the variable L:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	219	IN A,(N)
16515 / 32131	3	(N)
16516 / 32132	211	OUT (N),A
16517 / 32133	3	(N)
16518 / 32134	6	LD B,N
16519 / 32135	0	N
16520 / 32136	79	LD C,A
16521 / 32137	201	RET

STEP 9 RUN the BASIC program. The LEDs should display the switch setting, and the decimal number should be printed on the screen. Select a different switch setting, and press any key (except BREAK) to repeat the loop.

SUMMARY Lamps are generic output ports, and switches are generic input ports. Any other digital data sink (output) or source (input) can be emulated by lamps and switches respectively.

EXPERIMENT 4.6
PROGRAMMABLE INPUT/OUTPUT PORTS

COMPONENTS 1 * 74LS20 Dual Four-Input NAND Gate
1 * 8255 Programmable Peripheral Interface
8 * Logic switches (from Experiment 4.4)
8 * Lamp monitors (from Experiment 4.5)

DISCUSSION The 8255 Programmable Peripheral Interface (PPI) is one of a series of programmable very large scale integrated (VLSI) circuits designed to operate with a microprocessor. The PPI consists of three I/O ports (designated A, B, and C) controlled by a Control Output Port. It is programmable in the sense that a data byte, called a *control word*, output to the Control Port from the microprocessor, is used to configure each of the I/O ports as either an input or an output port. Each I/O port is composed of eight data latches having three-state outputs. Ports A and B always function as eight-bit ports. Port C functions as two four-bit ports (designated C Upper and C Lower) or as individual control bits in conjunction with certain programmed configurations of Ports A and B.

The PPI is a 40-pin DIP. Thirty-two pins are used in groups of eight for Ports A, B, and C, and for the Data Bus. Of the remaining eight, two pins connect to +5 V and ground (0 V), and one pin, RESET, when in the logic 1 state configures all three I/O ports as latched output ports. The last five pins are used for device selection. These include RD* (connected to IN*), WR* (connected to OUT*), a1 and a0 which connect to two address lines of the Address Bus (not necessarily Address Bus lines A1 and A0 as we shall see below), and CS* (Chip Select) which for Accumulator I/O is used to enable the PPI with a decoded pulse from the other six Low Address

Bus lines. There are four distinct devices in the PPI; namely, the Control Port and ports A, B, and C. Two address bits must be used in order to distinguish the four ports. The truth table for device selection is:

CS*	a0	a1	PORT SELECTED
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	Control
1	X	X	Three-State

The PPI can operate in three distinct modes, designated Modes 0, 1, and 2. In Mode 0, each I/O port can be either a latched output port or an unlatched input port. In Mode 1, Ports A and B may be either latched output ports (as in Mode 0) or latched input ports. As we noted in the chapter text, a latched input port requires additional control (handshaking) bits: one bit called STroBe to cause the gating of the latch of an input port; another called Input Buffer Full to indicate that data has been latched into the input port; a third bit called Output Buffer Full to latch the output port; and a fourth bit called ACKnowledge to enable the three-state outputs of an output port. These additional bits are assigned to certain bits of Port C when either Port A or B is configured as a latched input port in Mode 1 or 2. Table 4.2 summarizes the effect of the three modes on the three ports. In Mode 2, Port A can be configured as a bidirectional (that is, both input AND output) port. We shall not have occasion to investigate Mode 2 operation and will leave discussion of its operation for more advanced texts.

TABLE 4.2 PPI MODES

	PORT A	PORT B	PORT C
MODE 0	Latched Output OR Unlatched Input	Latched Output OR Unlatched Input	Latched Output OR Unlatched Input
MODE 1	Latched Output OR Latched Input	Latched Output OR Latched Input	PC7:OBF*(A):I/O PC6:ACK*(A):I/O PC5:IBF(A):I/O PC4:STB*(A):I/O PC3:INTR*(A) PC2:STB*(B):ACK*(B) PC1:IBF(B):OBF*(B) PC0:INTR*(B)
MODE 2	Latched Output AND Latched Input	same as Mode 0 OR Mode 1	PC7:same as Mode 1 PC6: " PC5: " PC4: " PC3: " PC2:same as Mode 0/1 PC1: " PC0: "

The control word data byte output to the Control Port of the PPI performs two functions. If the most significant bit of the control word, D7, is a logic 1, the control word is used to program the mode of the PPI and configure each I/O port as either an input or output port. The bitwise structure of the control word for mode format (D7 = 1) takes the form:

D7	D6	D5	D4	D3	D2	D1	D0
	Mode A,CU		Port A	Port CU	Mode B,CL	Port B	Port CL
1	0	0	1=I	1=I	0	1=I	1=I
	0	1	0=O	0=O	1	0=O	0=O
	1	X					

If D7 of the control word is a logic 0, individual bits of Port C can be Set (to a logic 1) or Reset (to a logic 0). This is of particular importance for Modes 1 and 2. The control word then takes the bit format:

D7	D6	D5	D4	D3	D2	D1	D0
	---NOT USED---				Port C Bit Select		1=SET
0	X	X	X		(Bit value 0-7)		0=RESET

PROCEDURE

STEP 1 Mount the DIP switches and resistors on the end of the breadboard, then mount the LED socket and resistors next to the DIP switches as in previous experiments.

STEP 2 Mount the two ICs on the breadboard with the 74LS20 located between the two cable sockets. Wire the circuit as shown in the Schematic, Figure 4.14. According to the decoding scheme used, verify that C3 = Port A, C11 = Port B, C19 = Port C, and C27 = Control Port.

STEP 3 ENTER the following BASIC program:

```
10 REM 1234567890          (for B&W models)
10 CLEAR 32129             (for Color models)
20 LET L = USR 16514       (for B&W models)
30 LET L = USR 16518
20 LET L = USR 32130       (for Color models)
30 LET L = USR 32134
40 PAUSE 33333
50 GOTO 30
```

STEP 4 Determine the control word for formatting the PPI for Mode 0 with Port A as input and Ports B and C as outputs. You should determine the following binary word:

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

which is 144 in decimal.

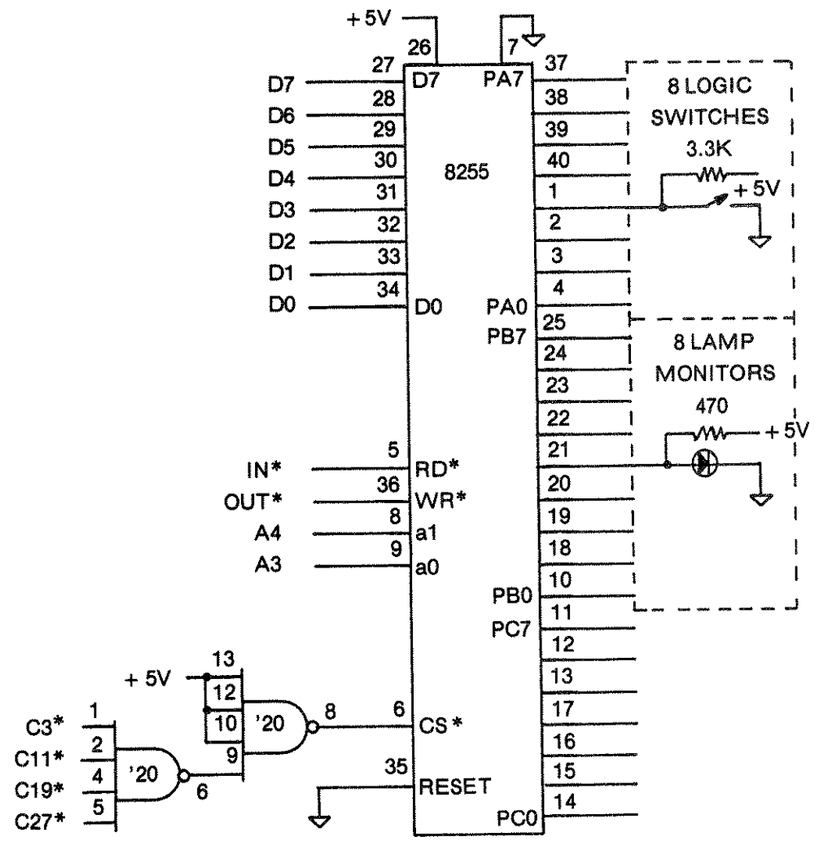


Figure 4.14 Experiment 4.6 Schematic.

STEP 5 Enter the following machine language subroutine using direct POKEs:

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC
B&W / Color		
16514 / 32130	62	LD A,N
16515 / 32131	144	N
16516 / 32132	211	OUT (N),A
16517 / 32133	27	(N)

16518 / 32134	219	IN A,(N)
16519 / 32135	3	(N)
16520 / 32136	211	OUT (N),A
16521 / 32137	11	(N)
16522 / 32138	201	RET

STEP 6 RUN your BASIC program. Set the DIP switches for some value, then press any key (except BREAK). The LEDs should light according to the switch setting. If they do not, remove the RESET line at pin 35 from Ground momentarily and then reinsert it into the 0-V rail. Continue looping by pressing any key to get past the PAUSE in line 40.

SUMMARY This cursory look at a programmable integrated circuit should serve to illustrate the power and potential of these devices. Two points are of immediate interest to the interfacer: first, there is a cost savings of about 33% using one 8255 instead of three 74LS373 ICs to achieve the same purpose; and second, both physical space on a circuit board is saved and fewer connections are required using the VLSI chip.

□ 5 □

digital conversions

Once you have constructed an input port using switches and an output port using lamps, you have accomplished the first step in interfacing a computer to the outside world. The most important fact to realize is that any digital input signal can be substituted for the switches of your generic input port. Similarly, any digital output device can be substituted for the LEDs of the generic output port. Bear in mind that the input port still needs its three-state buffers and the output port still needs its latches, and that both have to be selected with a unique Device Select Pulse generated from the Device Code of the Address Bus and the proper Control Bus pulse (IN* or OUT*).

There are many signals which are digital. Of course, these signals always correspond to some binary condition such as On/Off, Up/Down, True/False, Left/Right, Over/Under, Opened/Closed, etc. It is always necessary that input devices convert their binary conditions to the +5-V and 0-V signals to be interfaced to the computer input port. A very common example of a digital data source is a set point determination of temperature, pressure, extension (length), etc., where one bit of information is all that is needed. The one-bit signal indicates, for example, that a temperature is exceeded or not—exactly like a thermostat. With an eight-bit input port, eight individual set points can be monitored by the computer. When the computer reads the data bits of the input ports, it can make a decision based on the status of each bit. If those bits are thermostat set points, then the computer can use the individual bits of an output port to control heater relays, circulator motor switches, indicator lights, etc. The problem then is to convert the +5 V or 0 V of the output port bits to power levels appropriate to handle the output devices. We shall cover several methods for doing this in this chapter and in Chapter 6.

Before discussing digital input and output further, we should note that the other type of signals one encounters in the outside world are analog signals. These are signals whose values vary continuously over their operating range rather than having just two (binary) states. For example, the thermometer is an analog device whereas the thermostat is a digital device. Any signal that can be read with a pointer on a scale is an analog signal. There are hybrid integrated circuits which convert between an analog signal and an eight (or 10 or 12) bit number proportional to the value of the analog signal. These are called Analog-to-Digital (ADC) and Digital-to-Analog (DAC) converters. We shall discuss analog to digital conversions in Chapter 6. In the rest of this chapter, we shall consider the various ways that digital signals are used.

PARALLEL-SERIAL CONVERSIONS

We have seen that data can be transmitted to an output port and received from an input port as an eight-bit byte, or in other words, over eight parallel wires (the Data Bus). It is also possible to transfer data over one data line (actually two wires with one wire serving as the electrical common potential between the transmitter and receiver). Instead of sampling eight lines in parallel simultaneously, the data on one wire is sampled serially at successive times. Serial transmission has several practical advantages over parallel transmission not the least of which is the cost of the wire when long distances are involved. Of course, for large distances the pair of wires provided by the telephone system is the most widely used data transmission network.

When data is transmitted serially, there are several possible ways that the information can be encoded. One of the oldest methods is Morse code made up of dots and dashes. In this method, the information was transmitted as bursts of a tone with a dash being three times longer than a dot, and periods of silence between the tone bursts of a character (alphabetic letter, numeral, symbol, etc.) equal to the interval of a dot and between characters equal to a dash. The more common method since the teletypewriter replaced the telegraph key is with the American Standard Code for Information Interchange (ASCII) mentioned in Chapter 1. Each character is encoded in seven bits as shown in Appendix Chart A6. In terms of TTL signals, logic 1 and 0 states correspond to +5 V and 0 V respectively. When an ASCII character is transmitted serially, each bit lasts for the same period of time with one immediately following after another. If we were to use the tone analogy of Morse code, then a dot tone would be a logic 1 and a dot rest (of silence) would be a logic 0. The duration of the bits is determined by the *bit rate*, that is the number of bits that are transmitted in 1 second. For example, if the bit rate is 100 bits per second (bps) then the time duration of each bit is 10 milliseconds. Another term you may hear applied to bit rate is *Baud rate*. This term comes from the field of international radiotelegraphy which defined the Baud rate in terms of the shortest tone bursts of Baudot code (a five-bit, 32-character code similar to the seven-bit, 128-character ASCII code). When all bits have the same duration, bit rate is equivalent to Baud rate.

Before considering the various methods other than TTL signal levels used in serial data transmission, we should first consider how the parallel data byte from the microcomputer is converted into a serial string of bits. Within the 7400 series of ICs, there are several types of ICs called *shift registers* which perform these conversions. They are distinguished as parallel in-serial out (PISO) and serial in-parallel out (SIPO). Examples are the eight-bit registers 74LS165 and 74LS164 respectively. (There are also serial in-serial out and parallel in-parallel out shift registers.) The shift registers operate as a set of (eight) cascaded data latches where the Q output of the first latch is connected to the D input of the second latch and so forth. All latches have their clock inputs connected in common and may also have a common clear input. One of the first large scale integrated (LSI) circuits to be produced, which performed both conversions, was the Universal Asynchronous Receiver Transmitter (UART). The UART is not a TTL IC; however, it operates between +5 V and 0 V and therefore is TTL compatible. In addition to the input and output data lines, these circuits require timing (clock) inputs. The UART has additional control inputs and outputs whose functions are more easily understood once the conventions of serial transmission have been described.

In asynchronous serial transmission of data, only two wires are used as mentioned above. Because the only information that can be passed is over the data line, the receiver has no other means of synchronizing with the transmitter; that is, the receiver cannot know beforehand when the transmitter is going to start; therefore, the transmission is said to be *asynchronous*. Between the transmission of characters, the data line is held in the logic 1 state. To synchronize the receiver with the transmitter, the first bit transmitted is always a logic 0 "start" bit. Following the start bit, the data bits are transmitted in order with the least significant bit sent first. Following the seven data bits (for ASCII code), an error-checking bit called the *parity bit* may be appended, finally one or two stop bits in the logic 1 state conclude the transmission of a character. In all, a maximum of 11 bits per ASCII character are transmitted. The timing diagram for the transmission of the ASCII character "C" (1000011) at 110 Baud is illustrated in Figure 5.1.

The schematic and pin assignment of a UART is shown in Figure 5.2. In the diagram, each of the small rectangles corresponds to either a data latch or a set-reset flip-flop. The three-state buffers are self-explanatory. The frequency of the clock signals for the receiver and the transmitter operate at 16 times the bit rate and are usually tied to a common clock generator. Five programming bits permit formatting for the data transmission: two bits of the data word inputs select the length of the data word between 5 (00 at pins 37 and 38) and 8 (11 at pins 37 and 38) bits; inclusion of a parity bit is determined by the logic state of pin 35; even or odd parity is selected by the logic state of pin 39; the number of stop bits (one or two) is determined by the logic state of pin 36. Three status bits from the receiver indicate the various reception errors at pins 13, 14, and 15. We give only a brief description of the UART here in order to familiarize you with the terms. We shall examine a newer IC, the 8251, which is more computer compatible (i.e. programmable), in Experiment 5.6.

There are several other concepts concerning serial data transmission which we shall enumerate here in order to complete an introduction to the topic. There are three common transmission modes that are encountered in serial data transfer. These modes are: (1) *Simplex* in which only two wires connect the terminals and the data transfer is

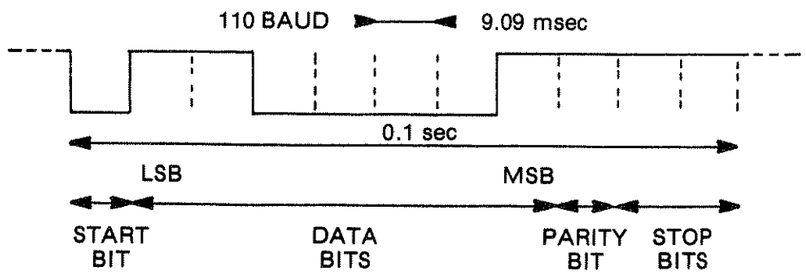


Figure 5.1 Serial Transmission of ASCII.

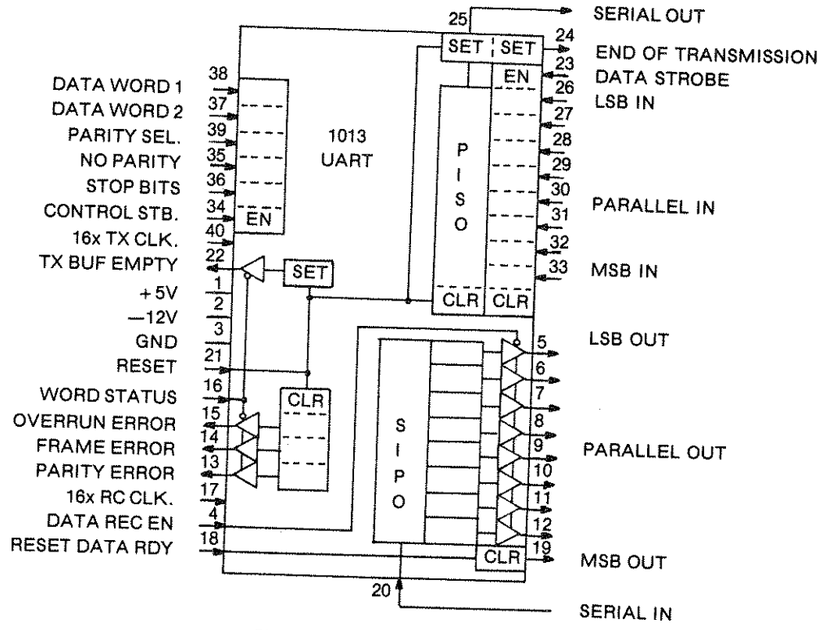


Figure 5.2 UART Block Diagram.

only in one direction, that is, one terminal is always the transmitter and the other is always the receiver; (2) *Half duplex* in which two wires connect the terminals but data can be alternately transferred in either direction (not at the same time!) with each terminal operating alternately as a receiver and a transmitter; and (3) *Full duplex* in which two pairs of wires connect the terminals and allow simultaneous data transfer in both directions. We shall see shortly that full duplex transmission is possible over one pair of wires (telephone connection) if the bits being transmitted are tone bursts of sufficiently separated frequencies rather than DC logic levels.

In addition to the method of transmitting data serially with TTL level signals (logic 1 = +5 V, logic 0 = 0 V), there are three other methods which have various advantages relating to noise immunity, speed, power, or convenience. The three methods are (1) current loops, (2) RS-232c voltage standard, (3) and frequency shift keying (FSK). Again, our immediate interest is to introduce the concepts rather than exhaustively cover the topic. In the serial transmission of data via a current loop, the pair of wires used in simplex or half duplex mode rely on the existence of a current flow to indicate a logic 1 state and the absence of current flow to indicate a logic 0 state. Data can be transmitted up to 5 miles (8 km) on a current loop. The standard for the current is either 20 mA or 60 mA, depending on the system design. The voltage applied to the

wires is whatever it takes to provide the specified current—usually between 12 and 18 V.

We should note that the pair of transmission wires referred to in serial transmission is either a twisted pair or coaxial cable. Both of these techniques increase noise immunity of the transmitted signals by reducing the amount of induced pickup from external sources.

The RS-232c is a voltage standard supported by the IEEE (Institute of Electrical and Electronic Engineers). The logic levels are defined as a voltage between +5 and +15 V for the logic 0 state and a voltage between −5 and −15 V for the logic 1 state. Data can be transmitted up to 50 ft without significant signal loss. To convert between TTL and RS-232 signal levels, the MC1488 and MC1489 Quad Driver and Quad Receiver integrated circuits have regular TTL inputs and outputs respectively. Connection of these ICs to a power supply providing the RS-232 voltages (typically +9 V and −9 V) gives RS-232 level outputs (driver) and accepts RS-232 inputs (receiver). In addition to the voltage level specification, the arrangement for the terminal connection is defined by the standard to consist of a 25-pin connector with the various signals defined as shown in Table 5.1 (not all pins defined). In many cases only the three (*) signals are used for asynchronous serial transmission.

The third method of serial transmission is the FSK technique used primarily for telephone transmission. There is no practical limit to the distance because the telephone system manages the signal conditioning. The frequency range of ordinary telephone lines lies between 300 and 3300 Hz. The logic levels are defined in terms of tone bursts of specified frequencies. Devices used to convert a digital signal (TTL, current, or RS-232 standard) to the proper transmission frequency (tone) are called *Modulators*, those devices which convert a received frequency back to a digital signal are known as *Demodulators*. The combined device used for two way communication of serial data is referred to as a *Modem* (MOdulator-DEModulator). For full duplex

TABLE 5.1 RS-232 PIN ASSIGNMENT

PIN NUMBER	FUNCTION
1	Frame Ground
2*	Serial Data Out
3*	Serial Data In
4	Request to Send Flag
5	Clear to Send Flag
6	Data Set Ready Flag
7*	Data Signal Common
8	Carrier Detect Signal
9	+12-V Power
10	−12-V Power
15	Synchronous Transmit Clock Signal
17	Synchronous Receive Clock Signal
20	Data Terminal Ready Flag
22	Ring Indicator Signal
24	External Transmit Clock

operation with a modem, the receive and transmit frequencies are different as shown in the following table:

Frequencies (Hz) for Full Duplex Transmission

	Logic 1	Logic 0
Send/Originate mode	1070	1270
Receive/Answer mode	2025	2225

This requires that one terminal operate in the Originate mode and the other operate in the Answer mode. Because this is full duplex operation, both terminals can send and receive, but agreement must be established between them. Thus terminal #1 operating in the Originate mode will send (modulate) its data at the lower frequency band and receive data from terminal #2 (demodulate) the higher frequency band. Terminal #2 will modulate its transmission on the higher frequency band and demodulate its reception from the lower frequencies. It might be noted that the less expensive modems are equipped for receive mode only.

SERIAL TIMING AND FREQUENCY CONVERSIONS

We have considered the transmission of data encoded into serial strings of bits but there is still another way of transmitting serial information. Whereas the serial communication described in the preceding section concerned bit rates with each bit in the string having an equal duration, it is also possible to transmit data on a single data line (really a twisted pair of wires) where the length of the time the data bit is, say, in the logic 1 state proportional to the value of the data. This type of digital conversion is the software equivalent of the gated digital counter or frequency converter examined in Experiment 2.5. Recall that two signals can be gated and fed to the input of a counter. If one of the signals is a monostable pulse of known duration and the other is an astable pulse of unknown frequency then we can determine the frequency of the astable based on the number of pulses in the known time of the gating monostable pulse. On the other hand, if the astable frequency is known, then the duration of the monostable pulse can be determined from the count.

A similar experiment can be performed with a computer by using the computer in place of the digital counter. For example, a computer could measure the duration of a monostable pulse by triggering the monostable, such as the 555 timer or one in the 7400 series, using an output Device Select Pulse and then immediately start to read the logic state of the monostable through a one-bit input port. The computer program would consist of reading the port and incrementing an eight- (or 16-) bit counter if the logic level were 1, and then staying in a program loop (i.e., jumping back to read the input data bit and increment the count) until the monostable pulse fell back to a logic 0. If either the resistor or the capacitor in the RC timing circuit of the monostable were a transducer whose value depended on some physical property such as temperature or pressure, then the count stored by the computer would be proportional to the value of the physical property. The advantage of this type of conversion is its simplicity in

hardware. Its disadvantages are the need for more sophisticated software and the long conversion times (typically several milliseconds). If the physical property changes more rapidly than the conversion time, the method is impractical. These problems are typical of the trade-offs between hardware and software always faced by the interfacer in designing and developing a measurement device.

The same interfacing technique can be used to measure the frequency of an astable (that is, of a square wave generator) again using the computer as the counter. This procedure requires that the program be written to examine the one-bit input port repeatedly until it observes the edge (positive or negative) of a square wave pulse. In a manner similar to the preceding description the computer then stays in a counting loop until it finds the subsequent opposite edge. By sampling the duration of the high pulse of the square wave and then sampling the duration of the low pulse of the square wave, the two counts can be converted to times by determining how long the program loop takes per count. Once the times are calculated, the frequency of the square wave is obtained as the reciprocal of the period. One interesting application of this method is the measurement of an unknown capacitance by insertion into the RC circuit of a 555 Timer astable circuit. The disadvantage of the method is the relative slowness of the computer operating in the microseconds per count range compared to that of a digital counter which can operate in the tens of nanosecond range.

We can conclude this section on timing and frequency conversions of a digital signal by pointing out that the gated counter could be directly interfaced to a computer rather than to seven-segment displays. If the counter consisted of two cascaded binary counters (74LS93s) then the computer could read the outputs of the pair as an eight-bit input port. Each additional cascaded pair of counters would form an additional input port. We shall defer further discussion of inputing measurement data until Chapter 6 when we consider analog-to-digital conversions.

DIGITAL-TO-ANALOG CONVERSION

The microcomputer is often required to deliver analog information to the outside world for the purpose of providing data, for example, to deflect the pointer of a meter or the pen of a recorder, or for providing proportional control of instruments such as turning on a heater or opening a valve to some partial setting. To do so requires conversion of information from digital form into analog form. Again use can be made of readily available integrated circuits such as the type AD558 Digital-to-Analog Converter (Analog Devices) shown in Figure 5.3.

The conversion process for all intents and purposes can be regarded as a precision voltage divider. Actually, the DAC first produces a current proportional to the digital value using an internal precision resistor network (called a *R-2R ladder*), this signal is then fed to an internal current-to-voltage amplifier to obtain a proportional voltage. Many other DAC ICs produce a current output and require an additional current-to-voltage (operational) amplifier. By having a built-in amplifier, the AD558 is particularly convenient to use. The supply voltage to the DAC must be larger than the maximum analog output voltage to allow reaching the full scale output voltage of the

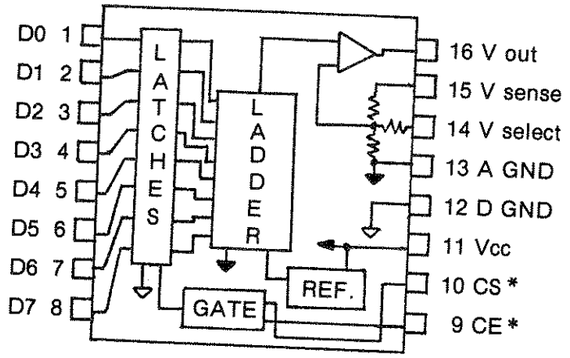


Figure 5.3 AD558 Pin Configuration.

analog output. The digital range from 0 to 255 provides an analog voltage range from 0 V to either 2.55 V or 10.0 V. Each bit of the digital data is converted to a signal proportional to its binary weight: from the least significant bit (D0) with a weight of one unit to the most significant bit (D7) with a weight of 128 units. For each 1 V of V_{ref}, the LSB has a unit weight of 0.0039 V (1/255) and an MSB weight of 128 times the unit weight, or 0.0039 * 128 = 0.502 V. Thus the DAC adds to the analog output voltage a precise voltage corresponding to the weight of the bit for each bit that is in a logic 1 state. For example, with a reference voltage of 2.55 V, the decimal value of 123 in binary is:

Bit weight(V):	1.28	0.64	0.32	0.16	0.08	0.04	0.02	0.01
Data bit position:	D7	D6	D5	D4	D3	D2	D1	D0
Binary data (100):	0	1	1	1	1	0	1	1

Analog output, V = 0.64 + 0.32 + 0.16 + 0.08 + 0.02 + 0.01 = 1.23 V.

The full scale voltage (all eight digital 1s) is 2.55 V. If higher voltage outputs are required, then it is possible for the supply voltage to be operated from a 12-V power supply to output an analog voltage range from 0 to 10.0 V (i.e. the binary weight of the least bit [unit weight] is 0.039 V). Can you show that the analog voltage on the 10-V range for the digital decimal value of 123 would be 4.804 V? Again we stress that for the low cost interface a single power supply is a prime requirement, so reference voltages less than +5 V will be used in our experiments. These can be easily obtained from the +5-V supply voltage. This is not too serious a disadvantage as circuitry is often at hand to boost the voltage, for example, by using an oscilloscope or operational amplifier.

Because the DAC functions as an output port and only receives information from the microcomputer, the only control signals required are the OUT* signal wired to the Chip Enable (CE*) input together with the Device Code wired to the Chip Select

(CS*) input. When these two active low pulses are produced by the machine language program in the microcomputer then the eight-bit digital data is latched into the input buffer of the DAC and converted to an analog voltage at the output pin for use by other devices. The latching action occurs only when the gating control signal makes the transition from its enabled to its disabled state. This means that as long as the latch is enabled, the analog output reflects the data at the D inputs.

The supply voltage, V_{cc}, can be from +5 V to +15 V. The reference voltage, V_{ref}, for the analog output can be wired for either a 0 to 2.55-V range or a 0 to +10-V range. The data inputs are TTL compatible irrespective of the supply (or reference) voltage. Besides connections to the two control pins, the supply voltage, and the ground pins (one for the digital and one for the analog signals), other connections to the DAC include the Data Bus connections of D0 to D7, the analog signal output pin, and two analog feedback resistor input pins. Connections to these three pins will be described in the experiments. Because the time required to make a conversion is 1.0 microseconds and a machine language routine takes about 2 microseconds to perform an OUT instruction, there is no need to provide time delays to permit the DAC enough time to produce a valid analog output from a digital input byte.

STEPPER MOTOR CONTROL

A stepper motor is a digital device which can be actuated by the parallel bits of an output port. Permanent magnet (PM) stepper motors are the simplest to understand and use with a microcomputer. We will consider only PM steppers in this survey. A PM stepper motor consists of a cylindrical permanent magnet rotor attached to the rotation shaft. The rotor body is magnetized around its circumference with alternating strips of north and south poles lying along the length of the cylinder. The number of strip poles is one of the determining factors in how many degrees of rotation the rotor turns for each step. A view of the rotor is shown in Figure 5.4. The stator, which is positioned around the circumference of the rotor, consists of two coils of wire housed in a metal sheath or case. In the less expensive steppers, known as "tin can" stepper motors, the stator cases are made in two interlocking sections of pressed sheet metal. When assembled, the case forms a "donut" with solid walls for its two sides and outer

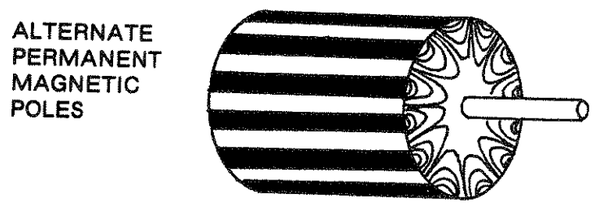


Figure 5.4 Stepper PM Rotor.

perimeter and sets of alternating teeth around its inner perimeter. Four wires extend from the case corresponding to the two ends of the two coils wound inside the stator case. The number of coils determines the number of phases of the motor.

Each alternate pair of teeth in the stator case can be imagined as a flattened nail bent into a square ring with its ends situated beside each other. If you were to wrap a coil of wire through the square and pass an electric current through the coil then you would have an electromagnet with one end of the "nail" as north pole and the other end as south pole. If a large set of these "nails" were laid side by side in a circle and the wire coil was wrapped inside them, you would have a stator case where every other tooth was a north pole and the other alternating set were south poles. If the PM rotor were arbitrarily positioned inside the stator, it would rotate to align its north poles with the south pole teeth, and vice versa. The number of teeth that form the inside perimeter of the stator case is the other determining factor in how many degrees of rotation (step size) will be involved in a single step. An exploded view of a tin can stepper is shown in Figure 5.5. The stepper illustrated has a stack of two stator cases positioned side by side around the rotor. This type of stepper is a four-phase motor.

The teeth of the one stator case are positioned around the rotor to lie halfway between the positions of the teeth of the other stator. If we assume that there are 24 (12 pairs) teeth on each stator and 12 bar magnets around the circumference of the rotor, then the smallest step will be 360 degrees/48 = 7.5 degrees. For example, suppose current is run through one of the coils in each stator case so that the 12 electromagnet poles of each stator are as shown in the following chart.

```
Rotor front:    s   n   s   n   s   n
Stator #1:    N:  S:  N:  S:  N:  S:  N
Stator #2:     :N  :S  :N  :S  :N  :S
Rotor rear:     s   n   s   n   s   n
```

Now suppose we run the current in stator #2 coil in the opposite direction, that is swap the ends of the coil at the voltage terminals. The poles in stator #2 would be reversed, and the rotor would turn to line up with the new positions, as shown in the following chart.

```
Rotor front:      n   s   n   s   n   s
Stator #1:    N  :S  :N  :S  :N  :S  :N
Stator #2:       S:  N:  S:  N:  S:  N:
Rotor rear:       n   s   n   s   n   s
```

Careful observation of the difference between the first and second charts shows that the rotor moved one-quarter of the distance between two of the like poles of stator #1. Because there are 12 like poles wrapped around the circumference, then the rotor moved 1/4 of 1/12 or 1/48 of a full turn of 360 degrees.

Rather than disconnecting the coil leads to run current in the opposite direction, the two coils in each stator case are wired for the current to flow in opposite directions.

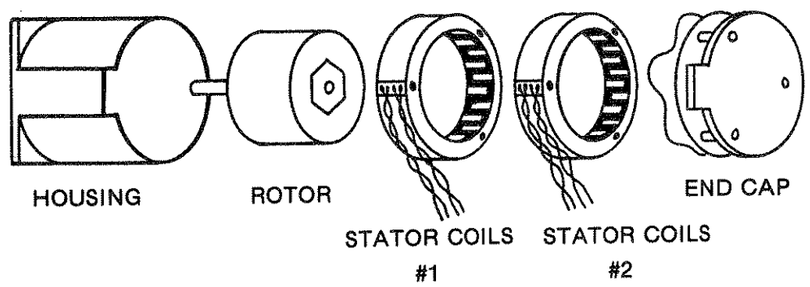


Figure 5.5 Disassembled Tin Can Stepper.

However, current only ever flows in one coil of each stator at a time; otherwise their electromagnetic effects would cancel each other out. If we label each of the coils in a stator as A and B, then we have to know the sequence of allowing current to flow through the four coils 1A, 1B, 2A, and 2B with the restriction that if 1A is turned on, then 1B must be turned off, etc. This is obviously a binary condition, and we can represent an "on" with a logic 1 and an "off" with a logic 0. The sequence for stepwise rotation is shown in the following table.

Direction:	CLOCKWISE				COUNTERCLOCKWISE			
Coils:	1A	2A	1B	2B	1A	2A	1B	2B
Start position:	0	1	1	0	0	1	1	0
Ist Step:	1	1	0	0	0	0	1	1
2nd Step:	1	0	0	1	1	0	0	1
3rd Step:	0	0	1	1	1	1	0	0
4th Step:	0	1	1	0	0	1	1	0

By the 4th step we have returned to the same binary pattern as the start position so subsequent steps just repeat the pattern. If you look closely at the pattern sequence, you will see that the four bits are shifted to the left for the clockwise direction and to the right for counterclockwise. When a bit is shifted off one end of the pattern it is brought back in on the other end. Because our computer has an eight-bit data output all we have to do is write the pattern of four bits twice. For example, the following machine language routine is an endless loop which would continuously run a stepper motor connected to output port #3 which latches data bus lines D7–D4.

```
START   LD A,N      :Stepper byte is
        51          :binary 00110011.
LOOP    OUT (N),A   :Activate stepper through
        3           :output port device code 3.
        RAL         :Shift the bits for clockwise
```

```
                (or)   RAR          :(for counterclockwise).
                       CALL DELAY   :A time delay subroutine
                       Lo Address   :because the stepper cannot
                       Hi Address   :respond fast enough.
                       JR           :Stay in endless loop
LOOP+7                 248          :by jumping back -8 to LOOP.
DELAY                  ...          :Address of DELAY subroutine.
```

Stepper motors are power devices capable of performing work. The current drive of the outputs of a TTL latch is not large enough to drive the coils of the stepper. To interface a stepper motor, a power source of several amps at the operating voltage of the motor is necessary. The TTL signals can be used to control transistor switches having the proper current rating. Transistors such as the NPN D40K (General Electric) are capable of handling a current load (per coil) of 2 A at 30 V. One lead from each of the four coils is connected to the power supply. The other lead from each coil is connected to one side of the NPN transistor switch (collector) with the other side of the transistor switch (emitter) tied to the common side (ground) of the power supply. The TTL signal is connected to the base of the transistor. When the TTL signal is a logic 1, the switch is closed, and current from the power supply flows through that coil. A schematic used by the authors with a stepper operating from a 5-V power supply and drawing 0.5 A per coil (12 ounce inch torque rating) is shown in Figure 5.6. Further discussion of transistor drivers is in Chapter 6.

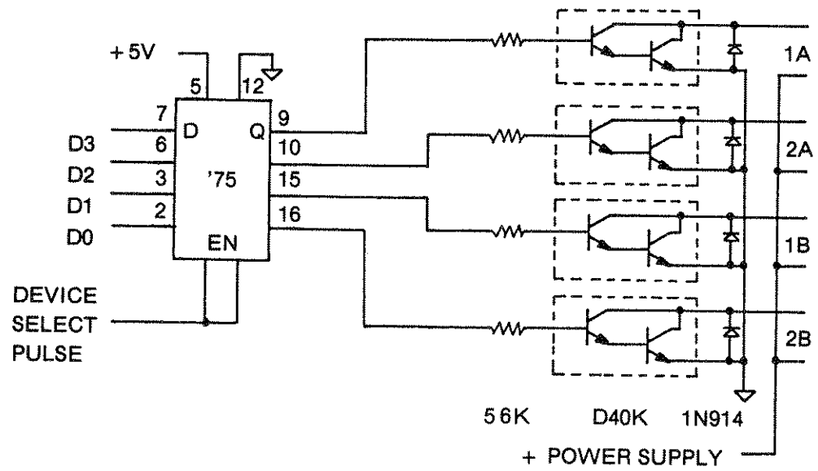


Figure 5.6 Stepper Motor Driver Interface.

EXPERIMENT 5.1
POSITION DETECTION AND DISPLAY

COMPONENTS
- 1 * Joystick control with four 100-Kohm potentiometers or two 50-Kohm potentiometers
- 1 * 556 Dual 555-type Timer
- 1 * 74LS32 Quad Two Input OR Gate
- 1 * 74LS125 Bus buffer
- 4 * 1-Kohm resistors
- 2 * 0.2-μF capacitors
- 2 * 0.01-μF capacitors

DISCUSSION In this experiment we shall examine the technique of digital data acquisition by timing conversion from resistance transducers. Control of the motion of a dot on the video screen in both the X and Y coordinate directions determined by the position of a joystick is an extremely useful experiment having applications in graphics techniques and, of course, games. Although the commonly available joysticks use potentiometers which tend to be highly nonlinear, the following experiment will result in a well defined X–Y border but a somewhat questionable set of diagonals. Center point of the joystick is the center of the screen.

The means of converting the position of the joystick to the position of a dot on the video screen is accomplished in a very simple manner. As the top of the joystick control moves, the wipers on the potentiometers connected to the base of the joystick change position and vary the resistance of each potentiometer. As the schematic shows, each pair of potentiometers, one pair referred to as the YY′ pair and the other pair as the XX′ pair, is connected to the RC (timing resistor and capacitor) input of one of the two 555-type timers configured as monostables.

In this mode each monostable pulse will have a period that depends on the value of R and C in its timing network. Because C is fixed at 0.2 μF, variation of R, which is dependent on joystick position, will vary the time period. By having the microcomputer measure the time interval produced by the X pair separately from the time interval produced by the Y pair, timing counts can be obtained which can then be plotted as X and Y values on the video screen.

The time interval of each timer is determined by triggering both timers simultaneously with a device select pulse (OUT 3*) applied to the trigger inputs, pins 6 and 8, of the two timers and then polling the outputs of each timer, pins 5 and 9, respectively, through a two-bit input port to determine the exact instants that each output falls back to a logic 0.

The microcomputer does this by inputting the state of the two timer outputs through three-state bus buffers of the 74LS125. One input is gated to data bus line D0 and the other to data bus line D7. Thus by inputting the states of the data bus lines into the accumulator register A, the microprocessor can check which timer's output fell first and note the time of the first fall while still maintaining a count for the second timer. By converting each time to a displacement in the X and Y directions, a position on the video screen can be PLOTted.

PROCEDURE

STEP 1 Inspect the four (or two) potentiometers controlled by your joystick. Opposite pairs should be wired in parallel, as shown in the schematic. Note the 1-Kohm limiting resistors wired to each potentiometer and make sure that the +5 V goes to each pair of potentiometers separately (don't try to save money by only using one or two 1-Kohm resistors, as the case might be).

STEP 2 Wire the remainder of the circuit as shown in the schematic, Figure 5.7, not forgetting to wire +5 V and 0 V to all integrated circuit chips used. Apply power after the circuit has been checked.

STEP 3 The BASIC program is quite straightforward. When the program returns from the USR subroutine called at line 30, the variable L has the 16-bit count of the B and C register pair returned automatically. The BASIC variable C has its value determined by the contents of memory locations 16561-16562/32177-32178. The values of the variables so formed are scaled to fit on the video screen. *NOTE:* You may have to change these scaling factors yourself if the potentiometers in the joystick you use are much different from the values stated in this experiment.

Load the BASIC program.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789 1234567890
                                                  (for B&W models)
20 FAST
30 LET L = USR 16514
40 LET C = PEEK 16561 + 256 * PEEK 16562
     * * * *
10 CLEAR 32129                                    (for Color models)
30 LET L = USR 32130
40 LET C = PEEK 32177 + 256 * PEEK 32178
     * * * *
50 LET N = 33333
60 LET X = L/11.5
70 LET Y = C/20
80 PLOT X,Y
90 PAUSE N
95 GOTO 20
100 FOR M = 16514 TO 16557                        (for B&W models)
100 FOR M = 32130 TO 32173                        (for Color models)
110 INPUT N
120 POKE M,N
130 PRINT M; "=" ;PEEK M;
140 NEXT M
```

STEP 4 Inspection of the machine language program will show that after inputting the states of the data lines at location 16524/32130, the accumulator is ANDed with the contents of E. Register E holds a mask having the decimal value 129, which in binary is 10000001. As can be seen from the binary representation there is a logic 1 state at each of the positions corresponding to D0 and D7, the two lines we wish to check.

Two loops are set up: the first loop waits until the first timer output has fallen to 0 while incrementing the BC pair counter register each time round the loop, the second loop saves the count of the first timer while continuing to increment the BC counter until the second timer output has fallen to 0.

Finally on exiting from the loops a check is made on which timer output fell first: the X or the Y. If

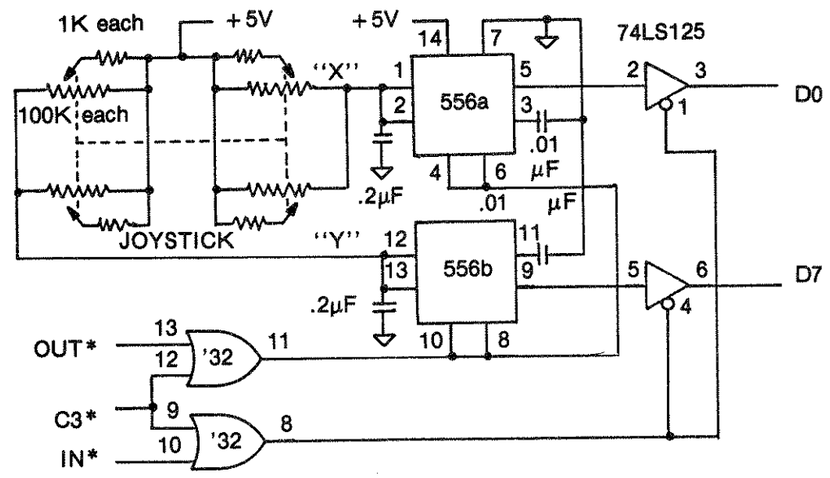


Figure 5.7 Experiment 5.1 Schematic.

the Y timer fell first, then the routine starting at 16547/32163 exchanges the contents of BC and HL before returning to the BASIC program.

Load the machine language routine by ENTERing RUN 100 and supplying the decimal values of the machine code.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16514 / 32130	211	OUT (N),A	:Trigger the monostables
16515 / 32131	3	(N)	:at Port 3.
16516 / 32132	30	LD, E,N	:Set the mask for both
16517 / 32133	129	N	:D7 and D0 to logic 1.
16518 / 32134	33	LD HL,NN	:HL points to address at
16519 / 32135	176	Lo N	:16560 or 32176 to tell which
16520 /	64	Hi N	:timer quit first.
/ 32136	125	Hi N	
16521 / 32137	1	LD BC,NN	:Zero the 16 bit counter
16522 / 32138	0	N	
16523 / 32139	0	N	
16524 / 32140	219	IN A,(N)	:Read the monostable levels
16525 / 32141	3	(N)	:from Port 3.
16526 / 32142	163	AND E	:Set the Flag register for A.
16527 / 32143	40	JR Z,d	:If both timed out then jump

16528 / 32144	12	d	:to 16541 / 32157.
16529 / 32145	3	INC BC	:Increment the count.
16530 / 32146	187	CP E	:Are both still high?
16531 / 32147	40	JR Z,d	:Yes. Jump back to 16524 /
16532 / 32148	247	d	:32140 and keep counting.
16533 / 32149	95	LD E,A	:No. Save a new mask
16534 / 32150	119	LD (HL),A	:to show which one quit.
16535 / 32151	35	INC HL	:Save the count low byte
16536 / 32152	113	LD (HL),C	:at 16561 / 32177, and
16537 / 32153	35	INC HL	:the high byte at 16562 /
16538 / 32154	112	LD (HL),B	:32178 for BASIC program.
16539 / 32155	24	JR d	:Jump back to 16524 / 32140
16540 / 32156	239	d	:keep count of second timer.
16541 / 32157	43	DEC HL	:Arrive here when both timers
16542 / 32158	43	DEC HL	:have quit.
16543 / 32159	126	LD A,HL	:Get mask from 16560 / 32177.
16544 / 32160	31	RRC	:Did D0(X) quit before D7(Y)?
16545 / 32161	56	JR C,d	:If not, then skip to 16557 /
16546 / 32162	10	d	:32163 to return to BASIC.
16547 / 32163	42	LD HL,(NN)	:If so, then Load HL with X
16548 / 32164	177	Lo N	:count stored at
16549 /	64	Hi N	:16561-2 or
/ 32165	125	Hi N	:32177-8.
16550 / 32166	229	PUSH HL	:Swap BC and HL by reversing
16551 / 32167	197	PUSH BC	:them on the stack.
16552 / 32168	225	POP HL	:Now HL has the Y count,
16553 / 32169	193	POP BC	:and BC has the X count.
16554 / 32170	34	LD (NN),HL	:Save the X count
16555 / 32171	177	Lo N	:low byte at 16561 / 32177
16556 /	64	Hi N	:high byte at 16562
/ 32172	125	Hi N	:or 32178.
16557 / 32173	201	RET	:Back to BASIC.

STEP 5 RUN the program. If your Timex / Sinclair responds by printing B / (line number), you will need to change the value or values of the divisors in line 60 and / or 70 of your BASIC program, because the values to be PLOTted are out of range.

STEP 6 If your variable values are not too large, a dot should appear in the center of your screen. Continue your program by putting it in a BASIC loop with a time delay of a second by changing line 50:

```
50 LET N = 80
```

Trace out the limits of your joystick control in the X and Y directions. A rectangle should result.

STEP 7 Investigate the nonlinearities of the diagonals. These are due to the logarithmic variation of most commonly available potentiometers.

SUMMARY A joystick control has been constructed using a minimum of inexpensive components. Alternate applications include varying resistances along the X and Y directions of a plane such as found in potentiometric X–Y flat bed recorders, or a pantograph arrangement for digitizing from a two-dimensional plane. Other extensions of the use of this control are left to the imaginations of the readers.

EXPERIMENT 5.2

DETECTION OF ROTATIONAL SPEED

COMPONENTS
- 1 * Slotted optical limit switch, type OPB861 (TRW)
- 1 * Opaque disc (see Figure 5.8)
- 1 * 74LS32 Quad OR Gate
- 1 * 74LS244 Octal buffer
- 1 * DAC558 Digital-to-Analog Converter
- 1 * LM358 Dual Operational Amplifier
- 1 * Transistor, type D40K (G.E.) or equivalent
- 1 * 150-ohm resistor
- 1 * 10-Kohm resistor
- 1 * 3-V permanent magnet DC motor
- 1 * 3-V DC power supply or batteries (Do not use the Timex / Sinclair power supply for driving the motor.)

DISCUSSION Rotational speed measurement is a useful measurement that can be made digitally. It requires a minimum of hardware coupled with the versatility of the microcomputer to provide rapid acquisition of rotor speed. Applications include both RPM (revolutions per minute) detection as in the case of a motor, or measurement of linear flow with a turbine, such as an anemometer for wind speed measurement. An alternate method for measuring the speed of a DC motor will be studied in Experiment 6.4.

The speed of the motor can be controlled within certain limits by the voltage applied to the motor. The voltage applied to the motor can, in turn, be controlled by the microcomputer using a digital-to-analog converter (DAC) to convert an eight-bit number to a proportional analog voltage. The analog voltage output of the DAC cannot drive the motor directly because it cannot provide sufficient current to the motor windings. However, by using an operational amplifier (op amp) to control a high gain power transistor between the DAC and the motor, the voltage of the motor's power supply can be controlled and the speed can be varied.

An optical sensor using an encapsulated infrared active LED and phototransistor separated by a narrow air gap or slot and shown in Figure 5.8, will be used to measure the speed of the motor. The detector has four leads to power the LED and pick up the current output from the phototransistor, these connections are shown in the schematic, Figure 5.10. When a thin cardboard or plastic disc having a small notch or gap in its perimeter is mounted on the rotor shaft and inserted into the slot of the detector, the phototransistor will conduct current only when the notch is aligned with the detector light beam. As the rotor turns, the disc will rotate, and, when the opaque section covers the LED, the phototransistor will turn off.

In this experiment we will use the microcomputer to control the speed of a small DC motor with the DAC driver and to time its interval of rotation with a phototransistor detector. A look at the waveform produced by the detector will assist in the description of just how the measurement will

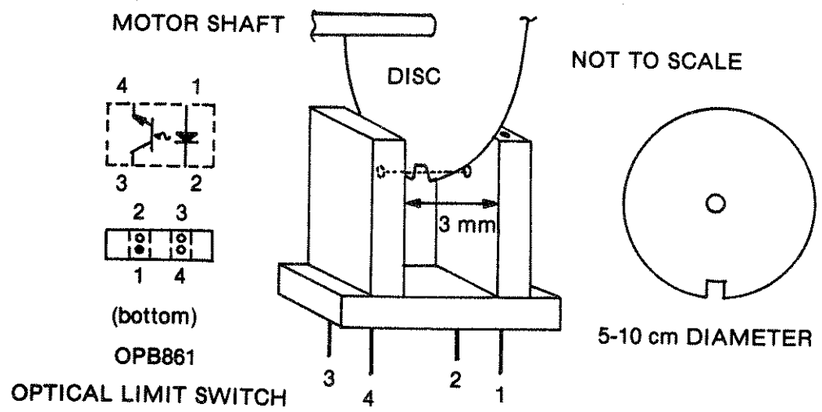


Figure 5.8 Diagram of a Detector.

be made. As the phototransistor turns on and conducts current, a voltage will be produced as shown in Figure 5.9. Obviously the time for one revolution is the time interval from one "on" time to the next succeeding "on" time. Once the time for one revolution, called the *period*, has been determined, the number of revolutions per minute (RPM) can be determined from the formula

$$RPM = 60/T$$

where T is the time for one revolution measured in seconds. For example, if T was measured to be 10 msec, then

$$RPM = 60/0.010 = 6000 \text{ RPM}$$

In order for the microcomputer to know when a period has elapsed it must first detect a rising pulse "turn on." Remember, the microcomputer is not synchronized to the speed of the motor so it

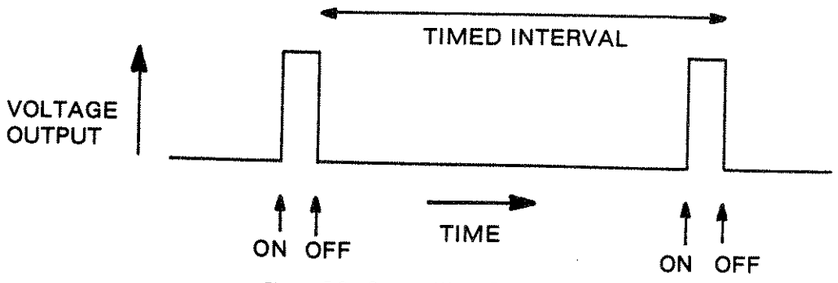


Figure 5.9 Square Wave Diagram.

might turn on its measurement program in the middle of the transparent section. To correct for this, we will cause the microcomputer to detect a rising edge and then go and time the interval between two successive falling edges. This will give us a reproducible value for the period T.

It would not be unusual, however, to expect variations in the speed of the motor, so one single value measured by the microcomputer might not be representative of the speed. This can be remedied by taking ten samples of the period and calculating the average period using a BASIC program.

The scheme then is to set the speed of the motor and start the microcomputer timing program, time the next period between falling edges, then jump back to BASIC to calculate the average period and print out a result. The ten values of period measured will be stored as a decimal count in the memory area of the REM statement allocated to the file of data values. These addresses can be stored in the HL register pair; the number of periods to be measured can be stored in the B register, and the counter used to time the period will be the count stored in the DE register pair.

To detect a rising edge, the pulse produced by the photodetector is fed to data bus line D7 via a three-state buffer which is enabled by the IN 3 device code. The state of D7, once input to the accumulator register of the Z80 microprocessor, can easily be checked for a high or low state by rotating the accumulator to the left into the carry flag. If D7 were a 1 (the "turned on" state) when input, then when the accumulator was rotated left, a 1 would have been moved out of bit 7 into the carry flag of the status register. (Recall that the carry flag is the LSB of the Flags register and so acts like a ninth bit of the accumulator.)

If the microprocessor now checks for a logic 1 it will know when bit 7 went high and immediately trigger the timing circuit to start on the next falling pulse, that is, when bit 7 falls to 0. The program will then wait until the next succeeding falling pulse before stopping the counter and transferring the count values into the file space of memory. The BASIC program can pick up the count values from memory and operate on them to produce the final result.

PROCEDURE

STEP 1 Wire the circuit with the power switched off, as shown in the schematic, Figure 5.10. Position the optical limit switch at the end of the socket board. The LM358 operational amplifier is connected as a "voltage follower" circuit. In this configuration, the DAC output feeds the noninverting (+) input of the op amp which in turn applies a voltage to the base of the (Darlington) D40K transistor driver. With the collector of the transistor connected to the 3-V power supply, the greater the voltage applied to the base of the transistor, the greater the amount of current that can flow from the collector (C) to the emitter (E) of the transistor). The voltage at the emitter then feeds the DC motor but also feeds back to the inverting (−) input of the op amp. As a voltage follower, the op amp uses the feedback signal to maintain equal voltages at both its + and − inputs by constantly adjusting its output to the base of the transistor.

STEP 2 Carefully attach the disc to the motor spindle so that it is centrally mounted (not eccentrically) and with a minimum of wobble.

STEP 3 Mount the motor on the edge of a stable base board that can be elevated using a wood block to fit into the narrow gap of the detector. The base board should be long enough to be weighted down because the high speed of the motor will tend to vibrate and move the board. Make sure that the detector lies along a radius through the center of the motor spindle for the most reliable operation.

STEP 4 Connect a separate power supply to the motor or use two heavy duty low-voltage (3 V total) DC batteries, some of the small 3-V motors can draw up to 0.5 A. If batteries are used, they can be used to weigh down the motor baseboard.

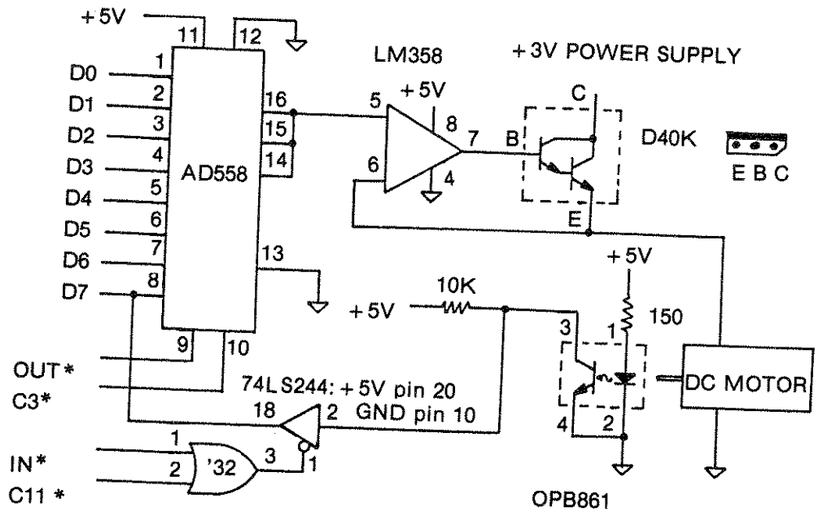


Figure 5.10 Experiment 5.2 Schematic.

STEP 5 Load the BASIC program into the Timex/Sinclair microcomputer. Line 30 of the BASIC program calls the subroutine at lines 150–190 to select the speed of the motor by outputting the variable V to the DAC at Port 11 with the USR function at address 16572/32188. The motor speed machine routine is called with the USR function at address 16514/32130 from line 40. The rest of the BASIC program averages the values stored by the microprocessor in memory locations 16552/32168 through to 16572/32188 and displays the results.

Ten pairs of bytes are held in consecutive memory locations 16552/32168–16571/32187, these are the 16-bit counts of the number of times the microprocessor went through the INCrement subroutine in the machine language program. Each increment takes 11.15 microseconds each time it is executed. Knowing this figure the BASIC program can calculate the RPM figure directly. Each pair of values is PEEKed by the BASIC program at line 70 and added to the sum S in line 80. This is repeated until all values are added in when the program moves onto line 100 and calculates the final average speed.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789 123456789
   123456789 12345
10 CLEAR 32129                                    (for B&W models)
20 FAST                                           (for Color models)
30 GOSUB 150                                      (for B&W models)
40 LET L = USR 16514
40 LET L = USR 32130                              (for B&W models)
50 LET S = 0                                      (for Color models)
60 FOR A = 16552 TO 16571 STEP 2
```



```
60 FOR A = 32168 TO 32187 STEP 2                  (for Color models)
70 LET C = PEEK A + 256 * PEEK(A + 1)
80 LET S = S + C
90 NEXT A
100 LET R = 6E8/(11.15*S)
110 PRINT V; "YIELDS SPEED = "; INT R; "R.P.M."
120 GOTO 20
150 PRINT "DAC VALUE = ";
160 INPUT V
170 POKE 16573,V                                  (for B&W models)
180 LET L = USR 16572
170 POKE 32189,V                                  (for Color models)
180 LET L = USR 32188
190 RETURN
200 FOR M = 16514 TO 16566                        (for B&W models)
200 FOR M = 32130 TO 32182                        (for Color models)
210 INPUT N
220 POKE M,N
230 PRINT M; "=" ;PEEK M,;
240 NEXT M
```

STEP 6 The machine language program itself was fairly well covered in the Discussion section of this experiment. Load the machine code by ENTERing RUN 200.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	133	LD HL,NN	:Point to start of counts
16515 / 32131	168	Lo N	:file at 16552/32168.
16516 /	64	Hi N	
/ 32132	125	Hi N	
16517 / 32133	6	LD B,N	:Number of file entries.
16518 / 32134	10	N	
16519 / 32135	17	LD DE,NN	:Zero time counter.
16520 / 32136	0	Lo N	
16521 / 32137	0	Hi N	
16522 / 32138	167	AND A	
16523 / 32139	219	IN A,(N)	:Read D7 bit from
16524 / 32140	3	N	:Port 3.
16525 / 32141	7	RLCA	:Shift it into Carry bit.
16526 / 32142	48	JR NC,d	:If light is out then jump back
16527 / 32143	251	d	:to 16523/32139, stay in loop.
16528 / 32144	219	IN A,(N)	:If light is on then read it
16529 / 32145	3	N	:from Port 3 again.
16530 / 32146	7	RLCA	:Shift D7 to the Carry bit.

```
16531 / 32147    56    JR C,d      :If light is still on then
16532 / 32148   251    d           :jump back to 16528/32144.
16533 / 32149    19    INC DE      :Arrive here on negative
16534 / 32150   219    IN A,(N)    :edge. Update count in DE
16535 / 32151     3    N           :and read D7 until
16536 / 32152     7    RLCA        :light goes on again
16537 / 32153    48    JR NC,d     :by jumping back to
16538 / 32154   250    d           :16533/32149.
16539 / 32155    19    INC DE      :Count while the light
16540 / 32156   219    IN A,(N)    :remains on to get the
16541 / 32157     3    N           :count of a complete
16542 / 32158     7    RLCA        :revolution by jumping
16543 / 32159    56    JR C,d      :back to 16539/32155.
16544 / 32160   250    d
16545 / 32161   115    LD (HL),E   :Done. Save the count low byte
16546 / 32162    35    INC HL      :in the memory file,
16547 / 32163   114    LD (HL),D   :and the count high byte
16548 / 32164    35    INC HL      :in the adjacent location.
16549 / 32165    16    DJNZ d      :Have all 10 counts been made?
16550 / 32166   224    d           :No. Go to 16519/32135.
16551 / 32167   201    RET         :Yes. Go back to BASIC.
16552 / 32168     0    NOP         :20 byte Data Table.
      to          :      :
16571 / 32187     0    NOP         :End of Table.
16572 / 32188    62    LD A,N      :Load motor speed value
16573 / 32189     0    N           :obtained from BASIC
16574 / 32190   211    OUT (N),A   :into DAC register
16575 / 32191    11    N           :at Port 11.
16576 / 32192   201    RET         :Back to BASIC subroutine.
```

STEP 7 Double check your machine code and hardware electrical connections then RUN your program. Enter a DAC value of 96. You may have to help the motor start running by rotating the disc with your finger.

STEP 8 Successful measurement of the speed of the motor will be indicated by a sensible answer printed on the video screen. If you have alignment problems with your disc in the narrow gap, it is possible to produce a number of spurious pulses each cycle (or period) giving rise to a higher speed than expected and a nonsensible result. If you suspect spurious pulses, carefully adjust the physical location of your detector with respect to the rotating disc. You should be able to adjust out any inconsistencies.

STEP 9 Check the linearity of your motor to applied DC voltage. Figure 5.11 shows the results we obtained by measuring the output of the DAC (curve A), the output of the op amp (curve B), and the motor speed (curve C) as a function of the number outputted to the DAC. Notice that curves B and C are straight lines over most of the range of values. We made four measurements per DAC value at intervals of 32 between 64 and 255. Above V = 192 the speed tends to flatten. Below V = 64 our motor tended to run in spurts, and sensible measurements could not be taken.

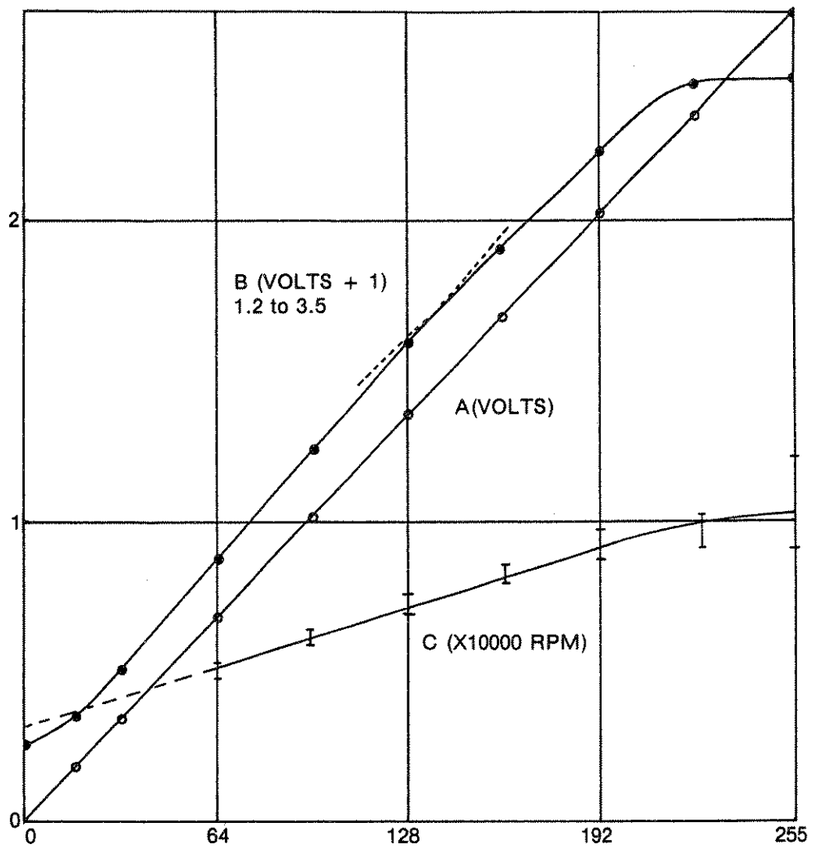


Figure 5.11 Analog Results.

STEP 10 Save the results of this experiment for Experiment 6.4 in which you could make use of the speed versus applied voltage relationship obtained here.

SUMMARY The determination and control of rotational speed are very useful measurements in science and engineering applications. The correspondence between the application in this experiment and Experiment 5.1 illustrates how different transducers can be implemented using the same digital timing technique. In this experiment, we have illustrated both digital output for analog conversion and digital input. For lower speeds the transparent and opaque disc sections can be proportioned differently or several sections of each can be laid out on the disc. Experiment 5.3 illustrates additional applications of this technique for digital data acquisition.

EXPERIMENT 5.3
ROTATIONAL POSITION DETECTION: SHAFT ENCODING

COMPONENTS
- 3 * Slotted optical limit switch, type OPB861 (TRW)
- 3 * Transparent plastic discs (see Figure 5.12)
- 1 * 74LS32 Quad Two-Input OR Gate
- 1 * 74LS244 Octal Three-state Buffer
- 3 * 330-ohm resistors
- 3 * 10-Kohm resistors

DISCUSSION The rotational or angular position of a shaft is often useful or necessary information in an experiment. Some examples are the determination of wind direction with a weather vane, the position of a valve, or the setting of a control knob. The determination of an angular position can be made digitally with optical sensors and sectioned transparent discs mounted on the shaft to be measured. Because the precision of the position doubles with each additional bit input, the maximum number of bits for an eight-bit input port would limit the precision to 360/256 or 1.4 degrees. Of course, higher precision would be possible using more bits and more than one input port to read the position. In many cases, such high precision is not needed. For example, the eight major compass directions of a weather vane require only three bits.

The design of the three discs and their alignment is shown in Figure 5.12. Because they are mounted on a common shaft, reading the bit pattern for a particular direction is done by reading a logic 1 if the disc is transparent in that direction or reading a logic 0 if it is opaque in that direction. One important factor in encoding direction is how to detect the positions reliably when one bit is uncertain. If we write the bit pattern for the eight directions by referring to Figure 5.12, the following table is obtained:

	A	B	C
N	0	0	0
NE	1	0	0
E	1	0	1
SE	1	1	1
S	1	1	0
SW	0	1	0
W	0	1	1
NW	0	0	1
N	0	0	0

where the north direction (N) is repeated to make it easier to compare codes for adjacent directions. The first observation we make is that the normal sequence of binary counting is not obtained. The encoding obtained is known as the Grey code. Its distinctive difference is that only one bit changes its value between adjacent directions. Therefore, if the weather vane should point exactly between W and NW, for example, only the middle bit would be uncertain. Whether the optical sensor detected a logic 1 or logic 0 for the middle bit would still let the computer decode the direction within the precision of one-eighth of the circle.

The circuit to interface the shaft encoder to the microcomputer is very simple. The collectors of the three sensors are each connected to a three-state buffer. The three-state buffers are enabled by Device Select Pulse (IN 3)* obtained by ORing IN* and Device Code 3. The emitters of the

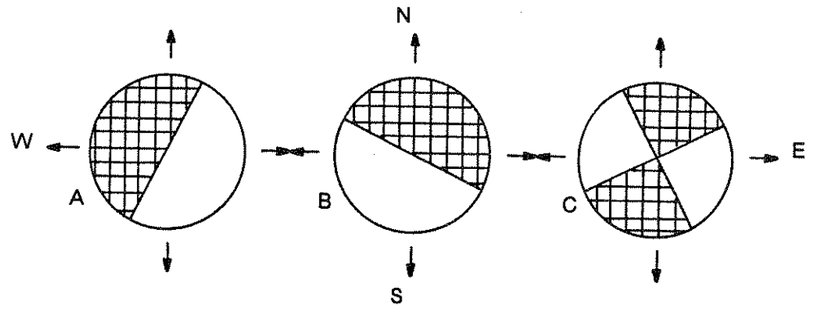


Figure 5.12 Disc Patterns for Eight Directions.

sensors are grounded, and their LEDs are connected to the power supply through current limiting resistors. The circuit is shown in Figure 5.13.

PROCEDURE

STEP 1 Prepare three 2-inch (50 mm) diameter discs from thin (1/16 inch or less) clear plastic sheet either by removing the protective paper for the transparent sections or by masking (tape or paint) each opaque section as shown in Figure 5.12. Align them on the shaft to be encoded.

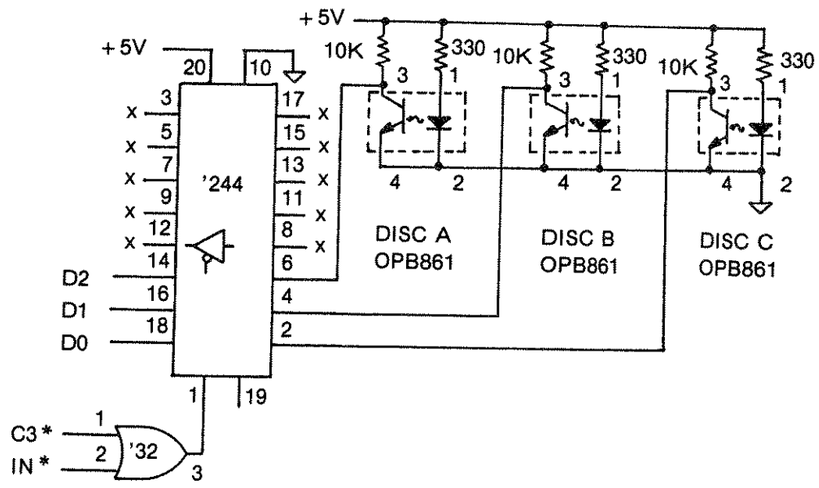


Figure 5.13 Experiment 5.3 Schematic.

A shaft made from a 1/8 inch (3 mm) × 2 (or 3) inch machine screw using washers and machine nuts against both faces of each disc will provide firm mounting and allow for easy alignment. This shaft can then be coupled to any shaft of interest.

STEP 2 The sensors can be mounted around the circumference of the discs at any position as long as the orientation of each disc to its sensor is as shown in Figure 5.12. For example, a box or cylinder housing with the sensors at positions of 3, 6, and 9 o'clock (right angles) can be used. The ends of the housing can then serve as shaft bearings if necessary.

STEP 3 With the +5-V power rail disconnected from the supply, wire the circuit as shown in the schematic.

STEP 4 The program to acquire the data is very straightforward. The position code obtained from the least three bits of Input Port 3 are returned from the USR subroutine and assigned to the variable P in line 20. Line 30 is the method to branch in Sinclair BASIC based on the value of P. Standard BASIC uses the command: ON P GOTO 50,60, . . . etc. The PAUSE at line 120 times readings about once every 5 seconds. Twenty-two readings will be obtained before you will have to input CONTinue. You could insert a line 25 SCROLL if desired to make continuous readings.

The machine language subroutine is equally simple. Because the value of the BC register pair is carried back to BASIC, it is initialized to 0. The input port is read and ANDed with 00000111 to zero the five most significant bits of the value read into the accumulator. The value of the least three bits are loaded into register C (B will be 0) and returned as the BASIC variable P.

Load the BASIC program. RUN 200 and enter the decimal code of the machine language routine.

BASIC PROGRAM

```
10 REM 123456789                    (for B&W models)
10 CLEAR 32129                      (for Color models)
20 LET P = USR 16514                (for B&W models)
20 LET P = USR 32130                (for Color models)
30 GOTO 40+10*P
40 PRINT  "NORTH"
45 GOTO 120
50 PRINT  "NORTHWEST"
55 GOTO 120
60 PRINT  "SOUTHWEST"
65 GOTO 120
70 PRINT  "WEST"
75 GOTO 120
80 PRINT  "NORTHEAST"
85 GOTO 120
90 PRINT  "EAST"
95 GOTO 120
100 PRINT  "SOUTH''
105 GOTO 120
110 PRINT  "SOUTHEAST"
120 PAUSE 300
```



```
130 GOTO 20
200 FOR M = 16514 TO 16522          (for B&W models)
200 FOR M = 32130 TO 32138          (for Color models)
210 INPUT N
220 POKE M,N
230 PRINT M;  "=" ; PEEK M,;
240 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	1	LD BC,NN	:Zero registers for BASIC
16515 / 32131	0	Lo N	:variable P.
16516 / 32132	0	Hi N	
16517 / 32133	219	IN A,(N)	:Read Optical sensors
16518 / 32134	3	N	:at Port 3.
16519 / 32135	198	AND A,N	:Zero all bits but
16520 / 32136	7	N	:D2, D1, and D0.
16521 / 32137	79	LD C,A	:Put in register C.
16522 / 32138	201	RET	:Take it back to BASIC

STEP 5 Having checked the software and hardware (detector and circuit), RUN your program.

STEP 6 Casually rotate the shaft and check it out for proper encoding. Note the position of due north, etc.—you might mount a short pointer on the shaft made from a piece of breadboarding wire wrapped around the shaft.

STEP 7 Align one of the section boundary lines of one of the discs with its sensor. Rotate the disc just enough on either side of the boundary to read adjacent compass points. What directions would you obtain if the disc were encoded in binary instead of the Grey scale?

STEP 8 You may want to save this interface and use it with the stepper motor in Experiment 5.4. The combination of the two experiments is left to the reader. It would be a simple matter to build a wind speed anemometer using Experiment 5.2 and a wind direction indicator with Experiment 5.3.

EXPERIMENT 5.4

STEPPER MOTOR CONTROL

COMPONENTS
- 1 * Stepper motor [e.g., Quadrapulse 8AU0705 (Septor) (5 V, 0.85 A/coil, 48 steps/rev)]
- 1 * Power supply for coils of stepper motors
- 4 * Driver transistors [e.g., D40K (GE) (2 A, 30 V)]
- 1 * 74LS75 Quad Latch
- 1 * 74LS02 Quad NOR Gate

DISCUSSION Robotics are gaining in importance and the control of movement in robots is most often accomplished using stepper motors. Various stepper motors are available in the marketplace; we chose one which worked from 5 V and drew 500 mA per coil, a total of 1 A DC at 5 V, hence the need for a DC power supply to drive this part of the experiment.

We have already seen that the rotor of the motor is stepped by applying pulses to one of each pair of the four coils in the sequence 0110 and then rotating the sequence to the right or left by one bit, depending on which direction you wish the motor to rotate. You will note that the microprocessor instruction set contains an ideal instruction with which to carry out this procedure: the RRCA (Rotate Right the Contents of the Accumulator) or the RLCA instructions. Each time such an instruction is executed and output, the motor will step one position to the right or left.

The interface uses a 74LS75 four-bit latch to hold the four-bit word output from the accumulator register of the Z80 microprocessor. The latch is enabled by the device select pulse generated by the 74LS02 NOR gate. The latch drives the bases of the four transistors connected to the four coils of the stepper motor. When the latch output is low (0 state) the base of the transistor has insufficient bias to cause the transistor to conduct so the collector output of the transistor stays high at near 5 V so no (or very little) current is drawn through the coil. When the latch output is high (1 state) the base of the transistor is forward biased, and the transistor conducts so bringing the collector voltage down to near 0 V and placing nearly 5 V across the coil. The coils are activated and apply a magnetic force to the rotor which causes the motor to advance one step around.

Robot arms used to demonstrate the principles of robotics have at least six degrees of freedom. Each degree of freedom would have a stepper motor attached to control movement in that particular plane of motion, for example, arm rotation, elbow bending, wrist twisting. Each motor would have similar interface circuitry activated by different device select pulses. To put the robot arm in a particular position starting from a known reference position would require a sequence of instructions to each motor telling it how many steps it would have to take to reach the designated position.

In this experiment you will accomplish the setting of the stepper motor at a particular position, but once again you will be able to visualize more complex situations.

PROCEDURE

STEP 1 You will have previously constructed the transistor driver circuitry needed to drive the stepper motor. This circuitry will depend somewhat on the specifications of the stepper motor you purchase or have available. Most power transistors however can have their bases driven directly by the output of a TTL latch chip if not by an LS latch chip. So even if you have 24-V coils (and need a 24-V supply) on your stepper motor, your driving transistors could be driven from the latch chip.

STEP 2 Wire the circuit as shown in the schematic, Figure 5.14, making certain that the positive (+) volt line from your motor DC power supply is *not* connected to the +5-V line of the Timex/Sinclair unit. *Do* however make sure that a GND, 0 V, line is connected between the two supplies. Wire your 74LS02 quad NOR gate to produce the device select pulse OUT 3 (an active high device select pulse).

STEP 3 Insert your program, and check both hardware connections and the software program.

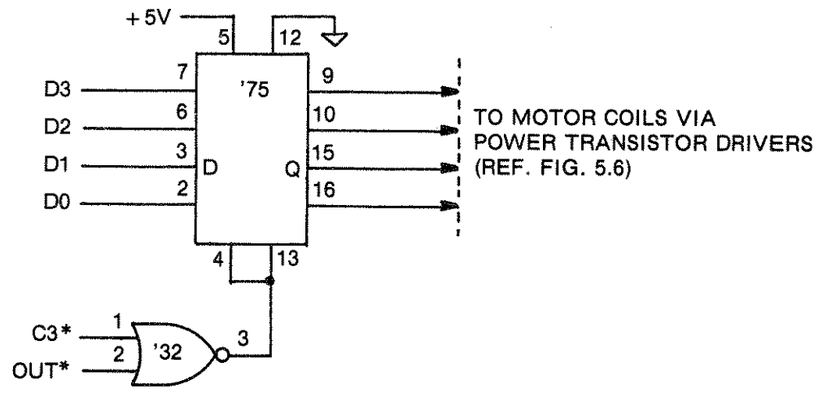


Figure 5.14 Experiment 5.4 Schematic.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 1234567        (for B&W models)
10 CLEAR 32129                                       (for Color models)
20 FAST                                              (for B&W models)
30 LET L = USR 16515                                 (for B&W models)
30 LET L = USR 32131                                 (for Color models)
40 PRINT AT 14,0;  "ENTER NUMBER OF STEPS REQUIRED"
50 INPUT DC
    * * * *
60 POKE 16514,DC                                     (for B&W models)
70 LET L = USR 16521
80 PRINT PEEK 16550
    * * * *
60 POKE 32130,DC                                     (for Color models)
70 LET L = USR 32137
80 PRINT PEEK 32166
    * * * *
90 LET N = 33333
100 PAUSE N
110 GOTO 40
120 FOR M = 16515 TO 16548                           (for B&W models)
120 FOR M = 32131 TO 32164                           (for Color models)
```

```
130 INPUT N
140 POKE M,N
150 PRINT M;  "=" ; PEEK M,;
160 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16514 / 32130	< >		:No. steps POKEd by BASIC
16515 / 32131	33	LD HL,NN	:Address pointer of bit
16516 / 32132	166	Lo N	:pattern mask stored
16517 /	64	Hi N	:at 16550.
/ 32133	125	Hi N	:or 32166.
16518 / 32134	54	LD (HL),N	:Load mask at 16550/32166.
16519 / 32135	51	N	:Mask = 00110011.
16520 / 32136	201	RET	:Back to BASIC line 40.
16521 / 32137	33	LD HL,NN	:Address points to mask
16522 / 32138	166	Lo N	:at 16550/32166.
16523 /	64	Hi N	
/ 32139	125	Hi N	
16524 / 32140	58	LD A,(NN)	:Get number of steps
16525 / 32141	130	Lo N	:at address 16514/32130.
16526 /	64	Hi N	
/ 32142	125	Hi N	
16527 / 32143	50	LD (NN),A	:Put number of steps into
16528 / 32144	147		:subroutine at 16531/32147
16529	64	Hi N	
/ 32145	125	Hi N	
16530 / 32146	6	LD B,N	:Load number of steps into B.
16531 / 32147	0	N	:Loaded from 16527/32143.
16532 / 32148	126	LD A,(HL)	:Put mask into accumulator.
16533 / 32149	17	LD DE,NN	:Set up delay counters
16534 / 32150	255	Lo N	:in register E
16535 / 32151	100	Hi N	:and register D.
16536 / 32152	211	OUT (N),A	:Send bit pattern to
16537 / 32153	3	N	:stepper at Port 3.
16538 / 32154	29	DEC E	:Countdown E
16539 / 32155	32	JR NZ,d	:Is E zero?
16540 / 32156	253	d	:No, go back to 16538/32154.
16541 / 32157	21	DEC D	:Yes, now countdown D.
16542 / 32158	32	JR NZ,d	:Is D zero?
16543 / 32159	250	d	:No, then countdown E again.
16544 / 32160	15	RRCA	:Rotate mask for next step.
16545 / 32161	16	DJNZ d	:Are all steps in taken?
16546 / 32162	242	d	:No, goto back to 16533.
16547 / 32163	119	LD (HL),A	:Yes, then save current mask
16548 / 32164	201	RET	:and go back to BASIC.

16549 / 32165	0	NOP
16550 / 32166	< >	MASK

STEP 4 Switch on your DC power supply to your stepper motor, and check that the motor is held. The rotor should resist your attempt to move it. Next RUN your program, and input how many steps you wish the motor to take. Try two or three steps first.

STEP 5 The program accepts this input and pokes the decimal value into memory location 16514/32130 where the machine language program collects it and transfers it to register B, which is used as a counter for the number of steps. The machine language program outputs the code 51 decimal, 00110011 in binary, which is the sequence of bits required by the motor coils. The program then executes a time delay before rotating the accumulator contents to the right at memory location 16544 and outputting another step. This carries on until register B is 0 whereupon the Z80 instruction DJNZ is executed and the code of the last step is stored in memory before the program returns to BASIC to wait on another entry.

STEP 6 The next entry picks up the previously stored code for the position of the motor and starts stepping from the last remembered position.

STEP 7 Change the code of the instruction RRCA to that for RLCA at memory location 16544/32150, and note that the motor steps in the reverse direction.

STEP 8 Determine how many steps per revolution for your stepper motor. Our motor had 48 steps per revolution. You might observe at the first run that the motor is unsure which direction to step. This is due to the code 51 decimal being used. The motor at switch on may not be lined up for this particular code; however, the motor is brought into synchronization quickly and then continues stepping in the correct direction.

STEP 9 Any unusual problems, such as missing out a step when rotating, could be due to a faulty drive transistor. These can be checked by connecting a voltmeter to the collector of each transistor in turn and noting as you step through at least four steps slowly, that the collector should go down to near 0 V as well as come back up to near 5 V. If one or more do not show this behavior, they should be replaced and the motor coils should be checked as well.

SUMMARY Stepper motor control is an important interfacing experiment, but it requires more than the basic equipment most of our experiments have utilized. This experiment also demonstrates the need for using discrete components, the transistors, to control power devices which draw heavy currents of 0.5 A or more. Much interfacing requires this level of sophistication.

EXPERIMENT 5.5

REAL TIME DIGITAL CLOCK

COMPONENTS
- 1 * 58167 Digital Clock IC
- 1 * 32.768-KHz miniature crystal
- 1 * 74LS02 Quad Two-Input NOR Gate
- 1 * 74LS373 Three-state Octal Latch
- 2 * 20-pF capacitors (preferably polystyrene)
- 1 * 0.1-μF capacitor

DISCUSSION In many experiments, especially ones which monitor slowly changing data continually or over relatively long periods of time, it is desirable to acquire the data as a function of real time. By *real time* we mean actual clock time or time of day. When we refer to slowly changing data, we mean slowly with respect to the speed of the microprocessor, so readings of once per second or longer might be considered slow. For example, monitoring temperature, pressure, or wind velocity and direction does not typically require data acquisition more than once every 15 sec to provide more than enough information to produce essentially continuous data. With a real-time clock we also have the option of using the computer as either a stopwatch or lapse timer. Many experiments can be designed where the start and stop controls for some event can be performed by triggering the computer to read a real time and calculate the difference. Accuracies of 0.01 seconds in BASIC programs or even milliseconds in machine language should be achievable.

The integrated circuits used in the manufacture of digital clocks can be interfaced to the computer to provide time readings however there are also microprocessor-compatible clock ICs, which are very easy to interface. The MM58167 is a 24-pin CMOS IC (National Semiconductor) typical of the latter. Its timing is controlled by a quartz crystal. It is a calendar clock made up of eight counter registers which keep the month (1 to 12), day of month (1 to 28, 30, or 31 depending on month), day of week (1 to 7), hours (0 to 23), minutes (0 to 59), seconds (0 to 59), fractions of seconds (0.00 to 0.99), and milliseconds (0 to 9). There are also eight presettable latches corresponding to the eight counters which can be used for alarm-type functions. The chip interfaces eight data lines directly to the Data Bus of the microcomputer. To access any one of the 16 counter or latch registers, there are five address input lines which would ordinarily be connected to the computer's Address Bus. Each counter and latch is thus treated as a separate input/output port. There are also eight additional control registers, which function as individual ports to bring the total count up to 24. The Device Codes (Port Addresses) are given in Table 5.2. We will confine our attention to only those ports related to the counters and a few of the controls. More advanced interfacing of the alarm latch ports can be performed as an advanced project using the manufacturer's data sheet.

The data byte transferred between the clock and the microprocessor is encoded in BCD (binary-coded decimal). The more significant four bits (D7–D4) contain the tens and the less significant four bits (D3–D0) contain the ones of the time unit. For example, 4 PM held in the hours register (counter or latch) is 16 (= 12 + 4) and is encoded in BCD as 0001 (value 1) and 0110 (value 6) giving 00010110 as the byte with the decimal value (for BCD 16) of 22. The number 22 would be output to Clock Port 4 (hours counter) to set the time for 4 PM. As another example, the largest value in the minutes and seconds registers would be 59: the byte value for BCD 59 is 01011001 or 89 (64 + 16 + 8 + 1).

Because the Timex/Sinclair decodes the address bus lines relatively, we cannot interface the MM58167 directly to the Address Bus. As shown in Figure 5.15, the address ports can be selected with five latches wired as an output port. The clock is interfaced as two separate ports; the 74LS373 latch will be the address port that selects the desired clock register as the first port. The data transfer either to or from the addressed clock register is the second port. In programming the computer to read (input) or set (output) the calendar/clock registers, we will first output to the clock address port the code to select a register, and then input/output the data from/to the clock data port.

TABLE 5.2 CLOCK DEVICE CODES

PORT ADDRESS					DEVICE CODE	FUNCTION
A4	A3	A2	A1	A0	Decimal	
0	0	0	0	0	0	Counter: milliseconds
0	0	0	0	1	1	Counter: 0.XX seconds
0	0	0	1	0	2	Counter: seconds
0	0	0	1	1	3	Counter: minutes
0	0	1	0	0	4	Counter: hours
0	0	1	0	1	5	Counter: day of week
0	0	1	1	0	6	Counter: day of month
0	0	1	1	1	7	Counter: month
0	1	0	0	0	8	Latch: milliseconds
0	1	0	0	1	9	Latch: 0.XX seconds
0	1	0	1	0	10	Latch: seconds
0	1	0	1	1	11	Latch: minutes
0	1	1	0	0	12	Latch: hours
0	1	1	0	1	13	Latch: day of week
0	1	1	1	0	14	Latch: day of month
0	1	1	1	1	15	Latch: month
1	0	0	0	0	16	Control: alarm status (Input)
1	0	0	0	1	17	Control: alarm mask (Output)
1	0	0	1	0	18	Control: reset counter select
1	0	0	1	1	19	Control: reset latch select
1	0	1	0	0	20	Control: counter read status
1	0	1	0	1	21	Control: "GO" command
1	0	1	1	0	22	Control: Standby
1	1	1	1	1	31	Control: Test

PROCEDURE

STEP 1 **WARNING:** When handling the CMOS clock chip, be careful to prevent static discharge. Before applying power to your circuit, make absolutely certain that all input pins are connected. Failure to observe either of these precautions may result in permanent damage to the IC.

STEP 2 With the +5-V power rail disconnected, wire the circuit as shown in the schematic.

STEP 3 We shall use several short BASIC and machine language programs to test the action of the calendar/clock. Our first interest is to test whether the clock is running. To do this, the BASIC program will read the seconds register and display the reading on the screen. The value of the register will be held in the USR variable, L, as two BCD digits. Lines 30 and 40 convert the byte to the tens and units values, T and U respectively. Recall that the two four-bit values are equivalent to a hexadecimal number, hence the integer value of L/16 yields the T value and the remainder, L − 16*T, yields the U value. Load the following BASIC program.

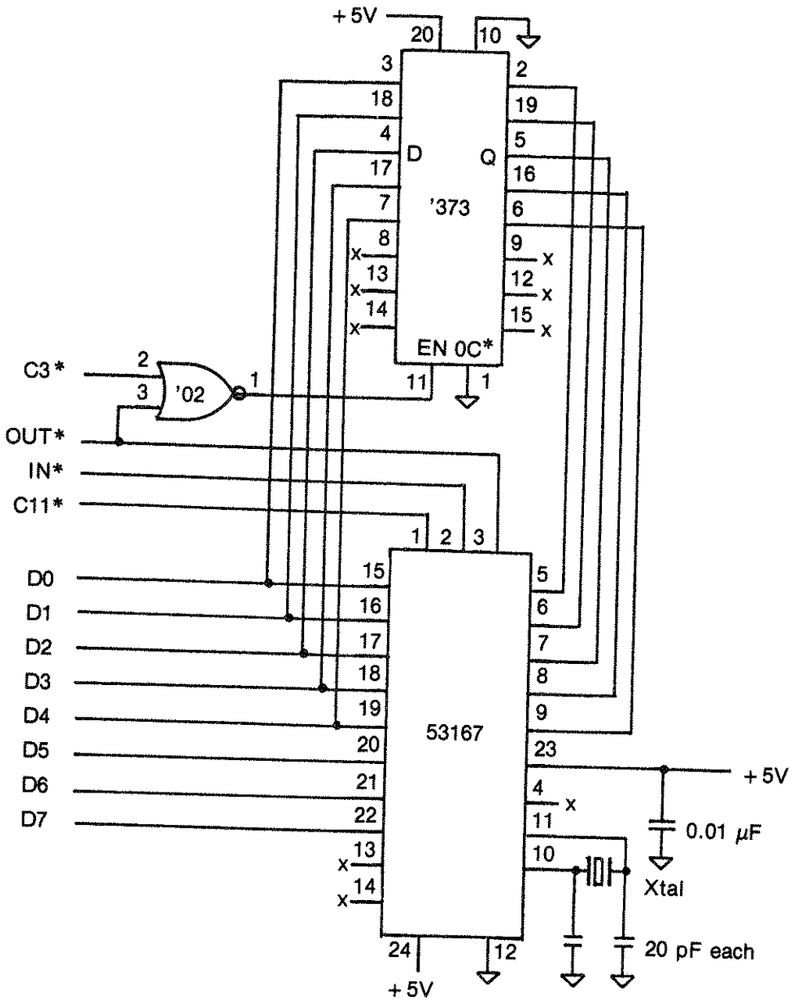


Figure 5.15 Experiment 5.5 Schematic.

BASIC PROGRAM

```
 10 REM 123456789 123456789 123456          (for B&W models)
 10 CLEAR 32129                             (for Color models)
 20 LET L = USR 16514                       (for B&W models)
 20 LET L = USR 32130                       (for Color models)
 30 LET T = INT(L/16)
 40 LET U = L - 16*T
 50 PRINT T;"   ";U
 60 GOTO 20
100 FOR M = 16514 TO 16528                  (for B&W models)
100 FOR M = 32130 TO 32144                  (for Color models)
110 INPUT N
120 POKE M,N
130 PRINT M; "=" ;PEEK M
140 NEXT M
```

STEP 4 The machine language subroutine loads the clock address port, Device Code 3, with Seconds Register address, Address 2, and then inputs the value from the seconds counter into the microprocessor's accumulator. The value is transferred to the C register after zeroing the B register for return to BASIC. Use RUN 100 to load the machine code for the subroutine.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	62	LD A,N	:Load Seconds Counter address
16515 / 32131	2	N	:into Accumulator,
16516 / 32132	211	OUT (N),A	:then load it into the Latch
16517 / 32133	3	N	:at Port 3.
16518 / 32134	219	IN A,(N)	:Read the counter value
16519 / 32135	11	N	:from Port 11.
16520 / 32136	6	LD B,N	:Zero the B register.
16521 / 32137	0	N	
16522 / 32138	79	LD C,A	:Put the counter value into C
16523 / 32139	201	RET	:Return BC value to BASIC.
16524 / 32140	62	LD A,N	:Load GO command register
16525 / 32141	21	N	
16526 / 32142	211	OUT (N),A	:to Port 3.
16527 / 32143	3	N	
16528 / 32144	201	RET	:Return to BASIC.

STEP 5 A second subroutine at addresses 16524/32140–16528/32144 was also loaded in Step 5. This subroutine uses the GO command register at address 21. By addressing this port, the clock registers for the seconds, fractions of seconds, and milliseconds are zeroed. Ordinarily

this subroutine would be used to synchronize the clock with real time. We use it here just to clear these counters.

STEP 6 Check your wiring once more, then connect power to the socket board. Now ENTER the direct command:

```
LET L = USR 16524        (B&W)
LET L = USR 32140        (Color)
```

to clear the seconds registers and then ENTER RUN.

STEP 7 The program should list the time readings until the display has filled the 22 lines of the screen. If you do not observe about one to ten readings per second, power down and check out the circuit and program. You can delete or insert REM at the beginning of lines 110 and 120 and RUN 100 to reread the machine code.

STEP 8 Enter the direct command FAST, and then RUN. The screen will blank and reappear in a few seconds with the next 22 readings. How many seconds did it take? Because BASIC runs about four times faster in FAST mode (in the B&W models), your list should contain about four readings per second.

STEP 9 Because both B and C are returned as a 16-bit value to the USR variable, we can revise the machine language subroutine to read and return two counter registers. By selecting the seconds fraction counter and the milliseconds counter at clock addresses 1 and 0, respectively, we can time how fast the BASIC routine takes to make a reading. RUN 100 (make sure lines 110 and 120 have been restored), and load the following code.

MACHINE LANGUAGE ROUTINE

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	62	LD A,N	:Load address for
16515 / 32131	1	N	:0.XX seconds counter
16516 / 32132	211	OUT (N),A	:to latch port.
16517 / 32133	3	N	
16518 / 32134	219	IN A,(N)	:Input 0.XX seconds
16519 / 32135	11	N	:counter from clock.
16520 / 32136	71	LD B,A	:Save it in register B.
16521 / 32137	62	LD A,N	:Load address for
16522 / 32138	0	N	:milliseconds counter
16523 / 32139	211	OUT (N),A	:to latch port.
16524 / 32140	3	N	
16525 / 32141	219	IN A,(N)	:Input 0.00X counter.
16526 / 32142	11	N	
16527 / 32143	79	LD C,A	:Save it in register C.
16528 / 32144	201	RET	:Pass BC back to BASIC.

STEP 10 To convert the value assigned to L into BCD digits, add the following lines to your BASIC program.

BASIC PROGRAM

```
25 LET B = INT(L/256)
30 LET C = L - 256*B
35 LET T - INT(B/16)
40 LET H = B – 16*T
45 LET M = INT(C/16)
50 LET U = C –16*M
55 PRINT  “.” ;T;H;M;U
```

Lines 25 and 30 split the 16-bit value of L back into the two bytes that were in registers B and C and assigns them to the BASIC variables B and C. The lines following then evaluate the BCD digits in each byte as before. The U variable should be 0 because the milliseconds counter only stores one digit.

STEP 11 If your computer is a B&W model, it should still be in FAST mode. Enter RUN. When the display reappears, the 22 values should all be different. Write down the list of numbers, and use your computer to take the difference between adjacent pairs by ENTERing PRINT (n2)–(n1), etc. Note that when the second number, (n2), is smaller than (n1), you have to add 1 to n2 to account for a rollover of the seconds counter. We found most of the differences were between 0.35 and 0.37 sec with the TS1000 but whenever there was a 0 in one of the digits the difference was 0.25. Apparently the BASIC interpreter processes a zero faster than a nonzero value. We can conclude that it takes 370 msec to execute the nine lines of the BASIC loop. An average difference of 0.08 sec on the TS2068 seems to indicate that the Color models run four times faster than the B&W models in FAST mode.

STEP 12 The final program will read all the counters of the calendar/clock, store the values in memory locations 16535/32151 to 16542/32158 at the end of the machine language routine, and display them each time any key (except BREAK) is pressed. ENTER the following BASIC program.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789
                                              (for B&W models)
20 LET L = USR 16514
30 FOR M = 16535 TO 16542
     * * * *
10 CLEAR 32129                                (for Color models)
20 LET L = USR 32130
30 FOR M = 32151 TO 32158
     * * * *
40 LET N = PEEK M
45 GOSUB 200
50 PRINT “ :” ;T;U;
```

```
 55 NEXT M
 60 PRINT
 65 PRINT "  M / D(W) H :M :S .XX XO"
 70 PAUSE 33333
 75 GOTO 20
100 FOR M = 16514 TO 16534                    (for B&W models)
100 FOR M = 32130 TO 32150                    (for Color models)
110 INPUT N
120 POKE M,N
130 PRINT M;" =" ;PEEK M
140 NEXT M
150 STOP
200 LET T = INT(N/16)
210 LET U = N - 16*T
220 RETURN
```

STEP 13 To read all eight counters, the machine code subroutine calls another subroutine. It uses register B as a countdown to know when all eight registers have been input. It also uses B to determine which register to input. Recall that the DJNZ d instruction decrements register B to test whether to jump or not. Load the following code by ENTERing RUN 100.

MACHINE LANGUAGE SUBROUTINE

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	33	LD HL,NN	:Point HL to start of
16515 / 32131	151	Lo N	:data file at memory
16516 / 32132	64	Hi N	:location 16535/32151
16517 / 32133	6	LD B,N	:Set B to count down
16518 / 32134	8	N	:the 8 counter registers.
16519 / 32135	205	CALL NN	:Subroutine to read and
16520 / 32136	141	Lo N	:store a counter register
16521 / 32137	64	Hi N	:at address 16525/32141.
16522 / 32138	16	DJNZ d	:All counters read?
16523 / 32139	251	d	:No. Jump back to 16519/32135.
16524 / 32140	201	RET	:Yes. Return to BASIC.
16525 / 32141	120	LD A,B	:Subroutine to read:
16526 / 32142	214	SUB N	:Clock register address
16527 / 32143	1	N	:equals B-1.
16528 / 32144	211	OUT (N),A	:Load clock address
16529 / 32145	3	N	:at latch port.
16530 / 32146	219	IN A,(N)	:Read data from clock
16531 / 32147	11	N	:at port 11.
16532 / 32148	119	LD (HL),A	:Store data in file.
16533 / 32149	44	INC L	:Point to next file entry.
16534 / 32150	201	RET	:Return to 16522/32138.

STEP 14 RUN the program. You should still be in FAST mode (on a B&W model) and should see a display of all eight counters. Of course, they have not been set to give the true time. A convenient way to load the counters is to add a small USR routine to the REM statement following the data file. Load the following machine code by modifying line 100 to:

```
100 FOR M = 16543 TO 16551          (for B&W models)
100 FOR M = 32159 TO 32167          (for Color models)
```

and then ENTERing RUN 100.

MACHINE LANGUAGE ROUTINE TO SET COUNTERS

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16543 / 32159	62	LD A,N	:Set up Clock address
16544 / 32160	7	N	:starting with Months.
16545 / 32161	211	OUT (N),A	:Latch address at
16546 / 32162	3	N	:port 3.
16547 / 32163	62	LD A,N	:Set up to load the counter
16548 / 32164	1	N	:counter with data.
16549 / 32165	211	OUT (N),A	:and output it to the
16550 / 32166	11	N	:clock at port 11.
16551 / 32167	201	RET	:Return to BASIC.

STEP 15 The counters can be individually loaded with the proper date and time by giving the direct commands:

```
POKE 16544, (Counter Address)          (B&W)
POKE 16548, (Data)
LET L = USR 16543
POKE 32160, (Counter Address)          (Color)
POKE 32164, (Data)
LET L = USE 32159
```

where the two quantities in parentheses will be numbers: the Counter Address value is obtained from Table 5.2 and the Data value must be determined according to the date you are setting. We need only set from the months to the minutes counters and then execute the GO command to synchronize to real time. Thus we need only load five counters. For example, suppose we wish to synchronize on Monday, November 14 at 10:22 AM; then the five pairs of POKEs will be:

REGISTER	MONTH	DATE	WEEKDAY	HOUR	MIN
(Counter Address)	7	6	5	4	3
Settings	NOV	14	MON	10	22
BCD	11	14	1	10	22
Binary	00010001	00010100	00000001	00010000	00100010
(Data)	17	20	1	16	35

To synchronize to real time, the address register of the GO command, 21, can be POKEd at 16544/32160. No data is required so we do not care what value is at 16548/32164. Type the USR call:

```
LET L = USR 16543        (B&W)
LET L = USR 32159        (Color)
```

and wait for the second hand of the clock you are using as a standard to reach the time you have set, then press ENTER. Now ENTER RUN. Each time you press a key, the date and time will be displayed.

EXPERIMENT 5.6
ASYNCHRONOUS SERIAL COMMUNICATION

COMPONENTS
- 1 * 8251 USART IC
- 1 * 556 Dual Timer IC
- 1 * 74LS00 Quad Two-Input NAND Gate
- 2 * 330-ohm resistors
- 2 * 15-Kohm resistors
- 2 * 0.1-μF capacitors
- 2 * 0.01-μF capacitors
- 1 * Lamp Monitor

DISCUSSION The Timex/Sinclair is a particularly good computer to use in a real measurement situation as a data acquisition and experiment control device because its modest cost permits dedicating it to a single instrument or experiment. An interface circuit can be developed for the particular apparatus and attached to the computer on a permanent (or semipermanent) basis. In such cases, a host computer would be available which has larger memory capacity, disk storage for data files, graphics and printer capabilities that would be too expensive to dedicate to one measurement apparatus. It is then highly desirable to be able to have the one or more dedicated microcomputers communicate with the host computer. We have seen that communication between computers up to 50 ft apart is easily achieved using a pair of wires in an RS-232 serial link.

In this experiment, we shall demonstrate how such communication is possible using a TTL compatible parallel/serial converter designed specifically for use with computers. Like the 8255 programmable peripheral interface studied in Chapter 4, the 8251 USART is also programmable in the sense that command control data can be sent to the chip to configure it for specific transmitting and receiving conditions. This is in contrast to the UART described earlier in this chapter, which had to be hardwired where the selection of number of stop bits, data word length, parity, etc. were pin inputs which had to have logic 1s or 0s wired to the chip.

The 8251 Programmable Communication Interface IC is a 28-pin TTL compatible Universal Synchronous/Asynchronous Receiver/Transmitter (USART). The pin-out diagram of the IC is shown in the schematic for the experiment, Figure 5.16. The USART is a clocked device requiring TTL level (+5 V) frequencies. There are three separate clock input pins: the transmitter clock, TxC* at pin 9, the receiver clock, RcC* at pin 25, and the internal clock, CLK at pin 20. Typically the transmitter and receiver clocks would be connected to the same frequency source. The

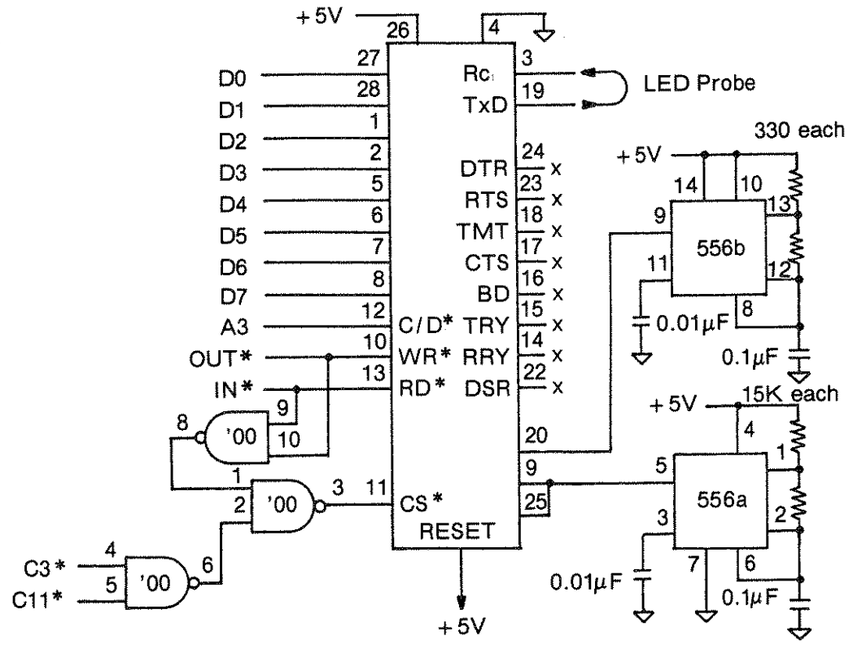


Figure 5.16 Experiment 5.6 Schematic.

frequency of the internal clock must be at least 30 times greater than the frequency (bit rate) of either the transmitter or receiver. There are eight data pins for parallel transfer of data between the chip and the computer, which are connected directly to the data bus of the computer, and two data pins for serial communication: the serial receiver, RcD at pin 3, and the serial transmitter, TxD at pin 19. The power to the IC is +5 V at V_{cc}, pin 25, and 0 V or ground at pin 4. There are eight additional control input or output pins whose use as various conditional flags are of more importance for synchronous, rather than asynchronous, communication. These are shown as not connected (nc) in Figure 5.16.

The five remaining pins of the 8251 are control pins. Four of these, WR*, RD*, CS* and C/D*, serve to select between the two I/O ports of the IC. One of these ports is used to write out a command instruction and to read in a status byte. The other port is used to load the data byte to the transmitter and to read the data byte from the receiver. WR* and RD* determine the direction of the transfer and are connected to OUT* and IN* of the computer's control bus, respectively. The C/D* input selects between the Command Port (logic 1 input) and the Data Port (logic 0 input). The Chip Select input, CS*, must be connected through the NAND gates shown in the schematic so that it is activated by any one of the four Device Select pulses: IN "Command Port," OUT "Command Port," IN "Data Port," OUT "Data Port." If we select device code 3 for the Data Port and device code 11 for the Command Port, then address line A3 can be used to activate the

C/D* pin because A3 is a logic 0 when device code 3 is present on the Address Bus and a logic 1 when device code 11 is present.

The RESET input at pin 21 is active high. Ordinarily it is connected to 0 V except when the circuit is first powered. Then it must be activated by raising the pin to +5 V momentarily. When the 8251 is reset, it takes the first command output to the Command Port to be a Mode Command. All subsequent commands are interpreted as instructions. When the Command Port is input to the computer, the Status Word provides eight bits of information on the status of the IC. The significance of the bits in the Mode Command, Instruction, and Status Word are given in Table 5.3. We shall examine how these command bytes are implemented in the experiment.

PROCEDURE

STEP 1 Mount the three ICs on the breadboard with the 8251 between the other two and the 74LS00 nearest the cable connectors. With the +5-V power rail disconnected, wire the circuit as shown in the schematic, Figure 5.16. Note that the serial output of the transmitter is tied to the serial input of the receiver. This connection makes the USART send to itself. Obviously, in a real situation, these pins would be connected to another USART.

STEP 2 The 555-type timers of the 556 IC have been wired as astable oscillators to generate the clock frequencies required by the 8251. The frequency connected to the CLK input at pin 20 of the 8251 should be greater than 10,000 Hz. The calculated value is 14,500; however, it was measured at 11,600 due to the large uncertainty of the capacitor value. The frequency wired to the TxC and RcC inputs should be around 300 Hz for a bit (Baud) rate of 300. The only requirement is that the CLK frequency be at least 30 times the TxC and RcC frequencies. If you have an oscilloscope available, you might check these frequencies before proceeding. Otherwise, if later you find the circuit does not work, you can use the gated counter circuit of Experiment 2.5 to check the frequencies.

STEP 3 The BASIC program consists of four parts corresponding to the four activities: initialize, check the condition of the USART, transmit a character, and receive a character. Each part of the BASIC program has a corresponding machine language routine that will be described in Step 4. The initialization routine consists of inputting the message to be transmitted and programming the USART at lines 20–70. Lines 80–150 form a FOR-NEXT loop, which

TABLE 5.3 COMMAND PORT BYTES

DATA BIT	OUT AFTER RESET MODE	OUT SUBSEQUENT INSTRUCTION	IN STATUS
D7	#Stop bits:	Synch. Hunt	Data Set Rdy
D6	11=2, 10=1 1/2	Internal Reset	Synch. Detect
	01=1, 00 NA.		
D5	Even Parity	Request to Send	Framing Error
D4	Select Parity	Error Reset	Overrun Error
D3	Word length:	Send BREAK char.	Parity Error
D2	11=8, 10=7	Enable Rc	Tx Empty
	01=6, 00=5.		
D1	Bit Rate:	Ready Data Term.	Rc Ready
D0	11=/64, 10=/16	Enable Tx	Tx Ready
	01=/1, 00=Sync.		

successively transmits each character in the message. The check routine at lines 90 and 100 and the subroutines at lines 240–340 and 210–230 reads the Status byte and decodes it to determine the state of the Error flags and the Ready flags. None of the error flags inhibit the USART. If an error is detected, a number is printed to indicate which error flags were set. The error flags are checked and reset if necessary after each character in the message is received. The ready flags are also decoded and used at lines 110 and 120 to decide whether to receive or transmit a character. If the transmitter is ready (H=1) and the receiver is not ready (G=0) then the next character is transmitted. If neither is ready, the program loops back and reads the flags again. It will stay in this loop until the Tx or Rc flag is set. If both are set, the program will not branch but proceed to input the received character at line 130. When all characters have been transmitted and received the program will accept another message.

Load the following BASIC program.

BASIC PROGRAM

```
 10 REM 123456789 123456789 1234567890                (for B&W models)
 10 CLEAR 32129                                       (for Color models)
 20 LET L = USR 16514                                 (for B&W models)
 20 LET L = USR 32130                                 (for Color models)
 30 PAUSE 10
 40 LET L = USR 16519                                 (for B&W models)
 40 LET L = USR 32135                                 (for Color models)
 50 CLS
 60 PRINT " ENTER MESSAGE"
 70 INPUT C$
 80 FOR K = 1 TO LEN C$
 90 LET L = USR 16524                                 (for B&W models)
 90 LET L = USR 32140                                 (for Color models)
100 GOSUB 240
110 IF F = 1 AND H = 1 AND G = 0 THEN GOSUB 210
120 IF G = 0 THEN GOTO 90
130 LET L = USR 16535
130 LET L = USR 32151
140 PRINT AT 1,K; CHR$ L
150 NEXT K
160 PRINT AT 21,0;  "MORE? (Y OR N)"
170 INPUT A $
180 IF A$=" Y"  THEN GOTO 50
190 LET L = USR 16539                                 (for B&W models)
190 LET L = USR 32155                                 (for Color models)
200 STOP
      * * * *
210 POKE 16531, CODE C$(K)                            (for B&W models)
220 LET L = USR 16530
      * * * *
210 POKE 32147, CODE C$(K)                            (for Color models)
```

```
220 LET L = USR 32146
* * * * * * * * * *
230 RETURN
240 LET L = L - 64*INT (L/64)
250 LET E = INT (L/8)
260 PRINT AT K+1,0; "ERROR=" ;E
270 IF E>0 THEN LET P = USR 16519          (for B&W models)
270 IF E>0 THEN LET P = USR 32135          (for Color models)
280 LET F = L - 8*E
290 LET G = F - 4*INT (F/4)
300 LET H = G - 2*INT (G/2)
310 LET G = (G - H)/2
320 LET F = (F - (2*G + H))/4
330 PRINT AT K+1,10; "TE=" ;F; " RR=" ;G; " TR=" ;H
340 RETURN
400 FOR M = 16514 TO 16543
400 FOR M = 32130 TO 32159
410 INPUT N
420 POKE M,N
430 PRINT M;" =" ;PEEK M,;
440 NEXT M
```

STEP 4 The machine code listed below should now be entered using RUN 400. It consists of six separate subroutines beginning at addresses 16514/32130, 16519/32135, 16524/32140, 16530/32146, 16535/32151, and 16539/32155. These subroutines perform the following respective tasks: initial Mode command, clear Error flags and enable receiver and transmitter instruction, read the status word flags, load a character into the transmitter, read a character from the receiver, and reset the USART. Study the Comments in the listing to understand how each subroutine performs.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	62	LD A,N	:Mode Command Byte: 01 00 11 01
16515 / 32131	77	N	:1 Stop-No Par-8 bits-/1
16516 / 32132	211	OUT (N),A	:Load in Command Port
16517 / 32133	11	N	:at Port 11.
16518 / 32134	201	RET	:Return to BASIC for PAUSE
16519 / 32135	62	LD A,N	:Instruction Byte: 00010101
16520 / 32136	21	N	:Error reset-Tx & Rc enable
16521 / 32137	211	OUT (N),A	:Load in Command Port
16522 / 32138	11	N	:at Port 11.
16523 / 32139	11	RET	:Done with Initialization.
16524 / 32140	219	IN A,(N)	:Read the Status Word
16525 / 32141	11	N	:from Command Port 11.
16526 / 32142	79	LD C,A	:Put it in register C
16527 / 32143	6	LD B,N	:and zero register B.
16528 / 32144	0	N	
16529 / 32145	201	RET	:Return BC value to BASIC.
16530 / 32146	62	LD A,N	:Transmit Subroutine:
16531 / 32147	0	N	:character to transmit
16532 / 32148	211	OUT (N),A	:loaded into Data Port
16533 / 32149	3	N	:at Port 3.
16534 / 32150	201	RET	:Done.
16535 / 32151	219	IN A,(N)	:Receive Subroutine:
16536 / 32152	3	N	:get character from Data Port
16537 / 32153	24	JR d	:Load it into BC by
16538 / 32154	243	d	:jumping to 16525/32141.
16539 / 32155	62	LD A,N	:Reset USART:
16540 / 32156	85	N	:Instruction byte 01010101
16541 / 32157	211	OUT (N),A	:loaded into Command Port
16542 / 32158	11	N	:at Port 11.
16543 / 32159	201	RET	:Back to BASIC.

STEP 5 Read through the BASIC listing and observe how the machine routines are related. The following REMark statements may help you interpret the action at the indicated line numbers. Do *not* load these into the computer.

25 REM Have to give USART enough time to implement Mode command before outputting Instruction command.
75 REM K is the position of the current character in the message.
115 REM F = 1 means that the transmitter is empty; H = 1 means the transmitter will accept a character; G = 0 means the receiver has not received a character.
185 REM This subroutine will internally reset the USART so that when the BASIC program is run again, the Mode Command will be accepted without having to reset with pin 21.
215 REM The operand of the LD A,N machine instruction at 16531/26732 is the value of the character at position K in the message.
245 REM L is initially the value of the Status Word. It is converted to be the value of bits D5–D0.
255 REM E is the value of the three error flags at bits D5, D4, and D3. It can range from 0 to 7 depending on which flags are set to logic 1.
285 REM Here F is the value of bits D2, D1, and D0.
295 REM Here G is the value of bits D1 and D0.
305 REM Here H is the value of bit D0, the Tx Ready flag.
315 REM Now G becomes the value of bit D1, the Rc Ready flag.
325 REM Now F becomes the value of bit D2, the Tx Empty flag.

STEP 6 There are three error flags on bits D5, D4, and D3 as shown in the third column of Table 5.3. The framing error flag is set (to a logic 1) if the receiver does not detect a stop bit. The overrun error flag is set when the computer does not read a received character before the next one is received. The parity error flag is set if the parity bit received plus the number of received bits in the logic 1 state is 1 (0) when even (odd) parity is selected.

STEP 7 Apply power to the circuit. Now perform an external reset of the IC by removing the jumper wire at pin 21 from the 0-V rail, momentarily connecting it to the +5-V rail, and then reconnecting it to the 0-V rail.

STEP 8 Enter RUN. Then input (between the quotes which appear on the bottom line of the screen) a message of fewer than 20 characters. As each character is transmitted, it will appear on the second line of the screen. Before it is transmitted the ERROR flag value will be printed on the next line along with the three binary values of the TE (transmitter empty), RR (receiver ready), and TR (transmitter ready) flags. The value of the ERROR flag should be 0. The RR flag will first be 0, then will change to 1 as the character is displayed.

STEP 9 If you do not successfully send and receive the message or if the flags do not all become 1s, press the BREAK key, power down the circuit, and recheck your hardware and software.

STEP 10 If you successfully sent and received, answer "Y" to send a second message. This time change to graphics mode (press Shift 9) and type in your message. If you are using a B&W model, the message should appear with inversed (white on black) letters. You should receive it with inversed letters. If you are using a Color model, the only characters you can send in graphics mode are R, S, T, and U. You should also receive what you send: Answer "N" after it has been received.

STEP 11 The Mode Command can now be changed to transmit seven-bit characters (D6–D0) instead of eight bits as originally programmed by ENTERing the direct command

```
POKE 16515, 73      (B&W)
POKE 32131, 73      (Color)
```

and then ENTERing RUN. Verify the value 73 from the first column of Table 5.3.

STEP 12 Repeat Steps 9 and 10. With a B&W model, you should observe that white on black characters are received as black on white. This is because the code value for the inversed video is greater than the normal character by 128, that is bit D7 is a logic 1 for inversed video. With a Color model, the graphics characters R, S, T, and U are the only four that are printable characters when 128 is subtracted from their codes; you should receive the characters !, ", #, and $, respectively.

STEP 13 Repeat Steps 11 and 12 by programming the Mode Command word for six and five bits per character. With a B&W model, verify that when the character FAST (Shift F) (=11100101) is loaded into the transmitter that the characters received and displayed are 9 (=00100101) and the graphics character on the five key (=00000101), respectively. If you are using a Color model, verify that when the character RESTORE (Extended S) (=11100101) is transmitted that the characters received are e (=01100101), % (=00100101), and ? (=00000101). (The question mark is printed in default because the codes between 0 and 31 are nonprinting in this model.)

STEP 14 POKE 16515/32131 with the Command Mode byte to give two stop bits, no parity, eight-bit character length, and a divide-by-1 Baud rate: 11001101 = 205. RUN and enter any message. Observe the LED on the serial line. You should see it dim each time a character is transmitted. Now POKE 16515/26716 with the Command Mode byte to send/receive at 1/16 of the Tx/Rc clock. The byte is 11001110 = 206. The rate will now be about 18 bits per second or slightly less than two characters per second. When you transmit, the LED will clearly show the logic levels change as the bits are transmitted. Observe the flag states displayed on the screen as the character flashes the LED.

STEP 15 Finally, POKE 16515/32131 to change the Baud rate to 1/64: 11001111 = 207. Change line 110 in the BASIC program to:

```
110 IF H = 1 AND G = 0 THEN GOSUB 210
```

Now send a message. Observe the flag states displayed as the LED indicates the character bits being sent. You will likely observe that by ignoring the transmitter empty flag, TE (variable F), the program runs fast enough at this Baud rate to read one character twice. Restore line 110 and repeat.

SUMMARY Because of its programmability, the USART is a very versatile and convenient IC to use with a computer for serial communication. Where two computers are available, a very good project is to develop the software to have them communicate serially; at the relatively low Baud rates used in this experiment, a distance of 10 feet between computers should be feasible. Note, however, that although the character code used by the Timex/Sinclair 2000 model is ASCII, the character code of the other models is not. Also, the same clock should drive both units, and the Ground rails of the two terminals must be connected.

□ 6 □

analog conversions

ANALOG-TO-DIGITAL CONVERTERS

One of the more useful tasks of microcomputers is to acquire data. In most cases, data occurs as an analog signal or voltage (that is, a signal or voltage that varies in a continuous fashion such as the mercury in a glass thermometer rises continuously as the temperature increases). Analog data can vary relatively slowly such as that produced by a temperature sensor and gathered over long periods of time, or they can vary relatively fast, like that produced by current flowing in an electrical circuit at the instant of switch on. Digital microcomputers can only accept data in digital form so we must transform the continuously varying analog data into the discrete steps of digital data. We have already seen how the reverse process is accomplished using digital-to-analog converter integrated circuits.

There are several techniques used to convert an analog signal to an equivalent digital signal having a binary value proportional to the analog voltage. We shall consider the technique known as the *successive approximation method.* In this technique, the analog voltage signal is compared in a series of steps to digital voltages having values which are exactly one-half of the previous step. For an eight-bit converter there are eight steps. Figure 6.1 illustrates the first four steps in a successive approximation conversion. It starts with the most significant bit in the first step and sets the digital voltage to one-half of the voltage range of the converter. If the analog voltage is greater than the digital voltage, then the most significant bit is set to a logic 1; if it is less, then the bit is set to a logic 0. The digital voltage is stored if the bit is a logic 1. For the second and subsequent steps, one-half of the digital voltage of the previous bit is added to the stored value of the digital voltage and compared to the analog voltage. For each step that the digital voltage is greater than or equal to the analog voltage, the bit value is set to a logic 1 and the voltage weight is added to the stored digital voltage. One distinctive advantage of the successive approximation method is that the time required for all conversions is the same and does not depend on the magnitude of the analog signal. The binary weight of the voltage of each bit per volt of the full scale of the converter's range is:

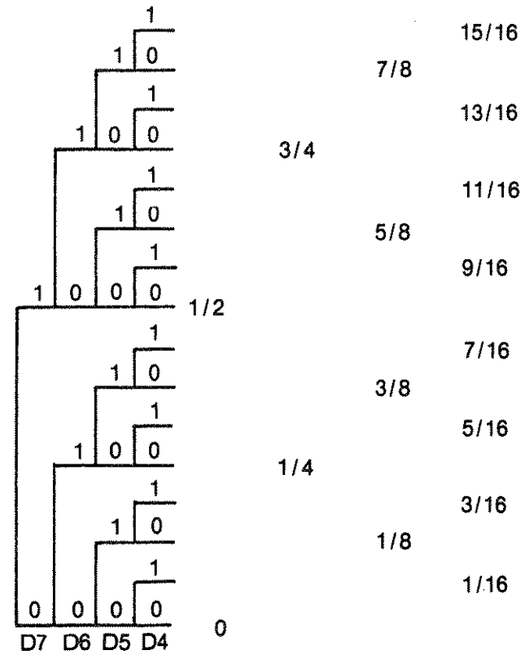


Figure 6.1 Successive Approximation Diagram.

	MBS							LSB
Binary Digits:	D7	D6	D5	D4	D3	D2	D1	D0
Voltage Weight:	.5000	.2500	.1250	.0625	.0313	.0156	.0078	.0039

What if you wish to convert an analog voltage of 7.30 V into eight-bit digital form? You could alter the LSB weighting to 0.039 V so that the total range now becomes 10.00 V, but in so doing the resolution of the converter is reduced. What is meant by the term *resolution*? It means just how closely the Analog-to-Digital Converter (ADC) can represent an analog voltage in digital form. For example, the 5-V full scale converter could not distinguish clearly between two analog voltages, say equal to 1.668 and 1.675 V, because its limit of resolution is 0.02 V and these values differ by less (0.007 V) than the resolution. If it is important in your measurement situation to achieve higher resolution and an extended voltage range also, then you must use ADCs that provide more bits. For example, a 5-V, 12-bit ADC will resolve down to a sixteenth of 0.02 V equal to 0.00125 V or 1.25 mV. The LSB voltage weighting is now equivalent to 1.25

mV with the MSB voltage weighting still being equivalent to 2.56 V. When it is realized that an eight-bit converter can achieve a 1 in 256 resolution or an accuracy of 0.4% then use of such simple integrated circuits in science laboratory experiments is justified—where often other errors completely swamp the 0.4% resolution of the converter.

The converter chip which will be used in the experiments is the ADC0804 which requires a 5-V supply. It uses an internal reference voltage derived from the 5-V power supply and provides a resolution of 0.02 V. The schematic block diagram and pin assignment is shown in Figure 6.2. It is easily interfaced to a microcomputer requiring, besides the power and data bus connections, a clock derived from an external RC timing circuit, and two Device Select Pulses: one to start the conversion and the second to read the data by enabling the three-state data outputs once the conversion is completed. A third output control pin is available on the chip, referred to as the INTR*, which is the End Of Conversion (EOC) flag. When finished with a conversion of the analog voltage appearing at the input, the ADC signals that it has finished the conversion by taking the EOC pin low.

When interfacing the converter to a microcomputer there are two approaches which can be made for inputting data to the microprocessor. One approach is to poll the EOC flag and when it goes low, activate the three-state outputs of the converter to transfer the data to the microcomputer data bus. This approach, however, requires further hardware (a one-bit input port) to sense the EOC flag. A second approach is to allow the microcomputer itself to wait a period slightly longer than the conversion time of the converter (the ADC0804 is rated at 100 microseconds) and then input data which would then be valid at the time. The second approach will be adopted here as it is an example of the substitution of software for hardware.

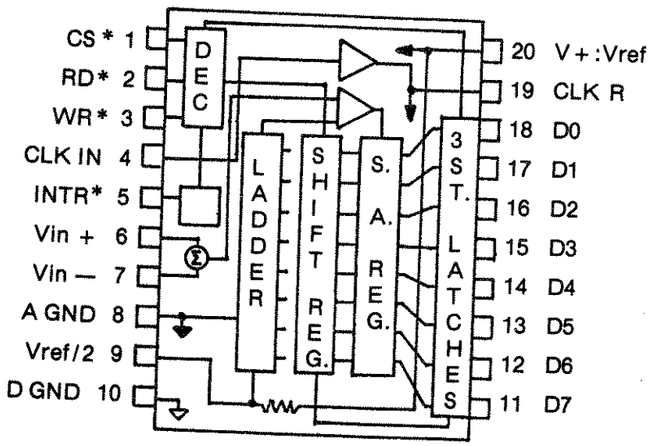


Figure 6.2 Schematic of the ADC0804.

TABLE 6.1 MACHINE LANGUAGE ROUTINE FOR ADC

```
START,  LD C,DELAY      :A timing delay byte is
        N               :loaded into register C.
        LD HL,STORE     :HL holds pointer address
        Lo N            :where converted data will
        Hi N            :be stored.
        OUT A,(N)       :Start ADC0804 conversion.
        N               :Device code for ADC0804
WAIT,   DEC C           :By decrementing C and
        JR NZ,WAIT      :jumping around loop until
        d               :C is zero we wait.
        IN (A),N        :Input data from ADC0804 latches
        N               :Device code for ADC0804.
        LD (HL),A       :Load memory with data.
        RET             :Return to BASIC.
STORE,  NOP             :Data storage location
```

The ADC0804 has been specifically designed to interface readily with microcomputer systems. In the case of the Timex/Sinclair which uses a Z80 microprocessor, the necessary control signals are simplified by connecting IN* to the IC's RD* pin, OUT* to the WR* pin, and a Device Code connected to the CS* (Chip Select) pin. The chip internally ORs the Device Code with the appropriate control pulse (OUT* or IN*) to start the conversion or enable the three-state data buffers, respectively. Which way round would you place the control signals to start the conversion and then read in the latch contents to the microcomputer using Device Code 3? You should use (OUT 3)* in your program to start the conversion and (IN 3)* to enable the three-state buffers and read in the contents of the ADC data latches. A simple machine language program is listed in Table 6.1, which could be used with the ADC0804 to input a single data value.

This program would have to be combined with a BASIC program as in previous cases with the converted value of the analog input voltage being stored at the memory location labeled STORE. Do not forget to allow an extra character in your REM statement to leave this memory location free for data. By using PRINT PEEK in your BASIC program, the contents of the memory location can be displayed on your TV screen in decimal form. We shall leave further description of the A-to-D converter software and hardware to the experiments.

SIGNAL CONDITIONING

When we described the conversion of an analog signal to a digital value we assumed the ADC could read the signal from our measuring device. That is, we figured that the analog voltage would be between 0 and +5 V and that a resolution of 0.02 V was sufficient. We have seen that the way to improve the resolution is to use an ADC that converts to more than eight bits. But what about an analog signal that ranges between 0 and only +1 V? Besides, where do we obtain the analog signals in the first place?

These two questions are closely related and we shall spend the rest of this chapter trying to answer them.

Transducers It is obvious that we need an electrical signal in order to be able to convert that (analog) signal to a digital value. Many of the interesting measurements are not electrical. For example, sound, temperature, pressure, length, light, force, motion, magnetic field, and strain, to name a few, are definitely not electrical. All of these quantities are continuously variable and are therefore what we have been calling analog signals. What we need are devices that can translate these signals into electrical signals which vary in an exactly analogous manner to the quantity of interest: in other words, the electrical analog of the signal. This is where the term *analog electronics* originated. The devices we are looking for are called *electrical transducers*.

There are two types of electrical transducers. Those that convert some physical property into an analog electrical signal are known as *sensors*, and those that convert in the opposite direction, that is an analog electrical signal into some other physical phenomenon, are called *actuators*. Can you think of the electrical transducers for sound? A microphone is a sensor and a loudspeaker is an actuator. This raises another point about transducers that has to be considered. Some transducers can only detect changes in a physical property while others can detect unchanging or constant values of the physical property. The former are *dynamic transducers* and the latter are *static transducers*. For instance, a microphone really measures the changing pressure of the air created by sound. It is a dynamic pressure transducer; you cannot use it to measure a static pressure such as atmospheric pressure. To measure a static pressure we need a transducer that can measure the force acting on a specified area.

Up to this point we have said that electrical transducers convert a nonelectrical physical property into an electrical signal. We have not specified what kind of electrical analog signal. There are several that are possible. An analog voltage is perhaps the most obvious electrical signal, but transducers are made that also convert to analog resistance, capacitance, inductance, current, and a few other electrical properties. By means of various circuits, these properties must first be converted to voltages in order to use an analog-to-digital converter. The more common transducers are listed in Tables 6.2 and 6.3 which give examples of sensors and actuators in terms of the physical property and the corresponding analog electrical property.

TABLE 6.2 COMMON ELECTRICAL SENSORS

PROPERTY	*VOLTAGE*	*CURRENT*	*RESISTIVE*	*CAPACITIVE*	*INDUCTIVE*
Mechanical	Piezoelectric	Switch	Strain gage	Pressure head	Generator Variable transformer
Thermal	Thermocouple, Bolometer	Diode Transistor	RTD Thermistor	?	?
Light	Photocell	Photomultiplier Photodiode Phototransistor	Photoresistor	?	?
Magnetic	Magnetorestrictor	Hall effect	?	?	Search coil
Chemical	Electrode	?	Conductivity	Dipolmeter	Susceptibility

TABLE 6.3 COMMON ELECTRICAL ACTUATORS

PROPERTY	*VOLTAGE*	*CURRENT*	*RESISTIVE*	*CAPACITIVE*	*INDUCTIVE*
Mechanical	Piezoelectric	Switch	?	?	Motor Solenoid
Thermal	Thermopile	?	Heater	?	?
Light	?	Lamp LED	?	?	?
Magnetic	?	Coil	?	?	?
Chemical	Electroplate	?	?	?	?

Now that we have seen how it is possible to convert a physical property measurement to an analog voltage, we are faced with the fact that most transducers will not produce large enough voltages to span the range of the ADC. It would be of little value to have a 0 to 5 V range with a resolution of 0.02 V and then find the transducer voltage to provide a maximum signal level of 0.10 V. If that were the case, the only digital values we could obtain would be the numbers from 0 to 5. In addition to this problem, we also have to consider that the transducer itself and the wires carrying the signal from the transducer are subject to noise pickup (spurious electrical voltages) from the surroundings, like static pops and hum on a radio. If our signal is small then the noise pickup can represent a sizable signal itself impressed on the signal.

To solve these remaining problems, we shall describe some of the practical means that can be used to condition a signal and make it possible to achieve good analog-to-digital conversions.

Transistors Most of this signal conditioning involves amplifying small voltages until they are large enough to be handled by analog-to-digital converters. Other forms of signal conditioning involve current amplification and noise reduction. The mention of the word amplification means that amplifiers are involved in this process of signal conditioning and the first type of amplifier we will consider is the transistor amplifier. We will confine our description of transistors to one type only, the NPN common emitter, and confine the frequency range of operation of our circuits to below about 500 KHz. In this way we can make a number of simplifying assumptions about the operation of the transistor amplifying circuits.

The NPN transistor is a three terminal device. Its terminals are called the *emitter*, the *base*, and the *collector*. To get some idea how a transistor operates, consider the circuit shown in Figure 6.3. As with the integrated circuits that we have discussed up to this point electronic circuits always need to be turned on. Transistors are no exception to this rule (because they tend to be the major element inside digital circuits). The transistor is switched on by applying a positive voltage $+V_{CC}$ to the collector terminal of the transistor. So far so good because this is just what we have been doing with our integrated circuit chips. But that's not the whole story because the digital integrated circuit chips which contain transistor amplifying circuits have had their circuits designed by a team of electrical engineers. Now what we have to show you is not only how to switch the transistor on but also how to design it into an electronic circuit to produce an amplified signal. This is not such an easy task as might first appear. The

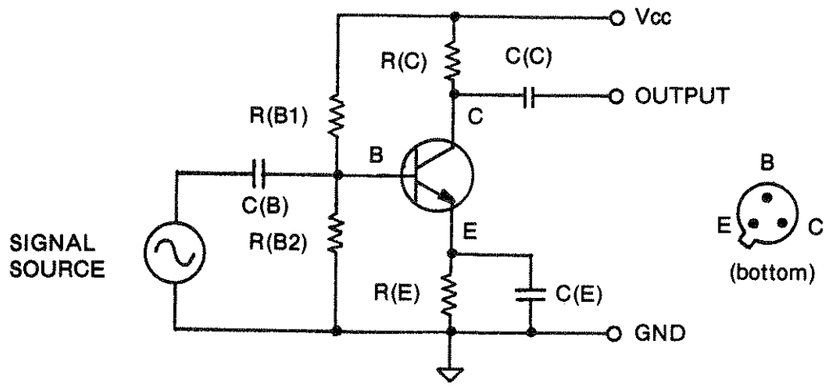


Figure 6.3 NPN Transistor Circuit.

transistor you buy from your local store differs, sometimes significantly, from a transistor of the same type which your friend bought from another store. So how can a circuit be designed to take into account such large variations? Well we can try by adopting a number of empirical rules which work quite well for most silicon NPN transistors which you wish to use as amplifying devices.

Just before we proceed, are you quite sure what an amplifier is? Probably, even if you are young or older, because everyone has heard the amplified sound from someone's hi-fi system at some time or another (any advances on 2 A.M. in the morning?). The electronic amplifier (a number of transistors) enlarges the small voltages produced by the tone arm on a record player, or magnetic pickup on a cassette recorder, to much larger voltages which can drive the magnet coils inside those very large loudspeaker boxes.

So back to the transistor amplifier; the empirical rules which can be used with some success are as follows:

1 Choose a transistor based on the amount of current required by the device connected to the output. For example, an analog-to-digital converter would only require a low power transistor, say less than 50 mA, because the converters usually have a relatively high input impedance. Should you have a digital-to-analog converter which has to drive the coils of an electrical motor, you may have to choose a transistor which passes a large current, say 5 A, to use in your amplifying circuit.
2 Next choose a collector resistance so that the voltage drop across the resistance will be about one-half the supply voltage at the expected current drain of your amplifier. Again, for example, assume that you wish to operate the transistor

at a current level of 5 mA, from a battery supply (V_{CC}) of 12 V. Then by using Ohm's Law, the value of resistance R(C) can be calculated as follows:

$$R(C) = V_{CC}/21 = 12E3/10 = 1.2 \text{ Kohm}$$

3 Calculate a value for R(E) that will allow for about a 1 V potential drop from the emitter E to the ground line. A preferred value of 220 ohms should be reasonable. Choose C(E) to be from 10 μF up to 47 μF if you can afford it. C(E) and R(E) are used to stabilize the circuit operation against thermal runaway.
4 The values of R(B1) and R(B2) can now be chosen. The resistors themselves should be of the order of ten times the size of R(C) and then must be chosen in a ratio that will put the base potential about 0.6 V above the emitter potential. So if the emitter potential was about 1 V, the base potential needs to be about 1.6 V. All we need to do is select R(B1) + R(B2) to be about 15 Kohm into a ratio to drop the 12-V supply line down to 1.6 V at the base. This turns out to be a ratio of

$$10.4/1.6 \text{ or } 6.5{:}1 \ (10.4 + 1.6 = 12)$$

R(B1) could then be chosen close to 13 Kohm and R(B2) chosen close to 2 Kohm. Preferred values of resistors would probably be 12 Kohm and 1.8 Kohm.
5 Switch on and check to see that the voltage at the collector C is about one-half the supply voltage, for instance 4.5 V to 6.5 V might be a suitable range. Measure the voltage between the base and emitter which should be close to 0.6 V. There is a direct relationship between the voltage drop from the base to the emitter and the voltage appearing at the collector. Should your transistor be so far from the normal that your amplifier cannot approach any of our empirical rules then you need to fall back on the last resort! Yes, you guessed it, replace R(B2) with a variable resistor and adjust it until the voltages agree with our rules! You can always remove it from your circuit later on and measure it with an ohmmeter and then replace it in the circuit with a fixed resistor closest to its value.

So now you have a transistor amplifier turned on and adjusted to provide amplification. The amount of amplification is governed to a certain extent by the value of R(C); by making R(C) larger you increase the gain (amplifying factor or amplification) of the transistor amplifier. But there is a limit to the amount of the increase, which is dependent on the size of the signal you wish to amplify.

The amplification occurs on a small voltage input signal applied to the base which is output as a much larger voltage signal at the collector. The ratio of the two signals is referred to as the *gain* of the amplifier, where the voltage gain

$$A(V) = \text{Output Voltage}/\text{Input Voltage}$$

Let us assume that the input signal is varying with time, say a sinusoidal waveform. In sound such a waveform can be produced by whistling or singing a pure note. Many electronic signals are sinusoidal in nature. The sinusoidal amplitude is often defined by measuring the peak to peak (P to P) amplitude as shown in Figure 6.4. An incoming signal of amplitude 20 mV P to P to an amplifier of gain 100 times will produce an output signal of 2 V P to P. This is what we term *voltage gain*.

Transistors are often described by their current gain characteristic as well, which we can define here as

A(I) = Output Current/Input Current

The output current would be, in the case of the sinusoidal input current, the varying current component in the collector resistor, R(C). This varying current can be represented by the term *delta* I(C). The varying current signal producing this change appears in the base circuit of the transistor amplifier and so can be designated delta I(B), the change in the base current. The current gain, A(I), can now be written as

A(I) = Delta I(C)/Delta I(B) = Beta

Beta is called specifically the common emitter current gain of the transistor and also is equivalent to another term often used by manufacturers and engineers called h(FE). Typical values for beta range from 50 to 300.

We haven't spent any time talking about the different configurations of transistor amplifiers, but there are two other amplifying circuits different from the one we have described called *common base* and *common collector* (or emitter follower) transistor

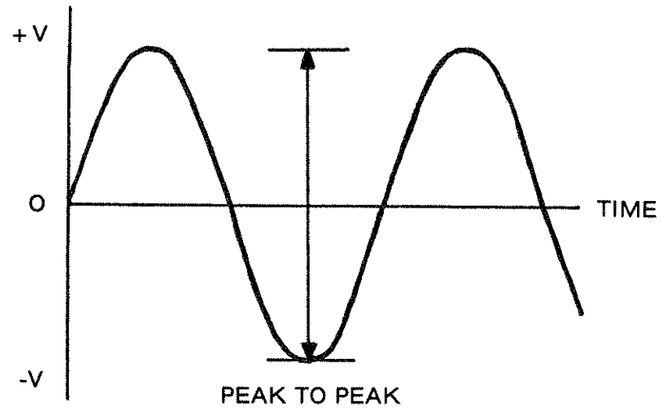


Figure 6.4 Sine Wave.

amplifiers. (It is not intended that this be an exhaustive text on transistor electronics, so we suggest that you look up additional texts to follow this matter in greater detail.)

The common base transistor uses the emitter lead as the input terminal and retains the collector lead as the output terminal. The current gain of this amplifier configuration is notable, namely

A(I) = Delta I(C)/Delta I(E) = Alpha

Alpha is the current gain of the common base transistor amplifier. Typical values for alpha range from 0.9 to 0.99.

The common emitter amplifier circuit has another interesting feature called *phase inversion*, which is of use in designing digital electronic circuits. This occurs when a rising voltage on the base causes a falling voltage on the collector of the transistor, and vice versa. The collector is in opposition to the base. Two waveforms are said to be in opposition when they are 180 degrees out of phase. The transistor amplifier is called "common emitter" because the emitter lead E is common to the circuit containing the input signal and to that containing the amplified output signal (see Figure 6.3).

Another aspect of signal transfer often overlooked is the fact that there are two cables or wires connecting the signal source to the amplifier and two wires connecting the amplifier to the output device. One lead is often referred to as the live lead because its potential fluctuates with respect to the second lead which is usually referenced to ground potential. In many instances the second lead is the earth shield around a central wire conductor as in coaxial cables which are much preferred for transporting electrical signals over long distances. By earthing or grounding the external braiding you minimize electrical interference with the inner conductor. Nevertheless, you always need the two leads even though one of the leads (the ground) is common to both the input and output signal wires.

The only components in Figure 6.3 that we haven't yet explained are the coupling capacitors C(C). These capacitors are used when the signal is varying with time to couple the signal into (and out of) the amplifier without the dc (direct current) voltages being able to leak through. Remember, capacitors do not allow dc to pass through but they do allow ac to pass. In the case of dc amplifiers the coupling capacitors have to be dispensed with and the transistor circuits modified to take account of temperature drift of the transistor characteristics. In our approach, amplification of dc signals can be handled by integrated circuit amplifiers, op amps, because of their very small drift characteristics. This will be treated in the next section.

In signal conditioning you will tend to find transistors being used when output signal currents need to be amplified significantly (of the order of amps) to drive heavy electrical machines, motors, robot arms, etc. We have already seen one such example used for the stepper motor interface discussed in Chapter 5.

Transistors in Digital Circuits We have already discussed using the transistor as an amplifier (That is, one where a varying input signal is made very much larger as a varying output signal). The transistor, however, can be operated in another mode apart from that of amplification, that is, it can act as a switch.

If you think back to the rules we laid down for the transistor as an amplifier, one of them was that there had to be 0.6 V potential between the base and emitter for the transistor to operate as an amplifier. You can think of this base emitter potential as a controlling potential, because if the potential difference is reduced below about 0.4 V you will find virtually no current flowing through the collector resistor R(C) (refer to Figure 6.3). Using Ohm's Law on resistor R(C), if there is no current flowing through the resistor there can be no potential drop across it. If there is no potential drop across it then the collector end of the resistor must be at the same potential as the power supply end. In other words the collector of the transistor is at the high potential of the power supply V_{CC} and the transistor is in fact turned off.

If we now increase the potential between the base and emitter to about 0.8 V, we find that the maximum current allowed by the circuit components is flowing through R(C) and of course the transistor. Under these conditions the voltage drop across R(C) is a maximum or, in other words, the collector is at the minimum potential allowed by the circuit; that is, it is at a low potential and the transistor, we say, is full on. The potential is usually very close to 0 V. In effect then we can make the transistor look like a switch by applying voltages less than 0.4 V to the base to turn it off and greater than 0.8 V to turn it on. You could use a square wave input signal to the base, through a 1-Kohm resistor, of amplitude 5 V to switch the transistor on and off continuously.

Transistors can be combined to produce digital gates such as the NOR gate shown in Figure 6.5. If $V_{CC} = +5$ V = logic 1 state and GND = 0 V = logic 0 state, then the truth table for this circuit is:

INPUTS		OUTPUT
A	B	Q
0	0	1
1	0	0
0	1	0
1	1	0

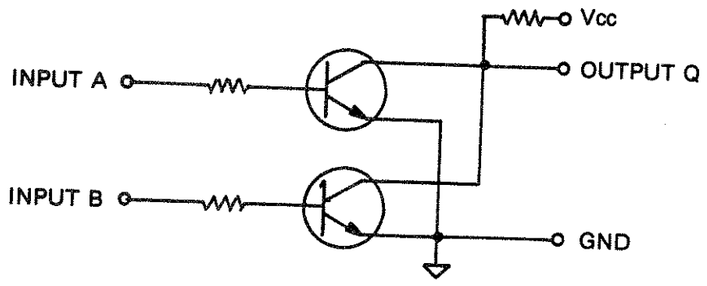


Figure 6.5 Transistor NOR Gate.

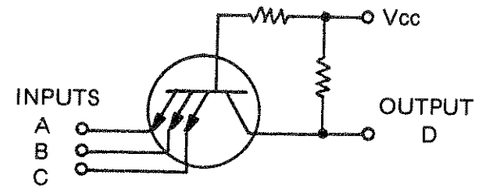


Figure 6.6 Three-input AND Gate.

This is identical to the truth table for a NOR gate. The natural characteristic of the common emitter amplifier of inversion and amplification enables them to be used as NAND and NOR gates as well as AND and OR gates. The current amplification of the transistors allows the output Q to drive many more amplifiers, each being a part of another gating circuit. You may now begin to understand the reasons for some of the terms used in digital IC work.

Further development of transistors for use in digital gates led to the manufacture of multiple emitter type transistors such as the three-input AND gate circuit shown in Figure 6.6. The inputs A, B, and C are either at 0 V or V_{CC} volts. The output D will appear as in the following truth table:

INPUTS			OUTPUT
A	B	C	D
0	0	0	0
1	0	0	0
0	1	0	0
:	:	:	:
1	1	0	0
1	1	1	1

Whenever any of A, B, or C inputs are low, the base is at a much higher potential than the emitter so the transistor conducts heavily (switched on) and the collector voltage is low (D). Only when A, *and* B, *and* C are at +5 V will the transistor turn off and the potential at the collector rise to +5 V (high). This again indicates how multiple input gates can be constructed.

Operational Amplifiers This term has been used for many years to describe electronic amplifiers that have very high gain (say, in excess of 1000 times). In modern times transistor amplifiers have been fabricated into integrated circuits often referred to as *linear integrated circuits*. The symbol of such a circuit is given in Figure 6.7. The amplifier has two inputs, labeled − and +, referred to as the inverting input and noninverting input, respectively. The other single lead is the output lead. These integrated circuits are characterized by very high open loop gain, usually greater than 10,000 times, are manufactured in DIL (DIP) packages, and have very low drift and high input impedance (draw very little current from the input signal).

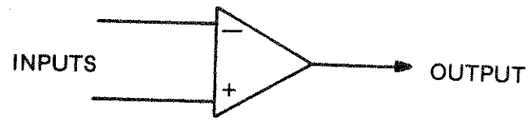


Figure 6.7 Linear Amplifier.

Discussion of how these circuits can be made into amplifiers can begin by considering the ideal operational amplifier (op amp). This ideal op amp has the following features: infinite open loop gain, infinite input impedance, and zero drift. To see how such an amplifier can produce a voltage gain consider the circuit shown in Figure 6.8 called an inverting amplifier.

Based on the ideal op amp characteristics, an input current I(I) will flow through R(I) then R(F) to the output, due to the fact that the input resistance is infinite. Also if we assume that the gain of the amplifier is infinite, then the value of V(I) must be 0, (otherwise there would be an infinite output voltage). The value of V(I) in practice approaches 0. The voltage gain of the closed loop is:

$$V(out)/V(in) = A(V) = -R(F)/R(I)$$

The negative sign indicates a phase inversion between input and output signals as was explained with the common emitter amplifier. The term *closed loop* refers to the use of the feedback resistor R(F) closing the loop from the output circuit to the input circuit. When precision resistors are used for R(F) and R(I) in an op amp whose gain is greater than 1000 times, the above relation can provide accurate predictions of the voltage gain of the amplifier—something that has been difficult to predict accurately with discrete transistor amplifiers.

When an amplifier is required with a high input impedance and no phase reversal between the amplified output signal and the input signal, then the noninverting amplifier should be used with the signal applied to the noninverting + input.

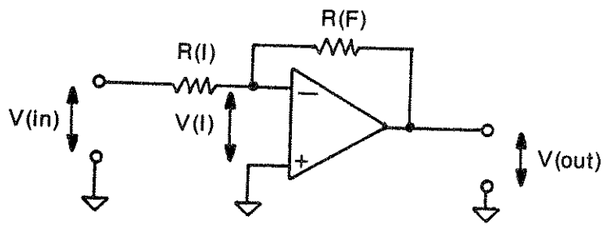


Figure 6.8 Inverting Op Amp.

A further op amp circuit which can be very useful in signal conditioning applications is the voltage follower illustrated in Figure 6.9. This amplifier is in fact a unity gain amplifier and requires no input or feedback resistors. The output waveform is an exact copy of the input waveform showing that there is no phase reversal between the input and output signals. The circuit characteristics exhibit high input impedance (draws negligible current from input signal) and very low output impedance (produces significant current in the amplified output signal). Such amplifiers are useful as current amplifiers or buffers.

In practice many op amp chips are operated from dual (bipolar) power supplies of around +12 V. Such chips would not be practical to use in our experiments with the Timex/Sinclair because of the need to provide additional power supplies. More recently, however, single voltage op amps have been produced which can work from +5 V only. We have obviously seized on this opportunity and carried out any signal conditioning needed in our experiments with single voltage op amps such as the LM358 which features high input impedance, very small drift, and high gain. They are used in the experiments involving strain gauges. They could be used in any experiments involving transducers which output only small dc voltages and very small currents, such as thermocouple thermometers.

We conclude this survey on signal conditioning in relation to op amps by describing their ability to reduce electronic noise present in input signals. Thermocouples are transducers which are sensitive to noise, and, by incorporating them in an op amp circuit, we can minimize the noise present by the inherent common mode rejection of the differential input op amp circuit shown in Figure 6.10. The thermocouple produces only a small potential per degree change in temperature of about 20 μV/Celsius degree. Using an amplifier of gain 1000 times this voltage can be amplified to a usable level. With such high gains, however, the noise is amplified the same amount as the signal. If the noise in the signal contains main frequency "hum," this alternating noise signal is present in both signal wires (the + and −) at the same time so that amplification of the noise takes place equally but with opposite phase. The signal on the other hand is producing a difference voltage between the inputs which is amplified with the − signal having been phase reversed. This adds to the + signal at the output. The Common Mode Rejection Ratio (CMRR) of a circuit is defined as:

CMRR = Differential Gain/Common Mode Gain

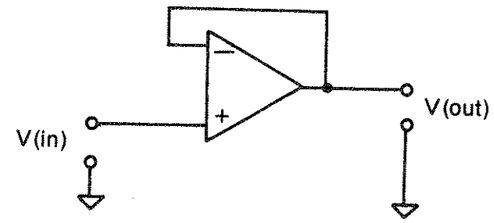


Figure 6.9 Voltage Follower Op Amp.

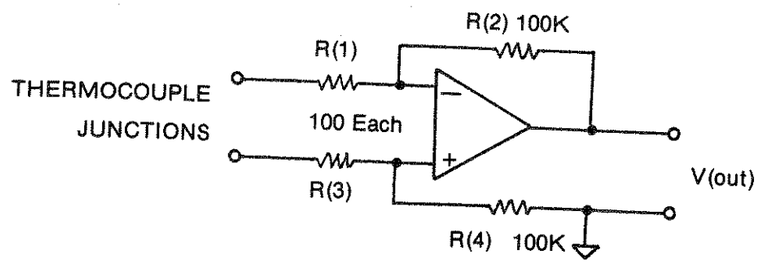


Figure 6.10 Differential Input Op Amp.

The differential gain can be calculated from the ratio of the two resistors, R2/R1. The common mode gain can be measured directly with a voltmeter. Common mode rejection ratios greater than 100 are acceptable.

EXPERIMENT 6.1
ANALOG-TO-DIGITAL DISPLAY OF RC CHARGING WAVEFORM

COMPONENTS
1 * 74LS32 Quad Two-Input OR Gate
1 * 74LS74 D latch IC
1 * ADC0804 Analog-to-Digital Converter
1 * 10-Kohm resistor
1 * 5.1-Kohm resistor
1 * 0.47-μF capacitor
1 * 150-pF capacitor

For optional second part of experiment:
1 * Oscilloscope
1 * AD558 Digital-to-Analog Converter

DISCUSSION When current flows from a constant voltage source through a resistor and into a capacitor, the electrical charge is stored in the capacitor. Consider the analogy to a water system supplying water at a certain pressure (voltage) through a pipe (resistor) connected to closed tank (capacitor). When a valve on the pipe is opened (a switch in the circuit is closed), water flows through the pipe at a certain rate, gallons per minute (current), determined by the size of the pipe (resistance). The initial pressure in the tank is 0 but starts to build up as the water (charge) is stored. As the pressure in the tank increases the rate of flow decreases until the pressure in the tank equals the pressure of the water supply and the water stops flowing. In the electrical circuit, the analogous situation corresponds to charging a capacitor. Because the rate of charging at a given moment is proportional to the amount of charge stored at that moment, then the increase (or growth) in voltage with time depends on the percentage of the total change. Such growth processes are exponential. In such processes, the time required to charge the capacitor up to the voltage of the supply obviously depends on the values of the resistance and the capacitor. A *time constant* defined as

TAU(seconds) = R(ohms) * C(farads)

is the time it takes for the voltage to reach 63% of its full value (actually 1 − 1/e where *e* is the base of natural logs; e = 2.71828).

In this experiment the Timex/Sinclair will be used to control the switching on of a circuit to charge a capacitor through a resistor. The computer will store converted values of the analog voltage developed across the capacitor as the charging process proceeds. The charging process should be exponential and a table of acquired values stored in memory can be displayed on the video screen. The circuit used to charge and discharge the capacitor is similar to the one used in Experiment 4.3 for turning an LED on and off and is repeated in the schematic, Figure 6.11.

PROCEDURE

STEP 1 Wire the circuit as shown in Figure 6.11 making sure that the power is disconnected between the Timex/Sinclair and the +5-V power rail. Resistor values are not critical, and values close to 4.7-Kohm or 5.6-Kohm resistors could be used in place of the 5.1-Kohm resistors. You will need space for three integrated circuit chips, so mount them as close to the cable socket as possible so that a fourth chip can be added later. Note that the differential input to the ADC0804 has been made single ended by grounding the negative input at pin 7.

STEP 2 Two device pulses are required to drive the 74LS74 D latch, and these are derived from the 74LS32 Quad OR Gate as shown in Figure 6.11.

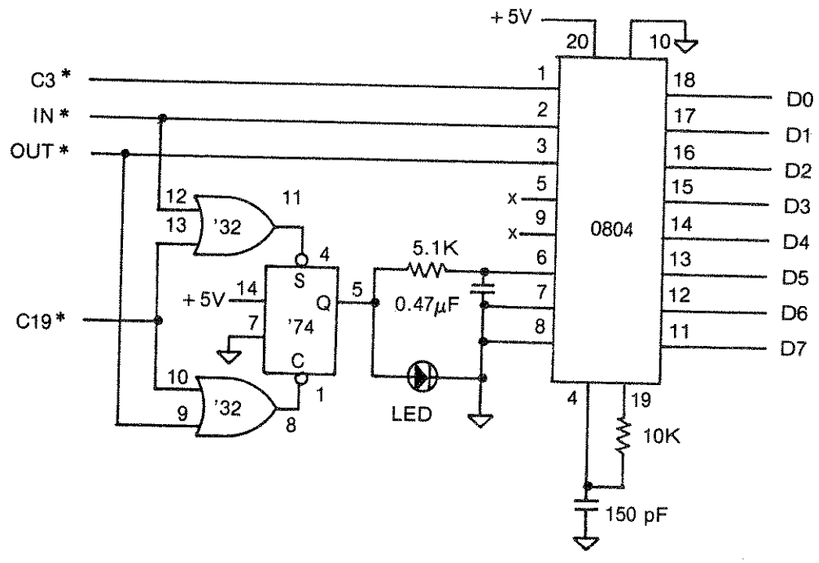


Figure 6.11 Experiment 6.1A Schematic.

STEP 3 Load the following BASIC program:

```
10 REM 123456789 123456789 123456789 123456789 123456789
   123456789 123456789 123456789 123456789 123456789
   123456789 123456789 1234567890                    (for B&W models)
10 CLEAR 32129                                       (for Color models)
20 FAST                                              (for B&W models only)
30 LET L = USR 16514                                 (for B&W models)
30 LET L = USR 32130                                 (for Color models)
40 CLS
50 FOR A = 16538 TO 16616 STEP 4                     (for B&W models)
50 FOR A = 32154 TO 32232 STEP 4                     (for Color models)
60 PRINT; TAB(8 - LEN "A" ); PEEK A; TAB(16-LEN "(A+1)" );
   PEEK (A+1); TAB (24 - LEN "(A+2)" ); PEEK (A+2);        TAB
   TAB (32 -LEN "(A+3)" ); PEEK (A+3)
70 NEXT A
90 PRINT  "ALL IN"
```

Note line number space left here.

```
150 STOP
160 FOR M = 16514 TO 16537                           (for B&W models)
160 FOR M = 32130 TO 32153                           (for Color models)
170 INPUT N
180 POKE M,N
190 PRINT M;  "=" ; PEEK M,;
200 NEXT M
```

If your Timex/Sinclair has 2K of W/R memory then the programs above will fit into the memory space. However, if you have only 1K of W/R memory then the few suggestions made in Chapter 2 should enable you to load the software without too much difficulty.

STEP 4 The following machine language routine is loaded in the usual manner by ENTERing RUN 160 and inputting the decimal codes for the routine.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16514 / 32130	1	LD BC,NN	
16515 / 32131	20	Lo N	:Register C holds timing byte
16516 / 32132	79	Hi N	:Register B holds count of events
16517 / 32133	33	LD HL,NN	:Starting address
16518 / 32134	154	Lo N	:of Data Table
16519 /	64	Hi N	:at 16538

/ 32135	125	Hi N	:or 32154.
16520 / 32136	219	IN (N),A	:Device code 19 sets Q high
16521 / 32137	19	N	:to charge capacitor.
16522 / 32138	211	OUT A,(N)	:Device select pulse to begin
16523 / 32139	3	N	:conversion of the ADC.
16524 / 32140	13	DEC C	:Delay loop: wait for
16525 / 32141	32	JR NZ,d	:conversion to complete.
16526 / 32142	253	d	:Loop to 16524/32140 20 times.
16527 / 32143	14	LD C,N	:Yes. Restore C count.
16528 / 32144	20	N	
16529 / 32145	219	IN (N),A	:Read ADC
16530 / 32146	3	N	:from Port 3.
16531 / 32147	119	LD (HL),A	:Store converted value in memory.
16532 / 32148	35	INC HL	:Point to next file location.
16533 / 32149	16	DJNZ d	:Is B=0?
16534 / 32150	243	d	:No. Jump to 16522/32138.
16535 / 32151	211	OUT (N),A	:Discharge capacitor for
16536 / 32152	19	N	:the next run
16537 / 32153	201	RET	:Return to BASIC
16538 / 32154		FILE START	:The first location in the :data file

STEP 5 Inspect the two programs above to see just how they interact. Line 30 calls the machine language program which starts at memory location 16514/32130. The machine language program initializes the BC register pair in one three-byte instruction with register C holding the timing byte for the time delay routine and register B holding the number of data values to be collected by the converter. Register B was used because of the Z80's special instruction DJNZ which automatically decrements register B and tests the zero flag. It uses a two's complement negative displacement to jump back to do another conversion and saves one byte of code, namely DEC B. Note that the code used in the above example is relocatable code—that is, the program could be run in any part of available W/R memory.

The program then initializes the HL register pair to mark the beginning of the area of W/R memory reserved for data and labeled "FILE START." Line 16520/32136 charges the capacitor using a device select pulse and then starts the converter allowing for a time delay of 20 * 14 clock cycles = 86.15 μsec before generating a further device select pulse at memory location 16529/32145 to input the converted data. The final device select pulse at memory location 16535/32151 discharges the capacitor ready for the next run.

STEP 6 Reconnect the power to the circuit and check that no integrated circuits are significantly overheated. Start by ENTERing RUN. After a short time four columns of acquired data will appear on the video screen. The time constant of the RC circuit can now be determined. To make an absolute determination, the time to make successive readings has to be calculated using the total number of clock cycles in the loop from 16522/32138 to 16534/32150 (dependent on the value put into register C) multiplied by the time for one clock cycle (0.3077 μsec). For example, look at your table of values and determine the maximum digital value to which the capacitor charged. Assume that the charging process was exponential so after one time constant the voltage should have increased to 0.63 of the final value. If your final value was 220 decimal, for example, then the value of voltage across the capacitor after a time interval equal to one time constant (TAU) would be 0.63 * 220 or about 139 decimal. Now find how many

samples were taken to obtain that value and multiply by the total time per sample.
If your count was 30 samples, then one time constant would be equal to

30 * (time for one sample, see Step 4) = 30 * 82.77 μsec = 2.48 msec

Because the time constant TAU = R*C, then we can determine C if R is known.
Suppose R = 5.1 Kohm, then

$$C = TAU/R = (2.48 \times 10-3)/(5.1 \times 103) = 0.48 \times 10-6 \text{ F}$$

or

$$C = 0.48\ \mu\text{F}.$$

STEP 7 You can POKE a different timing byte into register C at memory location 16528/32144 and then repeat the experiment to observe the effects. You can also try different values for R and C.

STEP 8 It would be more appropriate to make use of the BASIC programming facilities provided by the computer to plot a graph of voltage versus time on the video screen or on the screen of an oscilloscope. We shall leave the video display routine for the remaining experiments. In the rest of this experiment we shall describe how to interface an oscilloscope to display the acquired data. If you do not have an oscilloscope at your disposal, we suggest you read through the steps to understand how the display can be achieved.

STEP 9 The tabulated data collected by the analog-to-digital converter and stored in a file in W/R memory will be output to a digital-to-analog converter. The analog output will be used to drive the vertical Y channel of an oscilloscope and display the charging curve of the capacitor. Some simple software steps will be introduced to make the program run more smoothly under operator control. Wire the additional circuit shown in Figure 6.12.

STEP 10 Add the following lines to the BASIC PROGRAM:

```
100 LET N = 33333
110 PAUSE N
115 CLS
120 FAST                                        (for B&W models only)
130 LET L = USR 16619
130 LET L = USR 32135                           (for Color models)
140 PRINT  "PROGRAM HAS CONCLUDED"
150 STOP
```

and change the following two lines to read:

```
      * * * *
160 FOR M = 16619 TO 16639                      (for B&W models)
200 FOR M = 16619 TO 16639
```

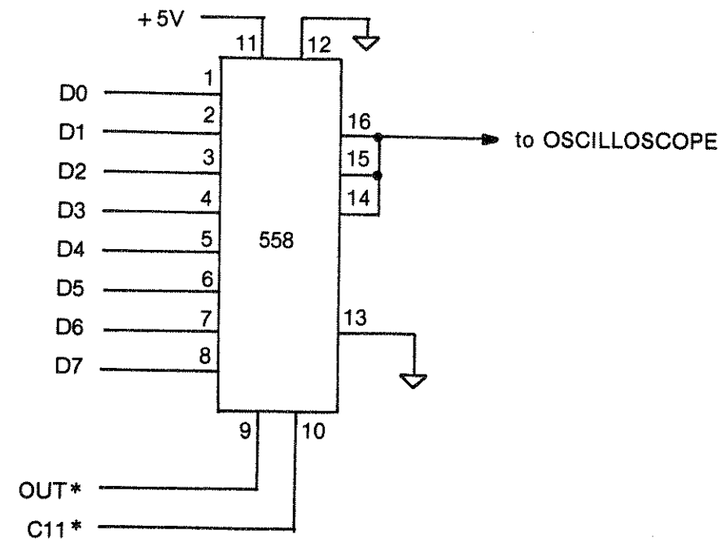



Figure 6.12 Experiment 6.1B Schematic.

```
      * * * *
160 FOR M = 32135 TO 32155                      (for Color models)
200 FOR M = 32135 TO 32155
      * * * *
```

STEP 11 Now load the decimal code for the following machine language routine which performs the digital-to-analog conversion of the stored data. RUN 160 and input the decimal codes.

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16619 / 32135	33	LD HL,NN	:Data Table starting
16620 / 32136	154	Lo N	:address
16621 /	64	Hi N	:at 16538
/ 32137	104	Hi N	:or 32154
16622 / 32138	6	LD B,N	:Register B holds the
16623 / 32139	79	N	:number of values in the file.
16624 / 32140	126	LD A,(HL)	:Place the value in Register A
16625 / 32141	211	OUT (N),A	:Output it to the DAC for

```
16626 / 32142    11    N            :conversion to analog.
16627 / 32143    35    INC HL       :Point to next value.
16628 / 32144    16    DJNZ d       :Done all data (B=0)?
16629 / 32145   250    d            :No. Jump to 16624/32140.
16630 / 32146   205    CALL NN      :Yes. Call the ROM subroutine
16631 /         187    Lo N
      / 32147   176    Lo N
16632 / 32148     2    Hi N         :test for key closure.
16633 /         124    LD A,H       :Make up a mask for
      / 32149   122    LD A,D
16634 /         133    ADD L        :when key 4 is pressed.
      / 32150   131    ADD E
16635 / 32151   254    CP N
16636 /         230    N            :B&W model code for key 4
      / 32152    11    N            :Color model code for key 4.
16637 / 32153   200    RET Z        :Return to BASIC
16638 / 32154    24    JR d         :Key 4 was not pressed so
16639 / 32155   235    d            :continue displaying the file
```

STEP 12 Note the simple interfacing required for this integrated circuit chip. Device select address 11* could be used to drive this DAC together with OUT*, because the DAC will be receiving information from the processor. If you have only 1K of W/R memory, you may have difficulty placing the whole program in memory so a number of word-saving actions will have to be taken.

STEP 13 Now that you have loaded both your machine language program and your BASIC routine, switch on your oscilloscope and center the trace. Use a voltage range of about 0.5 V/cm and switch the vertical or Y input to ac. Set the horizontal time base of the oscilloscope to 10 msec/cm until you have a picture on the screen of the oscilloscope. Then adjust to a different time base to obtain the most satisfactory trace.

STEP 14 RUN your program. As in the first part of the experiment, the table of voltage values will appear on your screen and will remain there until you press any key, whereupon BASIC line 110 will finish execution and the video screen will clear while the Z80 commences execution of the next machine language program called by the USR routine at line 130.

STEP 15 Now that the second machine language program, which starts at memory location 16619/32135, is being executed you should observe on your oscilloscope screen a plot of voltage on the vertical scale against time on the horizontal scale. The trace should be exponential. If you do not have such a plot, check all the oscilloscope settings including the automatic triggering setting. If you still do not have a picture and you are absolutely sure that the oscilloscope is not faulty, you need to check your machine language program and hardware.

STEP 16 The system should run automatically and will only return to BASIC when the 4 key is pressed. See machine code section from memory location 16630/32146 to 16639/32155, which makes use of a BASIC subroutine contained in the ZX81/1000/1500 ROM at location 699 (2 * 256 + 187) and the Spectrum/2068 ROM at location 688 to detect key closures. In this case, key 4 has been selected as the code that will cause the return on zero instruction to be executed

at memory location 16637/32153. This stops the charging curve from being displayed on the oscilloscope screen and returns control to the BASIC program at line 140. Note how use can be made of any of the self-contained Sinclair subroutines held in the BASIC ROM, by using a direct CALL instruction in your own machine code.

STEP 17 From the known properties of exponential charging curves for resistor capacitor networks as outlined in the first part of the experiment, determine the time constant from the picture on your oscilloscope screen.

STEP 18 If you do not have access to an oscilloscope you might like to try your hand at programming the Timex/Sinclair microcomputer to plot your table of voltage values. You can adapt one of the programs given in the following experiments.

SUMMARY You have now used your microcomputer to collect data and display the data on another instrument. It is also possible to send the analog signal produced by the digital-to-analog converter to a chart recorder and obtain a permanent record of the experiment.

EXPERIMENT 6.2
INTERFACING A LIGHT-SENSITIVE RESISTOR

COMPONENTS
1 * Photoresistor (CdS:3 Megohm dark)
1 * ADC0804 Analog-to-Digital Converter
1 * 1-Kohm resistor
1 * 10-Kohm resistor
1 * 150-pF capacitor

DISCUSSION Light-sensitive resistors are relatively inexpensive components that can provide a large enough change in resistance to produce a digital detector of a "light—no light" situation. They can be used, for example, in a burgular alarm or any other application involving the breaking of a light beam. By interfacing the photoresistor to an analog-to-digital converter, it is possible to detect slight changes in light intensity as well as the "full on—full off" situation. Applications can be as diverse as observing clouds passing over a solar collector or detecting change in a chemical experiment where you wish to monitor a color change in a titration experiment. By using filters in front of the light-sensitive resistor it is possible to detect specific color changes.

The photoresistor used in this experiment decreases its resistance as the intensity of light incident on its window increases. The signal of a resistance transducer must be conditioned to an analog voltage in order to interface it to an analog-to-digital converter. The circuit used to transform the resistance to a voltage is shown in Figure 6.13. The decreasing resistance of the photoresistor allows a larger current to flow through the 1-Kohm resistor and produces, by Ohm's Law, a larger voltage drop which is fed to the + input, pin 6, of the analog-to-digital converter for conversion into a digital value.

PROCEDURE

STEP 1 Wire the experiment as shown in the schematic diagram ensuring that the transparent window is directed at the source of light you wish to monitor.

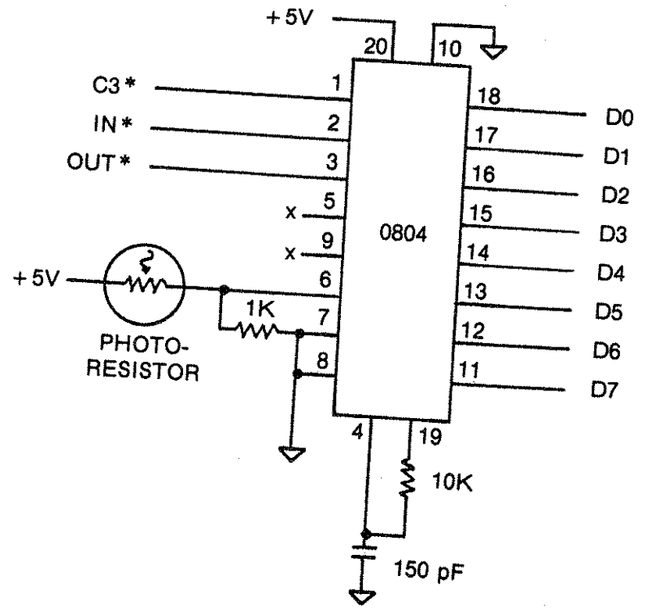


Figure 6.13 Experiment 6.2 Schematic.

STEP 2 If you have a voltmeter on hand, check the voltage output by the sensor by measuring across the 1-Kohm resistor. With no light you should have near zero voltage and with full light you should read a voltage near 5 V—showing that your light-dependent resistor circuit is functioning.

STEP 3 Load the BASIC program and the machine language routine in the usual manner. The machine language program must take care of controlling the converter as well as providing file space for the values of voltage, proportional to light intensity, which are input from the ADC. In this experiment the values input to the file will be plotted out by the BASIC program so you can detect movement over the detector. The machine language program is similar to that in Experiment 6.1, but the BASIC program has been altered to allow for plotting the values obtained. Because the converter only supplies the set of values proportional to the light intensity, these can be plotted on the vertical Y axis. To provide a graph of movement, the BASIC program provides a horizontal time displacement via the variable K as shown in lines 150 through 180.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789 123456789 123456789
   123456789 123456789 123456789 123456789 123456789
   1234567890
```

(for B&W models)


```
 20 FAST
 30 LET L = USR 16514
 40 FOR A = 16542 TO 16618 STEP 4
      * * * *
 10 CLEAR 32129                                        (for Color models)
 30 LET L = USR 32130
 40 FOR A = 32158 TO 32234 STEP 4
      * * * *
 50 PRINT ; TAB (6 - LEN "A" ); PEEK A; TAB(16 -
    LEN "(A+1)" ); PEEK (A+1); TAB(24 - LEN "(A+2)" ); PEEK
    (A+2); TAB(32 - LEN "(A+3)" ); PEEK (A;3)
 60 NEXT A
 70 PRINT  "ALL IN"
 80 LET N = 33333
 90 PAUSE N
100 CLS
110 LET A = 16452                                      (for B&W models)
110 LET A = 32158                                      (for Color models)
120 FOR X = 0 TO 63
130 LET Y = INT(( PEEK A)/6)
140 PLOT X,Y
150 LET A = A + 1
160 NEXT X
170 PAUSE N
180 STOP
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16514 / 32130	1	LD BC,NN	
16515 / 32131	20	Lo N	:Delay counter for ADC.
16516 / 32132	79	Hi N	:Number of data points.
16517 / 32133	33	LD HL,NN	:Data Table starting address
16518 / 32134	161	Lo N	:at location 16545/32161
16519 /	64	Hi N	
/ 32135	125	Hi N	
16520 / 32136	17	LD DE,NN	:Delay counter between data
16521 / 32137	255	Lo N	:acquisitions.
16522 / 32138	2	Hi N	
16523 / 32139	211	OUT (N),A	:Start ADC conversion with.
16524 / 32140	3	N	:pulse to Port 3.
16525 / 32141	13	DEC C	:Loop 20 times for delay.
16526 / 32142	32	JR NZ,d	:Done?
16527 / 32143	253	d	:No. Jump back to 16525/32141.
16528 / 32144	14	LD C,N	:Yes. Restore counter.

```
16529 / 32145    20     N
16530 / 32146   219     IN A,(N)      :Read ADC value from
16531 / 32147     3     N             :Port 3.
16532 / 32148   119     LD (HL),A     :Store data value in memory.
16533 / 32149    35     INC HL        :Point to next entry in table.
16534 / 32150    27     DEC DE        :Time delay between values.
16535 / 32151   122     LD A,D        :Is DE counter zero?
16536 / 32152   179     OR E
16537 / 32153    32     JR NZ,d       :No. Keep counting down
16538 / 32154   251     d             :by jumping back to 16535/32151.
16539 / 32155    16     DJNZ d        :Table full (B=0)?
16540 / 32156   235     d             :No. Jump to 16520/32136 again.
16541 / 32157   201     RET           :Yes. Return to BASIC.
```

STEP 4 RUN your program. If you wish to observe your hand motion over the photoresistor and have the results of the movement recorded on the video screen, you will need to synchronize the movement of your hand with the pressing of the ENTER key after the RUN key has been pressed. This may take a little practice.

STEP 5 Having stored data, presumably about your hand movement over the photoresistor, then by pressing any key on the board, the Timex/Sinclair will respond by coming out of the pause loop and displaying a plot of your hand movement over the photoresistor on the video screen.

STEP 6 You can vary the timing constant in register C (location 16515/32131) to some larger value to obtain the data over a longer period of time.

EXPERIMENT 6.3
ELASTIC BEAM MEASUREMENTS USING STRAIN GAGES

COMPONENTS 2 * Strain gages; resistance = 120 ohms Type CEA-06-125UW-120 (Measurements Group)
2 * 120-ohm 1/4-watt resistors matched to ±1%
1 * LM358 Dual Bi-FET op amp
1 * ADC0804 Eight-bit Analog-to-Digital Converter
1 * each 10-, 16-, 33-Kohm resistor
1 * 150-pF capacitor
1 * 0.47-μF capacitor
1 * Hacksaw blade and C-clamp

DISCUSSION Strain gages are resistance transducers made up of fine wire grids which are mounted rigidly to the surface of the object under study. When a stress, in the form of a force or pressure, is applied to the object the induced strain causes the surface to yield (elongate in some direction) to a degree dependent on its elastic properties. As the surface yields, the resistance of the strain gage changes. Strain gages are used extensively in engineering applications to measure stresses in structures as diverse as airplanes and bridges. Because the extent of strain is very small, the corresponding signal generated by the transducer is also very small. In addition,

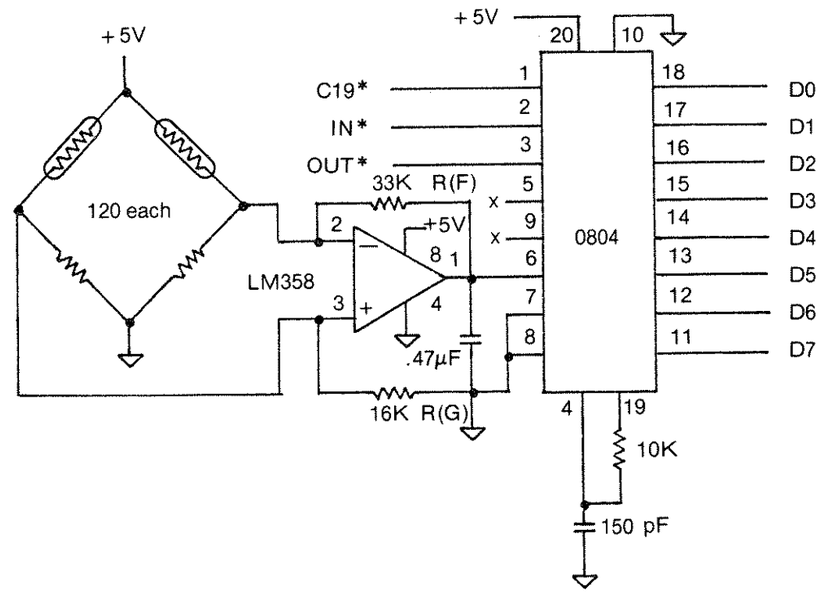


Figure 6.14 Experiment 6.3 Schematic.

the resistance of the strain gage also depends on the ambient temperature which introduces additional errors in the signal.

The technique to convert the resistance to an analog voltage for analog-to-digital conversion employs the Wheatstone bridge. The Wheatstone bridge is a network of four resistors laid out in a square with each resistor forming one side of the square and connected at the corners to two of the others (see Figure 6.14). When a voltage is applied to diagonally opposite corners and the resistances are adjusted such that the voltage drop across the other diagonal is zero, then the bridge is balanced. At balance, the product of the two resistances in opposite sides is equal to the product of the other two resistances. From this relationship:

$$R(1)*R(3) = R(2)*R(4).$$

If any three values are known, the fourth can be calculated.

By placing the strain gages in two of the arms of a balanced Wheatstone Bridge it is possible to put the bridge out of balance as the elastic straining force is increased. A high input impedance FET (Field Effect Transistor) amplifier can then be used to measure the out of balance voltage and produce a larger voltage proportional to straining force. This voltage can then be converted to digital form for the microcomputer by using an analog-to-digital converter.

In the first part of the experiment, a simple mechanics experiment is performed on the deformation of an elastic body to a deforming force. The deformation follows a simple relationship referred to as Hooke's Law:

$$F = kX$$

where F is the deforming force, X represents the displacement of the elastic body from its equilibrium position, and k is the elastic constant of the body. In the second part of the experiment, we investigate the damped harmonic oscillation of the elastic beam. The use of strain gages enables us to measure the disturbance from equilibrium of an elastic body by measuring the change in resistance as the strain in the elastic body changes.

PROCEDURE

STEP 1 Strain gages should be obtained with a resistance as close to a preferred range of resistor which you have available to you, or if at all possible close to a value of precision resistor which you might have in stock. Resistor values such as 100, 120, and 150 ohms should be suitable. The strain gages we used had a measured resistance of 109 ohms so we used 110-ohm high stability resistors initially in the arms of the bridge. We were able to complete the experiment using ordinary 1/4-watt carbon resistors by altering the values of R(F) and R(G) to bring the output of the amplifier to mid-range approximately 1.9 V with a load of 50 g on the elastic beam. Some trial and error may be necessary but an ordinary volt-ohmeter should be sufficient to enable you to set the values of components for satisfactory operation of your circuit.

Attach your strain gages to the elastic beam (hacksaw blade) a day or two before you anticipate carrying out the experiment. The strain gages were glued to a hacksaw blade on either side at the same distance from an end. Allow about 6 inches (15 centimeters) from an end to the clamp or fulcrum about which the hacksaw blade will be strained. Glue the strain gages as close to this point as possible and epoxy glue the leads to the blade to prevent flexing. Roughen the blade with No. 01 sandpaper or equivalent and wipe down with alcohol prior to gluing. This arrangement of strain gages doubles the change for straining (doubles the change in resistance) and still compensates for temperature changes. Use shielded twin core cable to connect the gages to the circuitry on your socket board. Twist the leads from each gage and join the two shielding braids at the gage end. There should be no need to ground the hacksaw blade or the shielding itself, or attach the braid to the hacksaw blade.

STEP 2 If at all possible match your Wheatstone bridge resistors as closely as possible to each other and to the resistance of the strain gages. If you have high stability resistors of the values required on hand, use them in the bridge arrangement.

STEP 3 Suspend a 50-g weight about 6 inches (15 centimeters) from your fulcrum and allow the system to come to equilibrium (see Figure 6.15). Using a voltmeter at the output of the LM358 amplifier, pin 1, measure the dc voltage. If the value you measure is grossly different from about 2 V, adjust the resistor, R(G), by about 1-Kohm increments until you bring the output voltage to about 2 V.

STEP 4 If your strain gages and resistors are so out of balance that a reading of 2 V cannot be obtained, reduce the gain of the amplifier significantly by putting R(F) = R(G) = 1 Kohm and look for an output of about 2 V. Now slowly increase R(F) and adjust the value of R(G) to keep the bridge on an output of about 2 V while the gain resistor, R(F), is increased. Any mismatch of R(F) and R(G) will lead to increased noise, but the averaging routine in the machine language program should eliminate most of the noise from that source. With sufficient gain your analog-to-digital converter will give a reasonable digital output per 1 g of weight.

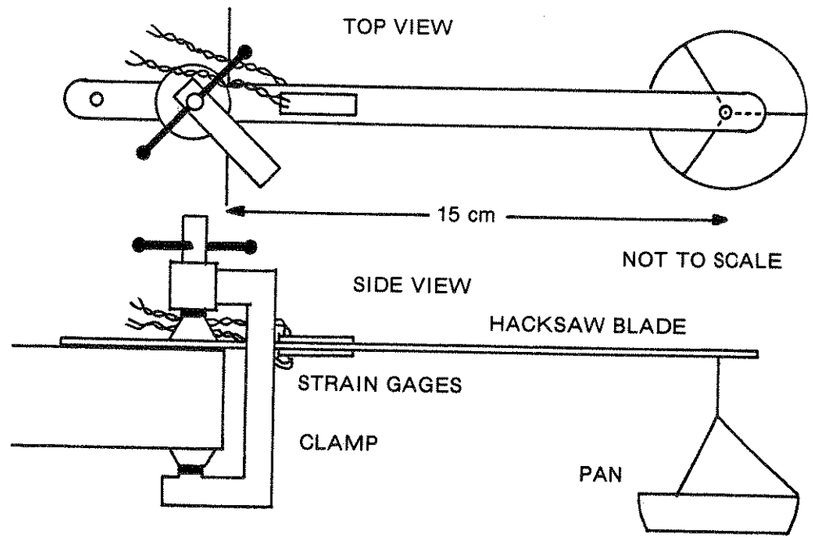


Figure 6.15 Elastic Beam Apparatus.

STEP 5 The software we have used will display a decimal value corresponding to the deflection voltage output from the amplifier. Inspection of the machine language program will show a machine code averaging program as well as a fairly long time delay loop between readings. The time delay loop occurs between 16522/32138 and 16529/32145. After the analog-to-digital converter has been started with a device select pulse at memory location 16544/32160, the system makes use of the Z80 instruction DJNZ to time out a small delay loop to allow the converter time to complete the conversion. DJNZ automatically decrements register B, loaded at location 16542/32158, until register B is 0. DJNZ also requires the calculation of a relative displacement in this case just back one location, or counting the displacement as position 255, the displacement becomes 255 − 1 = 254.

The averaging routine commences at location 16563/32179 where register C is loaded with the number 4 (that is the number of times registers D and E will be rotated to the right through carry). Because the number of readings taken (16) was loaded at 16548/32164 (17 − 1) and registers D and E contain the sum of the readings, rotation to the right by one bit is, in effect, division by 2, then by four times is division by 16, so that the result left in the E register is the average of the 16 readings. It is this value that is loaded into memory for later retrieval by the BASIC program. The BASIC program makes use of the tabulation procedure adopted in other programs. Setting the BASIC variable N to a number greater than 32,768 causes an indefinite PAUSE period (on the B&W models) that is terminated by pressing any key. Load the following BASIC program and machine language routine.

BASIC PROGRAM

```
 10 REM 123456789 123456789 123456789 123456789 123456789
    123456789 123456789 123456789 123456789 123456789
    123456789 123456789 1234567890
 20 FAST
 30 LET L = USR 16514                                    (for B&W models)
 40 FOR A = 16579 TO 16599 STEP 4
      * * * *
 10 CLEAR 32129
 30 LET L = USR 32130                                    (for Color models)
 40 FOR A = 32195 TO 32215 STEP 4
      * * * *
 50 PRINT TAB(4 - LEN STR$ PEEK A); PEEK A; TAB(12 - LEN STR$
    PEEK (A+1)); PEEK (A+1); TAB(20 - LEN STR$ PEEK (A+2));
    PEEK (A+2); TAB(28 - LEN STR$ PEEK (A+3)); PEEK (A+3)
 60 NEXT A
 70 PRINT  "ALL IN"
 80 LET N=33333
 90 PAUSE N
100 CLS
110 GOTO 30
120 FOR M = 16514 TO 16577                               (for B&W models)
120 FOR M = 32130 TO 32193                               (for Color models)
130 INPUT N
140 POKE M,N
150 PRINT M;  "=" ; PEEK M,;
180 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	33	LD HL,NN	:Pointer to Data Table
16515 / 32131	195	Lo N	:starting address at
16516 /	64	Hi N	:location 16579
/ 32132	125	Hi N	:or 32195.
16517 / 32133	6	LD B,N	:Number of entries
16518 / 32134	20	N	:in Data Table.
16519 / 32135	17	LD DE,NN	:Delay counter between
16520 / 32136	255	Lo N	:readings = 4 * 255
16521 / 32137	4	Hi N	:= 1020.
16522 / 32138	29	DEC E	:Begin delay countdown.
16523 / 32139	32	JR NZ,d	
16524 / 32140	253	d	

16525 / 32141	21	DEC D	
16526 / 32142	122	LD A,D	
16527 / 32143	131	OR E	
16528 / 32144	32	JR NZ,d	:Is DE zero?
16529 / 32145	248	d	:No. Go back to 16522 / 32138.
16530 / 32146	24	JR d	:Yes. Jump over End of
16531 / 32147	6	d	:Routine to 16538 / 32154.
16532 / 32148	115	LD (HL),E	:Store the data average.
16533 / 32149	35	INC HL	:Point to next Table entry.
16534 / 32150	16	DJNZ d	:Collected 20 averages?
16535 / 32151	239	d	:No. Jump to 16519 / 32135.
16536 / 32152	201	RET	:Yes. Back to BASIC.
16537 / 32153	0	NOP	
16538 / 32154	17	LD DE,NN	:Initialize sum of 16
16539 / 32155	0	Lo N	:readings per data
16540 / 32156	0	Hi N	:point to zero.
16541 / 32157	197	PUSH B	:Save data counter.
16542 / 32158	6	LD B,N	:Load wait counter.
16543 / 32159	20	N	
16544 / 32160	211	OUT (N),A	:Start ADC conversion
16545 / 32161	19	N	:with pulse to Port 19.
16546 / 32162	16	DJNZ d	:Wait for ADC to finish.
16547 / 32163	254	d	
16548 / 32164	14	LD C,N	:Number of readings to be
16549 / 32165	17	N	:averaged = 17 − 1 = 16.
16550 / 32166	13	DEC C	:Collected 16 readings?
16551 / 32167	40	JR Z,d	:Yes. Jump to 16563 / 32179.
16552 / 32168	10	d	
16553 / 32169	167	AND A	:No. Clear the carry flag.
16554 / 32170	219	IN A,(N)	:Read ADC at Port 19.
16555 / 32171	19	N	
16556 / 32172	131	ADD E	:Add sum of readings,
16557 / 32173	95	LD E,A	:and store new sum.
16558 / 32174	48	JR NC,D	:Carry overflow from E?
16559 / 32175	246	d	:No. Go back to 16550 / 32166.
16560 / 32176	20	INC D	:Yes. Collect overflow in D
16561 / 32177	24	Jr d	:for 16 bit sum, then go
16562 / 32178	243	d	:back to 16550 / 26771.
16563 / 32179	14	LD C,N	:Set up divide-by-2
16564 / 32180	4	N	:for 4 times = 1 / 16.
16565 / 32181	203	p	:Get average of readings
16566 / 32182	26	RR D	:by dividing by 16 with
16567 / 32183	203	p	:Right Rotates of D and E.
16568 / 32184	27	RR E	
16569 / 32185	13	DEC C	
16570 / 32186	32	JR NZ,d	:Done with division?
16571 / 32187	249	d	:No. Go to 16565 / 32181.
16572 / 32188	193	POP BC	:Yes. Restore data counter.
16573 / 32189	123	LD A,E	
16574 / 32190	47	CPL	

```
16575/32191    95     LD E,A
16576/32192    24     JR d
16577/32193    210    d          :Jump to End of Routine
                                 :at 16532/32148.
```

STEP 6 Add different weights and record the decimal output. A graph of output versus added weight should yield a very good straight line.

STEP 7 The strain gages attached to a hacksaw blade as an elastically strained system can be used to display the damped simple harmonic motion of a vibrating system. Pay particular attention to the positioning of the fulcrum of the beam and the rigid clamping of the stationary part of the beam. We noted that if the stationary part of the beam was not firmly fixed to the bench, secondary oscillations occurred which tended to mask out the original motion (and led to some strange results). We also noted that if the wires connecting the strain gages to the bridge were allowed to flex, errors in your graph could result. Check that your bridge output is correctly adjusted when the system is not vibrating. Difficulties experienced at this point could be due to not being able to balance the bridge, again use a small value trim pot (10 ohms) in series with one of your fixed resistors and adjust it until your bridge is balanced.

STEP 8 The machine language program is identical to that used for the elastic beam apart from changes to the major time delay loop to allow a sample time of a few seconds of the damped motion to be taken and the extending of the data file from 20 readings to 70 readings. Modify the machine language program by executing the following direct commands:

```
POKE 16518,70      (B&W)
POKE 16521,12
POKE 32134,70      (Color)
POKE 32137,12
```

STEP 9 The BASIC program is extended appreciably. The table length in line 40 has been extended to 16649/32265. The larger file of values can then be plotted on the video screen with the program commencing at line 200. To allow for the small range of output voltages and for the fact that smaller weights can cause smaller deflections, the first 20 values are scrutinized to determine the maximum and minimum values of the excursions of the bridge when the hacksaw blade is displaced from equilibrium and allowed to oscillate. From these values the factors required to produce full screen deflection for the first oscillation with the equilibrium position on the midscreen line can be calculated. *V* is the midscreen deflection factor and *W* is the scaling factor.

The program will not run if you have the system in equilibrium so that the denominator term in line 300 becomes 0 hence the need for line 310 and 390. The Timex/Sinclair would otherwise break the program and return to the video screen with the error report:"out of range."

Add the following lines to the BASIC program:

```
 40 FOR A = 16579 TO 16649 STEP 4      (for B&W models)
 40 FOR A = 32195 TO 32265 STEP 4      (for Color models)
200 LET A = 16579                      (for B&W models)
200 LET A = 32195                      (for Color models)
210 LET M = PEEK A
220 LET X = M
```



```
230 LET N = M
240 FOR A = 16580 TO 16599             (for B&W models)
240 FOR A = 32196 TO 32215             (for Color models)
250 LET M = PEEK A
260 IF M > X THEN LET X = M
270 IF M < X THEN LET N = M
280 NEXT A
290 LET V = (X + N)/2
300 LET W = 42/(X - N)
310 IF X - N <= 0 THEN GOTO 390
320 LET A = 16579                      (for B&W models)
320 LET A = 32195                      (for Color models)
330 FOR K = 0 TO 63
340 LET Y = INT(((PEEK A) - V)* W) + 20
350 PLOT K,Y
360 LET A = A + 1
370 NEXT K
380 STOP
390 PRINT "YOU ARE IN EQUILIBRIUM START THE OSCILLATION AND
    RUN AGAIN"
```

STEP 10 Attached a weight on the beam with adhesive and depress it from its equilibrium position. The experiment can be performed using various weights and deflections. By measuring the amount of the first deflection and the time of decay, the damping constants of the system can be determined.

STEP 11 Type in RUN and just after you press ENTER, release the elastic beam.

STEP 12 A damped simple harmonic waveform should appear on your screen. Measurements can be made directly or handled by additional software. If a picture does not appear, the problem is most likely in your additional software so go back and check everything again.

EXPERIMENT 6.4

TO CONVERT VOLTAGE APPLIED TO A MOTOR TO A DECIMAL VALUE

COMPONENTS
- 1 * ADC0804 Eight-bit Analog-to-Digital Converter
- 1 * 3-V DC motor plus 3-V power supply or batteries
- 1 * LM358 Dual Op Amp
- 1 * 1-Kohm resistor
- 2 * 10-Kohm potentiometers
- 1 * 150-pF capacitor
- 1 * 1.0-μF capacitor
- 1 * DC voltmeter

DISCUSSION Knowledge of the relationship between motor speed and applied voltage would enable the motor be be used as a tachometer or, if a fan blade were attached to the shaft, as an indicator of air speed. Because the maximum voltage that can be applied to the motor is only 3 V, use will be made of the characteristics of the analog-to-digital converter to adjust its reference voltage and zero offset to cause the full scale digital output of the converter, 0 to 255, to correspond to the 0 to 3 V range.

The reference voltage adjustment and zero offset circuitry is shown in Figure 6.16. The values of the two potentiometers are not critical, values between 1 Kohm and 10 Kohm should be acceptable. The voltage applied to the motor is fed directly to the V+ input, pin 6, of the converter. When the program is run, the voltmeter will give a reading of the applied voltage and the microcomputer will display the decimal equivalent on the video screen.

Once the system is calibrated, the microcomputer could be used to convert the decimal values directly to voltages and plot a table for you.

PROCEDURE

STEP 1 Connect the circuit as shown in the schematic using a separate dc source to power the motor. If a power supply is to be used rather than batteries, you must ensure that the voltage output to the motor is strictly limited to below 5 V to protect the input of the converter from excessive voltage.

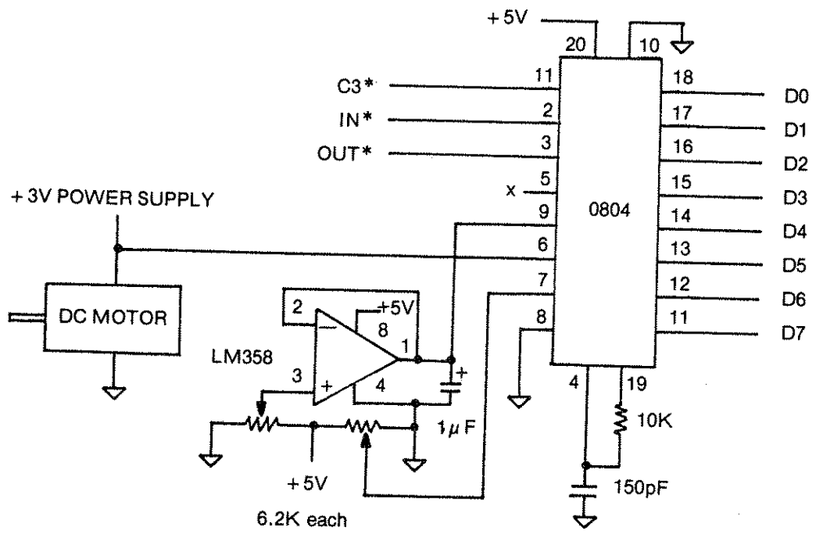


Figure 6.16 Experiment 6.4 Schematic.

STEP 2 Load the BASIC program and the machine language routine.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 12345        (for B&W models)
20 FAST
30 LET L = USR 16514
40 FOR A = 16543 TO 16547
     * * * *
10 CLEAR 32129                                    (for Color models)
30 LET L = USR 32130
40 FOR A = 32159 TO 32163
     * * * *
50 PRINT A
60 NEXT A
70 STOP
80 FOR M = 16514 TO 16541                         (for B&W models)
80 FOR M = 32130 TO 32157                         (for Color models)
90 INPUT N
100 POKE M,N
110 PRINT M;  "=" ; PEEK M,;
140 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	1	LD BC,NN	:Load counters for
16515 / 32131	20	Lo N	:converter delay and
16516 / 32132	4	Hi N	:number of measurements.
16517 / 32133	33	LD HL,NN	:Pointer to starting
16518 / 32134	159	Lo N	:address of Data Table
16519 /	64	Hi N	:at location 16543
/ 32135	125	Hi N	:or 32159.
16520 / 32136	17	LD DE,NN	:Delay counter between
16521 / 32137	255	Lo N	:data measurements.
16522 / 32138	2	Hi N	
16523 / 32139	211	OUT (N),A	:Start ADC conversion
16524 / 32140	3	N	:with pulse to Port 3.
16525 / 32141	13	DEC C	:Wait for conversion.
16526 / 32142	32	JR NZ,d	:Long enough time?
16527 / 32143	253	d	:No. Jump back to 16525/32141.
16528 / 32144	14	LD C,N	:Yes. Restore delay counter.
16529 / 32145	20	N	
16530 / 32146	219	IN A,(N)	:Read ADC from

```
16531 / 32147     3      N           :Port 3.
16532 / 32148   119      LD (HL),A   :Store value in table
16533 / 32149    35      INC HL      :and point to next entry.
16534 / 32150    27      DEC DE      :Start waiting between
16535 / 32151   122      LD A,D      :measurement.
16536 / 32152   179      OR E
16537 / 32153    32      JR NZ,d
16538 / 32154   251      d           :Downcount finished?
16539 / 32155    16      DJNZ d      :No. Jump back to 16534/32150.
16540 / 32156   235      d           :All measurements taken?
16541 / 32157   201      RET         :No. Jump back to 16520/32136
                                     :Yes. Return to BASIC.
```

STEP 3 Connect the voltmeter across the motor with the positive lead connected to the converter input to measure the input voltage directly.

STEP 4 Increase the voltage applied to the motor until it begins to revolve. RUN the program and record the decimal display and the voltage applied. If no value is recorded, then adjust potentiometer 1 until a reading is obtained.

STEP 5 Increase the voltage to the maximum recommended and RUN the program again. If the decimal values displayed do not show the maximum value of 255, then alter potentiometer 2 until a value of 255 is obtained. If the decimal value displayed is 255, then decrease potentiometer 2 until a reading of 254 is obtained. This should result in a full scale decimal reading for your 3-V motor when running at maximum speed.

STEP 6 Now continue the experiment recording values of voltage and decimal count over the full speed range of your motor and check your results for linearity.

STEP 7 Check the stall speed voltage by reducing the voltage applied to the motor slowly from the value needed to start the motor initially. At one voltage the motor will stop running. The difference between the stall voltage and start voltage represents the amount of energy your power source needs to supply to the motor windings to overcome friction, inertia, and magnetic flux losses.

STEP 8 Compare the results of this experiment with those obtained in Experiment 5.2, where you determined the relationship between applied voltage and rotational speed of the dc motor.

EXPERIMENT 6.5
TEMPERATURE RECORDING AND DISPLAY

COMPONENTS
1 * AD590 Temperature sensor
1 * ADC0804 Analog-to-Digital Converter
2 * 10-Kohm ten-turn potentiometer
1 * 10-Kohm resistor
1 * 5.1-Kohm resistor

1 * 10-μF capacitor
1 * 150-pF capacitor

DISCUSSION Temperature is an often sought after measurement, whether it be for monitoring the temperature in your home, greenhouse, fish tank, or the outside weather. Using modern semiconductor technology and your Timex/Sinclair microcomputer, both control and sensing of temperature is easily accomplished together with the ability to display the information on the video screen. Once again only a minimum of electronic components are required along with your interface unit.

The temperature sensor type AD590 is a low-cost solid state sensor which produces an output current equal to 1 μA (1E-6 ampere) per degree Kelvin change in temperature, that is the output current at 300K (degrees Kelvin) = 27 C (degrees celsius) = 81 F (degrees Fahrenheit) is 300 * 1 μA or 300 μA. Such a current can flow through a resistor to produce a potential drop which is also proportional to temperature.

This variation of voltage with temperature is another example of analog information and before we can make use of such information we must convert the analog signal into a digital signal using the ADC0804 analog-to-digital converter. This converter is quite useful because "offsetting" voltages can be applied to its minus input (pin 7) and its voltage reference input (pin 9) to change the range of input analog voltage over which the converter will operate.

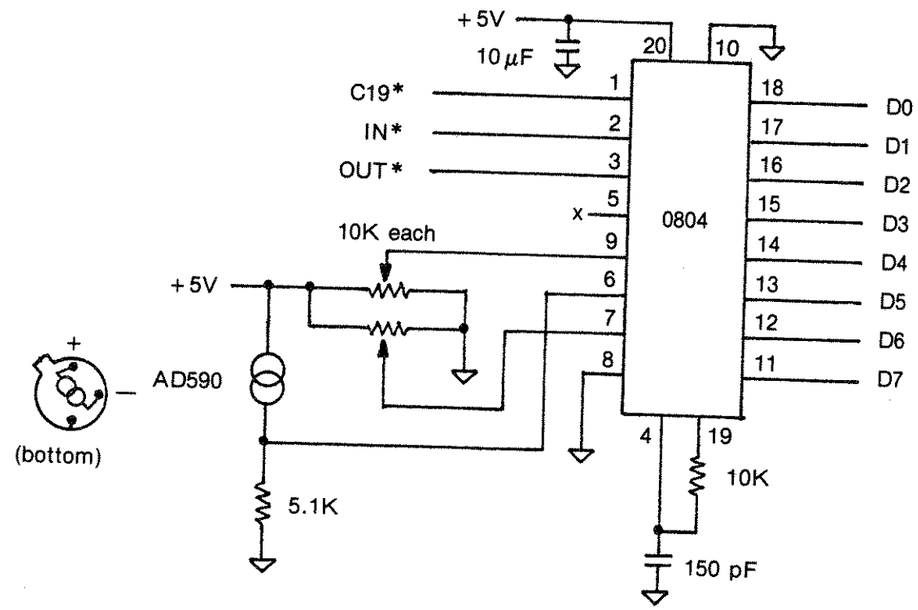


Figure 6.17 Experiment 6.5 Schematic.

By checking the schematic diagram, Figure 6.17, you will see that the AD590 has been placed in series with a 5.1-Kohm resistor. This is not a critical value and the experiment will work equally well with a 4.7- or 5.6-Kohm resistor. You should be able to calculate, using Ohm's Law, that the voltage drop across the 5.1-Kohm resistor will be 1.53 V at 300K. By adjusting the trim pot connected to pin 7 of the ADC0804 to give a voltage of 1.35 V and by adjusting the trim pot attached to pin 9 to give a voltage of 0.245 V, a satisfactory range of values should be achieved. Note that careful adjustment of this latter trim pot is required. A reading below about 0.1 V will produce no worthwhile data, while values above 0.27 V may cause the converted digital temperature values to at first decrease as the temperature rises.

PROCEDURE

STEP 1 Arrange your components on the socket board close to the bus end of the board to leave room for the addition of more chips in the next experiment. Wire your components with the power disconnected according to the schematic shown in Figure 6.17.

STEP 2 In the experiment, successive temperature values will be recorded and placed in W/R memory in a file 63 bytes long. When the file is filled, each new value added will displace the earliest entered value, providing a continually updated file for display on the video screen. By checking the machine language program you will see that the first task of the program is to clear the file space in memory by exclusive ORing, XOR A (setting to 0), each memory location. This subroutine should only be executed once when you first run the program so a RETurn instruction has been provided at memory location 16525/32141. The second USR routine starts the machine language program at location 16526/32142 which starts the conversion with an OUT 19 Device Select Pulse. The first task of this data collection program is to update the file and make space for the acquired eight-bit byte from the converter. After this adjustment in the file has taken place the next converted value will be ready to input at location 16520 using an IN 19 instruction. The value is then stored on file and the machine language routine returns to BASIC to await the next command to collect a temperature value. This command is initiated after a PAUSE time set by the BASIC program.

Load the BASIC program and the machine language routine.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789 123456789
   123456789 123456789 123456789 123456789 123456789 123456
20 FAST
30 LET C = USR 16514                                    (for B&W models)
     * * * *
10 CLEAR 32129
30 LET C = USR 32130                                    (for Color models)
     * * * *
40 LET N = 33333
50 PRINT "PRESS ANY KEY TO START RECORDING TEMPERATURE"
60 PAUSE N
70 LET L = USR 16526
70 LET L = USR 32142                                    (for B&W models)
                                                        (for Color models)
```



```
 80 CLS
     * * * *
130 PRINT PEEK 16556                                    (for B&W models)
140 LET A = 16556
     * * * *
130 PRINT PEEK 32172                                    (for Color models)
140 LET A = 32172
     * * * *
150 FOR X = 0 TO 63
160 LET Y = INT((( PEEK A )/4) - 20)
170 PLOT X,Y
180 LET A = A + 1
190 NEXT X
200 LET N = 300
210 PAUSE N
220 CLS
230 GOTO 70
240 FOR M = 16514 TO 16547                              (for B&W models)
240 FOR M = 32130 TO 32163                              (for Color models)
* * * * * * * * * *
250 INPUT N
260 POKE M,N
270 PRINT M;  "=" ; PEEK M,;
280 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENTS
B&W / Color			
16514 / 32130	33	LD HL,NN	:Clear Table subroutine.
16515 / 32131	172	Lo N	:Pointer to Table starting
16516 /	64	Hi N	:address at 16556
/ 32132	125	Hi N	:or 32172.
16517 / 32133	22	LD D,N	:Number of entries.
16518 / 32134	63	N	
16519 / 32135	175	XOR A	:Zero accumulator.
16520 / 32136	119	LD (HL),A	:Load the Table with 0's.
16521 / 32137	35	INC HL	
16522 / 32138	21	DEC D	
16523 / 32139	32	JR NZ,d	:Table filled?
16524 / 32140	250	d	:No. Go to 16519/32135.
16525 / 32141	201	RET	:Yes. Return to BASIC.
16526 / 32142	211	OUT (N),A	:Start ADC conversion with
16527 / 32143	19	N	:pulse to Port 19.
16528 / 32144	33	LD HL,NN	:Point to first entry in Table

```
16529 / 32145    172    Lo N          :at location
16530 /           64    Hi N          :16556 or
      / 32146    125    Hi N          :32172.
16531 / 32147     22    LD D,N        :Number of entries.
16532 / 32148     63    N
16533 / 32149    126    LD A,(HL)     :Get first entry.
16534 / 32150     35    INC HL
16535 / 32151     78    LD C,(HL)     :Get next entry.
16536 / 32152    119    LD (HL),A     :Bump entry back.
16537 / 32153    121    LD A,C        :Hold next entry.
16538 / 32154     21    DEC D         :Downcount entries.
16539 / 32155     32    JR NZ,d       :All moved back one?
16540 / 32156    249    d             :No. Go to 16534/32150.
16541 / 32157    219    IN A,(N)      :Yes. Read the ADC
16542 / 32158     19    N             :at Port 19.
16543 / 32159     33    LD HL,NN      :Point to first entry
16544 / 32160    172    Lo N          :at location 16556/32172.
16545 /           64    Hi N
      / 32161    125    Hi N
16546 / 32162    119    LD (HL),A     :Store new entry there.
16547 / 32163    201    RET           :Go back to BASIC.
   :      :      :     :
16556 / 32172     :     NEWVAL
```

STEP 3 With the temperature of the sensor near 300K check the voltage input to pin 6 of the ADC0804. Use a digital voltmeter if available and check that a voltage close to 1.5 V is noted. If a voltage very different from this is observed, you need to check the polarity orientation of your AD590 and the value of the series resistor just in case you put in a 510-ohm resistor or a 51-Kohm resistor.

STEP 4 With 1.5 V on pin 6, adjust the voltage on pin 7 to 1.35 V and that on pin 9 to 0.245 V, then RUN your program and check the decimal value printed out. A value close to 80 should be observed. This value is only an example, as variations in component values could cause significant variations and you can adjust the range anyway with the trim pots. If you keep obtaining values 255 or 0, then the chances are that the trim pot attached to pin 9 needs careful adjustment around the suggested value to bring your decimal readout into range.

STEP 5 We obtained decimal readout values close to 80 and were able to increase this to about 180 using the heat from a 12-V light bulb.

STEP 6 After careful adjustment of the voltages you can use the sensor and microcomputer to track temperature changes of about one degree K. A good quality thermometer could be used to calibrate your system and the BASIC program could then be easily altered to read out temperatures in degrees Kelvin, Celsius, or Fahrenheit. This is left as an exercise for the readers.

STEP 7 Save your circuit and program if you are going to continue with the next experiment.

SUMMARY This experiment indicates how useful laboratory measurements can be made using a minimum of components with your microcomputer and demonstrates how visual information can be displayed on the video screen.

EXPERIMENT 6.6
TEMPERATURE CONTROL

COMPONENTS
- 1 * AD590 Temperature sensor
- 1 * 74LS32 Quad OR Gate
- 1 * 74LS74 Latch
- 1 * ADC0804 Analog-to-Digital Converter
- 2 * 10-Kohm trim pots
- 1 * 10-Kohm resistor
- 1 * 150-pF capacitor
- 1 * Solid state relay: Sigma 226 or equivalent. TTL input - ac mains output 2 A
- 1 * 12-V 12-watt bulb
- 1 * 12-V ac transformer or equivalent

DISCUSSION Experiment 6.5 described how the temperature of an environment could be sensed and the collected data displayed on your video screen with the aid of a microcomputer. There are many instances in the world around us when the control of temperature between set limits is also of importance, for example, in maintaining the temperature of a greenhouse or of an oven.

In this experiment, we demonstrate the principles involved in automatic control using high power devices. Various temperature control schemes can be visualized, one where the temperature is maintained constant within ±1 degree and another where the temperature is allowed to cycle between set temperature limits which might be 10 or 20 degrees apart or whatever the operator and system determine.

We will pursue the second suggestion above because it highlights the software required by the microcomputer to seek out limits set by an operator and provides us with the means of automatically keeping the temperature between set values by turning on and off a solid state relay which in turn controls the current flowing through a low voltage bulb. If the bulb is placed in contact with the temperature sensor then the sensor will heat up when the bulb is turned on and cool down when the bulb is turned off.

The solid state relay itself is an example of current technology because a dc input voltage of between 3 and 30 V (such as the TTL logic 1 state of +5 V) can control an ac output voltage of 110 to 240 V RMS at 1 or 2 A. This indicates how most household machines can be controlled by your microcomputer.

CAUTION: In many parts of the world ac mains voltages are at a level of 240 V RMS, and it is illegal for a person without an electrician's certificate to wire equipment to mains-operated machines. It is also potentially lethal to experiment with such circuits. We strongly advise readers of this book to seek qualified assistance should you wish to interface your household machines to your microcomputer. None of the experiments in this book are to be used with mains-operated machines.

With this caution in mind, we have found that our experiments that involve the use of ac voltages work quite satisfactorily from 12 or 15 V ac. So the reader should have no fear of receiving an electrical shock from carrying out any of our experiments.

PROCEDURE

STEP 1 If you are performing this experiment following Experiment 6.5, then you will only need to add the extra components indicated in the schematic, Figure 6.18. Otherwise, wire the total circuit, mounting the components in the same physical position as shown in the schematic. In this way there should be sufficient room on the socket board to plug in your solid state relay.

STEP 2 If you have saved the program from the previous experiment, you should load it again and modify the BASIC program from line 40 through to line 120 and EDIT line 130: 56 to 86; line 140: 56 to 86; line 230: 70 to 120; line 240: 53 to 83; line 280: 53 to 83. This procedure should save you some programming time. You must then of course RUN 240 and enter the machine language program for this experiment.

BASIC PROGRAM

```
10 REM 123456789 123456789 123456789 123456789 123456789
   123456789 123456789 123456789 123456789 123456789
   123456789 123456789 123456
20 FAST                                              (for B&W models)
30 LET C = USR 16514
     * * * *
10 CLEAR 32129                                       (for Color models)
30 LET C = USR 32130
     * * * *
40 PRINT  " INPUT HIGH TEMPERATURE LIMIT"
50 INPUT H
```

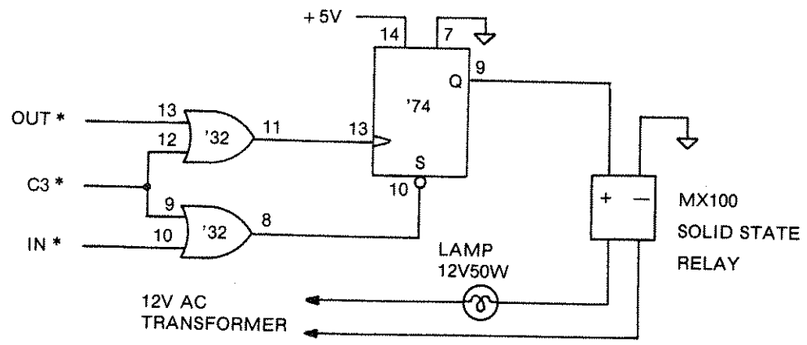


Figure 6.18 Experiment 6.6 Schematic.


```
 60 CLS
 70 PRINT ''INPUT LOW TEMPERATURE LIMIT''
 80 INPUT W
      * * * *
 90 POKE 16573,H                                     (for B&W models)
110 POKE 16574,W
110 LET L = USR 16517
120 LET D = USR 16531
130 LET A = 16575
      * * * *
 90 POKE 32189,H                                     (for Color models)
100 POKE 32190,W
110 LET L = USR 32133
120 LET D = USR 32147
130 LET A = 32191
      * * * *
140 PRINT PEEK A
150 FOR X = 0 TO 63
160 LET Y = INT(((PEEK A)/4) - 20)
170 PLOT X,Y
180 LET A = A + 1
190 NEXT X
200 LET N = 300
210 PAUSE N
220 CLS
230 GOTO 120
240 FOR M = 16514 TO 16572                           (for B&W models)
240 FOR M = 32130 TO 32188                           (for Color models)
250 INPUT N
260 POKE M,N
270 PRINT M;  "=" ; PEEK M,;
280 NEXT M
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DECIMAL CODE	INSTRUCTION MNEMONIC	COMMENT
B&W / Color			
16514 / 32130	211	OUT (N),A	:Clear controller latch
16515 / 32131	3	N	:at Port 3.
16516 / 32132	201	RET	:Back to Basic
16517 / 32133	33	LD HL,NN	:Clear Table subroutine.
16518 / 32134	191	Lo N	:Pointer to Table starting
16519 /	64	Hi N	:at location 16575
/ 32135	125	Hi N	:or 32191.
16520 / 32136	22	LD D,N	:Number of entries.

```
16521 / 32137    63    N
16522 / 32138   175    XOR A          :Zero accumulator.
16523 / 32139   119    LD (HL),A      :Load Table with 0's.
16524 / 32140    35    INC HL
16525 / 32141    21    DEC D
16526 / 32142    32    JR NZ,d        :Table filled?
16527 / 32143   250    d              :No. Go to 16522/32138.
16528 / 32144   219    IN A,(N)       :Yes. Enable controller
16529 / 32145     3    N              :latch at Port 3.
16530 / 32146   201    RET            :Back to BASIC.
16531 / 32147   211    OUT (N),A      :Start ADC conversion with
16532 / 32148    19    N              :pulse to Port 19.
16533 / 32149    33    LD HL,NN       :at location 16575/32191.
16534 / 32150   191    Lo N
16535 /          64    Hi N
      / 32151   125    Hi N
16536 / 32152    22    LD D,N         :Number of entries.
16537 / 32153    63    N
16538 / 32154   126    LD A,(HL)      :Get first entry.
16539 / 32155    35    INC HL
16540 / 32156    78    LD C(HL)       :Get next entry.
16541 / 32157   119    LD (HL),A      :Bump entry back.
16542 / 32158   121    LD A,C         :Hold next entry.
16543 / 32159    21    DEC D          :Downcount entries.
16544 / 32160    32    JR NZ,d        :All moved back one?
16545 / 32161   249    d              :No. Go to 16539/32155.
16546 / 32162   219    IN A,(N)       :Yes. Read the ADC
16547 / 32163    19    N              :at Port 19.
16548 / 32164    33    LD HL,NN       :Point to first entry
16549 / 32165   191    Lo N           :at location 16575/32191.
16550 /          64    Hi N
      / 32166   125    Hi N
16551 / 32167   119    LD (HL),A      :Store new entry there.
16552 / 32168    33    LD HL,NN       :Point to HILIM at
16553 / 32169   189    Lo N           :location 16573/32189.
16554 /          64    Hi N
      32170      125    Hi N
16555 / 32171    86    LD D,(HL)      :Get HILIM.
16556 / 32172    35    INC HL         :Get LOLIM at
16557 / 32173    94    LD E,(HL)      :16574/26775.
16558 / 32174    35    INC HL         :Get NEWVAL at
16559 / 32175   126    LD A,(HL)      :first Table entry.
16560 / 32176   187    CP E
16561 / 32177    56    JR C,d         :Is NEWVAL < LOLIM?
16562 / 32178     4    d              :Yes. Jump to 16567/32183.
16563 / 32179   186    CP D           :No.
16564 / 32180    48    JR NC,d        :Then is NEWVAL > HILIM?
16565 / 32181     4    d              :Yes. Jump to 16570/32186.
16566 / 32182   201    RET            :No. Then back to BASIC.
16567 / 32183   219    IN A,(N)       :Turn on controller at
```



```
16568 / 32184     3    N              :Port 3 and
16569 / 32185   201    RET            :go back to BASIC.
16570 / 32186   211    OUT (N),A      :Turn off controller at
16571 / 32187     3    N              :Port 3 and
16572 / 32188   201    RET            :go back to BASIC.
16573 / 32189   . . .  HILIM
16574 / 32190   . . .  LOLIM
16575 / 32191   . . .  NEWVAL
```

STEP 3 In the machine language program we have used two additional device select pulses, OUT 3* and IN 3*, to clear and preset the 74LS74 latch controller. Note that when the latch is preset, at memory locations 16528/32144 and 16567/32183, an LED probe at pin 9 should light up, and when it is cleared, at memory locations 16514/32130 and 16570/32186, the LED should turn off. This LED can therefore be used to check whether your solid state relay is being turned on and off by the program.

STEP 4 Sixty-three temperature values are collected and stored in the file of data starting at memory location 16575/32191. The BASIC program then picks up values from the data file, lines 150 and 160, and displays the data on the video screen. When the microcomputer is switched on the data file locations in W/R memory contain random data, therefore the program clears the file locations using the small routine from memory locations 16517/32133 to 16530/32146 before PLOTting the data. This is the reason why a solid black line appears on your video screen at the commencement of your program. This is an important feature because when collecting data you want to make sure that no values in the file have been inserted randomly by the system.

STEP 5 To start this experiment, check your circuit connections and program, and turn on the ac supply to the bulb. RUN the program. The bulb, if it was on should go off, as confirmed by the LED. The screen now asks you for a high temperature set limit so insert a value of about 140 decimal. This is stored as variable H. The program then asks for an input for the low temperature limit which is stored as variable W. Choose a value about 110 decimal. These values will be determined by the range to which you adjusted the analog-to-digital converter to respond. If you have any difficulties at this point refer to Experiment 6.5 procedure on how to adjust the inputs to the converter.

STEP 6 After the low temperature set limit has been inserted, the program should continue and plot the temperature values as they are input to the converter. The time between readings is governed by the PAUSE time delay at line 200. To make the PAUSE time longer increase N up to a maximum of 32767.

STEP 7 Vary the low and high set limits to visualize how simple a procedure it is using a microcomputer to control an industrial process.

SUMMARY This experiment has been an example of automatic control where the microcomputer senses temperature and relays instructions back to a heating system to maintain the temperature between set limits. The experiment also demonstrated how versatile a microcomputer system could be in altering set limits in any industrial process.

□ 7 □

control signals

We have seen that the signals of the microcomputer consist of the data bus, the address bus, and the control bus. Figure 3.2 showed the three different interface connectors of the Timex/Sinclair computer models. At that time, our interest was in those signals necessary for Input/Output interfacing; namely, the data bus, the low address bus, and those control signals used to create IN* and OUT*. We observed that these signals were all available at the same positions relative to the keyway slots on all three connectors. There are additional control bus lines on these connectors whose description we have deferred until now. Some of these are Z80 microprocessor control lines while some are unique to the particular model of computer, that is the particular model's hardware and operating system. All but two of the control signals on the B&W models (TS1500, TS1000, and ZX-81) are Z80 microprocessor control lines. The additional connections on the Spectrum and TS2000 models are all operating system control lines which we shall not consider. The two operating system control lines on the B&W interface connector are ROMCS* (23B) and RAMCS* (2A). These are memory enable signals: Read Only Memory Chip Select and Write and Read Memory Chip Select, respectively. They can be used to disable the on-board memory ICs by forcing their logic levels into the logic 1 state, thus permitting external memory to be addressed. The two main reasons for needing this capability are: first, managing memory blocks that have been relatively decoded when additional memory is added, and second, bank switching memory blocks sharing the same nominal addresses. Further description of this technique is left to more advanced texts.

The Z80 microprocessor is a 40-pin IC. Eight pins for the data bus, 16 pins for the address bus, and two pins for +5-V and 0-V power inputs leave 14 pins for control signals. All of the Z80 microprocessor control lines are available on the interface connector of the B&W models and include the following signals:

< IORQ* (15A)	< RFSH* (23A)
< MREQ* (14A)	> BUSRQ* (20A)
< RD* (16A)	< BUSAK* (18A)
< WR* (17A)	> WAIT* (19A)
< M1* (22A)	< HALT* (13A)
> PHI (6B)	> NMI* (12A)
> RESET* (21A)	> INT* (11A)

You should recognize those in the left column as control signals we have previously described. The first four are used to generate the four unique control pulses for device and memory selects: IN*, OUT*, MEMR*, and MEMW*. The latter three are related to the timing operations of the microprocessor. PHI is the externally generated clock signal which controls each operation—it is the heartbeat of the computer. In addition to directly driving the microprocessor, PHI is also available on the interface connector to allow other devices to be synchronized with the Z80. M1 is the signal that marks the first machine cycle of each instruction; that is, the fetch operation when the microprocessor is performing the memory read operation to load its instruction register with the next program instruction from memory. The RESET line is the external control signal, generated by a pushbutton or on power up, that forces the microprocessor to reset its program counter to 0 and restarts the entire system. Except for RESET*, all of these signals were illustrated in the output timing diagram in Chapter 4, Figure 4.5.

The control signals shown in the right column have only been mentioned incidentally, if at all, up to this point. The dynamic memory refresh signal, RFSH*, is also referenced in Figure 4.5, although it is not shown directly. This special control line allows dynamic memory ICs to be addressed during the third and fourth clock cycles of the M1 machine cycle. It is illustrated in the Address Bus line of Figure 4.5 by the caption R. By placing the address held in its Refresh (R) register on the address bus and triggering the RFSH control pulse, the Z80 allows dynamic memory to be refreshed. Dynamic write-and-read memory must be addressed in this manner about once a millisecond or its contents will be lost. We shall see below that this consideration must be kept in mind when interfacing a computer that uses dynamic memory.

With the exception of the RESET and PHI lines, all of the control lines we have encountered have been lines whose signals are output from the microprocessor. This is also the case for the HALT* and BUSAK* lines. However, the remaining four control lines are inputs to the microprocessor that are generated externally. The outputs were marked in the list with the bra symbol—<("less than" sign) and the inputs are shown with the ket symbol—> ("greater than" sign). Because the output signals are generated outside of the microprocessor, it is possible for them to occur at any time; that is, they will be asynchronous unless they are somehow controlled by the PHI line or some other clocking control line. There are many cases in interfacing where control signals come in pairs. One signal "asks" and the other "answers." Typically, the signal that asks is called a *request* and functions as a stimulus; the signal that answers is called an *acknowledge* and functions as a response. Whether you refer to these control lines as ask/answer, request/acknowledge, or stimulus/response, the concept is the same. The four signals: INT*, NMI*, WAIT*, and BUSRQ* are all request inputs to the microprocessor.

BUSRQ* and BUSAK* are the most obvious pair of request/acknowledge control signals in our list. Although they would be very difficult to implement successfully on a personal computer because of the complexity of the operating system, they are of interest because they illustrate one of the more advanced applications in interfacing known as Direct Memory Access (DMA). These signals are necessary when two microprocessors share common memory registers (addresses) in a master-slave

relation. It should be obvious by now that only one microprocessor can use the address bus, data bus, and control bus lines at any given moment. This situation arises when large blocks of memory data are to be transferred between computers as fast as possible and the usual serial or parallel methods are not fast enough. The master microprocessor, which controls the transfer, places a logic 0 on the BUSRQ* line of the slave microprocessor which ordinarily controls the memory. When the slave microprocessor is ready to relinquish its buses, it puts all its address, data, and (relevant) control lines into a high impedance state (three stated) and then triggers its BUSAK* line. The master microprocessor interprets this response as a signal to commence using the shared bus lines. When it has completed its transactions, the master microprocessor relinquishes the shared buses and removes the logic 0 from the BUSRQ* line. The return of the BUSRQ* to a logic 1 state signals the slave microprocessor to resume operation.

The WAIT* input is another control line that is used in memory management. Although it is a request input to the microprocessor, it is a response to the MEMR* (memory request) that is usually implemented by the memory address decoding logic of the computer. Its function is to effectively slow the microprocessor down enough to give the memory ICs sufficient time to read or write their contents onto the data bus. Many memory ICs are manufactured whose response times are slower than the microprocessor's operating speed. For example, a Z80 operating at 4 MHz allows only 250 nanoseconds for a memory IC to respond to its memory select pulse (the decoded memory address signal and read or write control pulse). If the response time of the memory is longer than 250 nanoseconds, the microprocessor must be requested to wait. When the WAIT* line is brought to a logic 0, the microprocessor's timing is altered by adding clock cycles between the second and third clock cycle of the current machine cycle. When the WAIT* is brought back to a logic 1, then the microprocessor resumes with the third clock cycle. The one instance that extra WAIT states are of particular interest to the interfacer is when IN and OUT machine language instructions are executed. In these instances, the Z80 automatically inserts one WAIT state in these instructions to allow for the slower response times of peripheral devices.

Throughout the experiments in Chapters 4, 5, and 6, we have assumed that whenever an input port was ready to provide a data byte that the computer would be ready to take it. This may not always be the case. For example, when the computer has other tasks to perform in addition to servicing a particular input port, it may be busy when the port needs to be serviced. We have already mentioned some of the considerations of synchronizing the actions of a port with those of the computer when the concept of handshaking was discussed. We saw that handshaking signals were individual lines connecting a peripheral device and the computer. Each line can function as a one-bit input port to indicate the status of some aspect of the device. In Chapter 3, we learned that individual bits that indicate the status of an operation are called flags. The Flags (F) register of the microprocessor was described as a set of individual bits which provided the Z80 with information on the arithmetic and logical operations such as the zero or nonzero status of the accumulator (A register), whether

a carry bit was used or not. In Chapter 4 other flags were described in terms of whether a port was ready to provide or accept data, which were called Ready or Busy flags, and which indicated by a logic 1 or 0 the status of ready/not ready, or busy/not busy, or ready/busy. Finally, in Chapter 6, we saw that there is an End of Conversion (EOC) flag assigned to one of the pins of the ADC0804 analog-to-digital converter IC. In this instance, the manufacturer of the IC labeled the pin as "INTR*" which stands for "INTerrupt Request."

The INT* and NMI* are both request control inputs to the Z80 microprocessor. They are interrupt requests which serve to literally interrupt the computer while it is doing something else. They differ from each other in that the microprocessor can be programmed to ignore the INT* signal but it can never ignore the NMI* input. The two machine language commands that control the INT* line are EI (Enable Interrupt) and DI (Disable Interrupt). The terminology for being able to ignore the INT* signal is that it can be masked, that is, prevented from being seen. The NMI* stands for "Non-Maskable Interrupt." The distinction between a request and a flag is fairly subtle. A flag signal is passive. It indicates the status of whatever it is supposed to monitor, but it does not provoke any action on its own. It simply waits to be read by an interrogating device such as the microprocessor. A request signal, however, is active. It triggers the device it communicates with (such as the microprocessor) into some form of action. The acknowledge signal from a device may function either passively or actively. Thus the flag can be used to initiate action by functioning as a request.

Before getting into the details of interrupts, we shall finish our account of the rest of the control signals with the HALT* flag. The HALT* line is brought into a logic 0 state when the machine language instruction HALT is executed. It would seem strange to even have such an instruction if it were not possible to have the computer resume operation. The halt state of the computer is more like an interminable wait state than it is a power off state. The important point is that if the computer receives an interrupt request when it is in the halt state it will resume operation. The HALT* line is an output control line from the microprocessor and can be used as a flag to other peripheral devices to indicate the state of the microprocessor.

There are several kinds of interrupt requests. Suppose you are at home in your comfortable easy chair reading an entertaining book (maybe that's where you are right now). The telephone rings. You finish the sentence you are reading, place your bookmark in your book, and go to answer the phone. As you pick up the telephone receiver, the front door bell rings. Before you go to the front door you first have to ask the person on the phone to hold. Now you can answer the door, then you can get back to the phone, and finally, providing there are no more interruptions, you can return to your first task of reading your book. The computer can be interfaced to operate in the same fashion. When the computer receives an interrupt request it completes the machine language instruction it is currently executing, just as you finished the sentence you were reading when the phone rang. After the last clock cycle of the present instruction cycle, it acknowledges the interrupt. Because there is no specific control line dedicated to an Interrupt Acknowledge signal, the Z80 implements an INTA* by executing an IORQ* pulse during an M1* pulse. Because the first machine cycle is

always a memory read and never an input/output request, the two pulses are unique and can be ORed together to form the INTA* control output. The next thing the microprocessor does is to disable the interrupt request line so that no additional interrupts can be made. The TS Interface circuit shown in Figure 4.6 shows the INTA* control line.

Of course, the computer needs a program to perform every task it does. The main task, comparable to your reading a book, is called the *background program.* The interrupt program, called a *service routine* or foreground program, is a subroutine that must be CALLed when an interrupt request is acknowledged. Just as you wouldn't answer the front door when the telephone rings, the computer must know where (the address in memory) to go. There are three ways that microprocessors can determine how to execute a service subroutine. These are:

1 Multi-level/priority interrupts
2 Single-line/Polled interrupts
3 Vectored interrupts

The Z80 microprocessor can be programmed to implement any of the three types. There are three machine language instructions having the mnemonics, IM0, IM1, and IM2, meaning Interrupt Mode 0, 1, and 2, respectively. When the microprocessor is reset, it is automatically placed in Interrupt Mode 0 and the interrupt is disabled.

A multi-level (or multi-line) interrupt capability simply means that there is more than one interrupt request line to the microprocessor. Because the Z80 has both the INT* and the NMI* lines it is multilevel. (The 8085 and NSC800 have several such lines.) The microprocessor must have some means of deciding priority in case two lines are simultaneously triggered. In the present case, the NMI request gets automatic priority. The microprocessor also knows where to call the NMI service subroutine: it is at decimal address 102.

Considering the INT* input, we find that the INT line can function either as a single line or a vectored interrupt. We shall consider the vectored mode first. A vector is a pointer arrow which gives a direction. As applied here, the vector will be a machine language instruction which tells the microprocessor where to call the service routine. This mode is obtained by executing the IM0 instruction. In Mode 0, the microprocessor expects to receive an instruction on the data bus from the device that initiated the INT* pulse. It expects the instruction at the time it responds with the INTA* signal (i.e., the INTA* pulse can be used actively to clock the instruction byte from the device on to the data bus). Several devices may share (using logically ORed signals) the INT* line. The usual method of vectoring is to "jam" one of the eight restart, RST X, instructions onto the data bus. Under the conditions of a vectored interrupt, the jammed vector is read from the data bus and placed in the microprocessor's instruction register. Because the RST X instruction function as one-byte CALLs to one of the decimal addresses: 0, 8, 16, 24, 32, 40, 48, or 56; there are eight possible interrupt service routines. Note, however, that RST 0 would be equivalent to a pushbutton RESET.

Vectored interrupts can also be performed in Mode 2 using the IM2 instruction. In this case, the vector is the low address in a look-up table of addresses. The high address

of the table is held in the I register of the Z80. The table holds the subroutine call addresses. Because 128 pairs of addresses (high and low address bytes) can be stored in a table having the same high address (I register contents), Mode 2 vectors allow for extensive interrupting capabilities.

The Mode 1 interrupt capability of the Z80 is a true single line interrupt that can be implemented by the IM1 instruction. In this mode, when the INT* line is activated with a logic 0 pulse, the microprocessor automatically performs a RST 56 instruction. Thus without having a vector jammed onto the data bus, the program calls the subroutine at decimal address 56. Mode 1 interrupts are obviously the simplest to implement. They also are the most limited, because only one service routine can be called. This does not mean that the system is limited to only one interrupting device. When it is desirable or necessary for more than one device to be able to interrupt the computer, the request lines can be logically ORed together to drive the INT* line. Because the computer can only call one service routine, that routine must consist of a means of determining which device originally triggered the request. This is done by "polling" each device through an input port and having each device set a flag to indicate that it triggered the request. Once the device is identified the subroutine can cause an additional branch in the program to service the particular device.

It should be noted that because all the branching in interrupt servicing, whether vectored or polled, is by means of subroutine calls, once the device has been serviced, a return instruction is all that's required to have the computer resume its background task. Of course, if the microprocessor's registers are used in the service routine, their contents must be saved and restored (pushed and popped) before returning to the background program. It should also be recalled that when an interrupt request is received by the microprocessor, the INT* line is immediately disabled. Therefore at some point in the interrupt service routine the EI instruction must be executed in order to re-enable the interrupt line. Usually, the EI instruction is given just before the RET instruction when the service routine has been completed. In some cases, however, it may be necessary to have the EI instruction executed as soon as possible in order not to lose an interrupt request from another device. This is an extremely treacherous condition because the computer can become "interrupt bound" and spend its time trying to service interrupts which are interrupting interrupts. One final comment about the EI instruction is that the microprocessor enables the interrupt line after the instruction following the EI instruction. Thus, if that is the RET instruction, the interrupt bound condition can never develop because the computer will always be in the background program when an interrupt request is received.

It may strike you that interrupt interfacing is a complicated affair. If so, you are in agreement with the authors. We have surveyed these topics in order to provide you with some background material on these aspects of interfacing. The techniques are rather advanced and not easily implemented. This is particularly true in reference to the Timex/Sinclair B&W models because they use both the INT* (192 times for each NMI*) and NMI* (50 to 60 times per second) lines to provide the video display. We included the INT* and INTA* lines on the I/O Interface board for advanced projects which go beyond the scope of this book. In our experience, the best method for interfacing interrupts is the simplest. One of the simplest ways, particularly in a

measurement laboratory or household environment, where service requests are relatively slow, is to use an external clock to drive the interrupt once a second. If you are interested in learning more about Z80 interrupt interfacing, you might be interested in the book listed below.

Reference

Field, Paul E.: *Computer Assisted Home Energy Management*, Howard Sams Publishing, Indianapolis, 1982.

□ appendix a □

Z-80 decimal assembler

The Z-80 Decimal Assembler consists of six (6) 3″ × 5″ charts labeled A1 through A6. In their final form, charts A1 and A4 (instruction mnemonics) form an envelope opened at its right end. The other four charts form two slides which insert into the envelope. Charts A2 (decimal code table) and A3 (timing table) are printed on reverse sides of one slide and charts A5 (decimal code table) and A6 (code conversion table) are printed on reverse sides of the second slide.

To assemble the Assembler, you will need a 3″ × 5″ index card, six 3″ × 5″ sheets of clear laminating plastic of the type used to protect documents, rubber cement, scissors, a razor knife, and a metal-edged ruler.

STEP 1 Carefully cut out the four charts by trimming along their outer borders with scissors. If desired, the charts may be reinforced by mounting them on index cards with rubber cement. Doing each chart in succession, remove the backing from a piece of the laminating film and carefully align the right edges of the chart and film. Let the film contact the chart in a smooth right to left motion. Be careful that the chart doesn't jump up to the film due to electrostatic attraction. Rub over the entire surface to ensure complete contact, and trim the excess lamination.

STEP 2 Use the razor knife to remove the "windows" marked "cut out" on charts A1 and A4. Use the ruler to guide the knife. Three or four light scribes work better than one deep cut. For best results, trim each window to leave just a trace of its border. When cutting, place the ruler over the window to prevent tearing. Use scissors to cut out the thumb slots on the right edge of each cover chart.

STEP 3 Remove *in one piece* the three-sided border of each slide card by cutting from the top, left end, and bottom of each. This border is about 5/16 inch wide. Keep the borders with their respective slide charts. Now measure off a similar 1/4 inch three-sided border from the index card and cut it out.

STEP 4 Using rubber cement, mount each slide border to the back side of the corresponding cover card. Finally, clean all excess cement from the backside of each cover and cement the 1/4-inch border between the cover borders. Note that the top edge of one cover joins the bottom edge of the other cover. To complete the job, you may want to bind the three sealed edges with 1/2-inch Scotch brand "Magic" tape. Insert the two slides back-to-back and work in and out to remove any excess rubber cement.

CHARTS A1 AND A2

Z-80 DECIMAL ASSEMBLER
IO ASSOCIATES
1304 Westover Dr., Blacksburg, VA. 24060

BRANCH

RST 0	199
RST 8	207
RST 16	215
RST 24	223
RST 32	231
RST 40	239
RST 48	247
RST 56	255

cut out

NN
NZ,NN
Z,NN
NC,NN
C,NN
PO,NN
PE,NN
P,NN
M,NN

cut out

JP (HL)		233
JP (IX)	221	233
JP (IY)	253	233

Copyright 1983
by P. E. Field

STACK

LD SP,HL		249
LD SP,IX	221	249
LD SP,IY	253	249

EX(SP),HL		227
EX(SP),IX	221	227
EX(SP),IY	253	227
EX DE,HL		235

PUSH AF		245
PUSH BC		197
PUSH DE		213
PUSH HL		229
PUSH IX	221	229
PUSH IY	253	229

POP AF		241
POP BC		193
POP DE		209
POP HL		225
POP IX	221	225
POP IY	253	225

EI		251
DI		243
IN A,(N)		219
OUT (N),A		211
HALT		118
NOP		0

CONTROL

MATH

RLCA	7
RRCA	15
RLA	23
RRA	31

DAA	39
CPL	47
SCF	55
CCF	63

-128 ≤ d ≤ 127
0 ≤ N ≤ 255
M = (HL)
X=(IX+d) p = 221
Y=(IY+d) p = 253

LD A,(BC)	10
LD A,(DE)	26
LD A,(NN)	58
LD (BC),A	2
LD (DE),A	18
LD (NN),A	50

LD HL,(NN)		42
LD IX,(NN)	221	42
LD IY,(NN)	253	42
LD (NN),HL		34
LD (NN),IX	221	34
LD (NN),IY	253	34

TRANSFER

ADD A,
ADC A,
SUB
SBC A,
AND
XOR
OR
CP
INC
DEC

cut out

ADD HL,
ADD IX,
ADD IY,
INC
DEC

cut out

LD ___,NN
LD A,
LD B,
LD C,
LD D,
LD E,
LD H,
LD L,
LD (HL),
LD (IX+d),
LD (IY+d),

cut out

cut out

```
                                Yp134d Xp134d M 134 L 133 H 132 E 131 D 130 C 129 B 128 A 135 N 198
                                Yp142d Xp142d M 142 L 141 H 140 E 139 D 138 C 137 B 136 A 143 N 206
                                Yp150d Xp150d M 150 L 149 H 148 E 147 D 146 C 145 B 144 A 151 N 214
                                Yp158d Xp158d M 158 L 157 H 156 E 155 D 154 C 153 B 152 A 159 N 222
                                Yp166d Xp166d M 166 L 165 H 164 E 163 D 162 C 161 B 160 A 167 N 230
                                Yp174d Xp174d M 174 L 173 H 172 E 171 D 170 C 169 B 168 A 175 N 238
                                Yp182d Xp182d M 182 L 181 H 180 E 179 D 178 C 177 B 176 A 183 N 246
                                Yp190d Xp190d M 190 L 189 H 188 E 187 D 186 C 185 B 184 A 191 N 254
                                Y p52d X p52d M  52 L  44 H  36 E  28 D  20 C  12 B   4 A  60
                                Y p53d X p53d M  53 L  45 H  37 E  29 D  21 C  13 B   5 A  61

RET  CALL  JP  201  205  195
RET  CALL  JP  192  196  194                     SP 57        HL 41        DE 25        BC  9
RET  CALL  JP  200  204  202          IX p41     SP p57                    DE p25       BC p9
RET  CALL  JP  208  212  210  IY p41             SP p57                    DE p25       BC p9
RET  CALL  JP  216  220  218  IY p35     IX p35  SP 51        HL 35        DE 19        BC  3
RET  CALL  JP  224  228  226  IY p43     IX p43  SP 59        HL 43        DE 27        BC 11
RET  CALL  JP  232  236  234
RET  CALL  JP  240  244  242  IY p33     IX p33  SP 49        HL 33        DE 17        BC  1
RET  CALL  JP  248  252  250

                                Yp126d Xp126d M 126 L 125 H 124 E 123 D 122 C 121 B 120 A 127 N  62
                                Y p70d X p70d M  70 L  69 H  68 E  67 D  66 C  65 B  64 A  71 N   6
                                Y p78d X p78d M  78 L  77 H  76 E  75 D  74 C  73 B  72 A  79 N  14
                                Y p86d X p86d M  86 L  85 H  84 E  83 D  82 C  81 B  80 A  87 N  22
                                Y p94d X p94d M  94 L  93 H  92 E  91 D  90 C  89 B  88 A  95 N  30
                                Yp102d Xp102d M 102 L 101 H 100 E  99 D  98 C  97 B  96 A 103 N  38
                                Yp110d Xp110d M 110 L 109 H 108 E 107 D 106 C 105 B 104 A 111 N  46
                                                L 117 H 116 E 115 D 114 C 113 B 112 A 119 N  54
                                                Lpl17dHpl16dEpl15dDpl14dCpl13dBpl12dApl19dNp54d
                                                Lpl17dHpl16dEpl15dDpl14dCpl13dBpl12dApl19dNp54d
```

CHART A3

CLOCK CYCLES/INSTRUCTION

Referenced	r	N	(rp) (HL) (X/Y)	rp	HL	X/Y
LD r,_	4	7	A,7 7 19			
LD _,N	7		10 19			
LD -(NN)-,A,13				20	16	20
LD _,NN				10	10	14
LD SP,_					6	10
EX (SP),_					19	23
A Math	4	7	7 19		6	10
INC/DEC	4		11 23			
Rot/Shf	8:A,4		15 23			
BIT	8		12 20			
SET/RES	8		15 23			
ADD _,rp				15	11	15
ADC/SBC HL,				15		
IN/OUT	12:A,11					
PUSH				11		15
POP				10		14

	Uncond.	Conditional		Absolute	
		Pulse	True	t=4	t=8
Block	16	16	21	CCF	IM_
CALL	10	10	17	CPL	JP (X/Y)
DJNZ		8	13	DAA	NEG
JP	12	7	12	DI	t=9
JR	10	10	10	EI	LD A,I
RET	10	5	11	EX AF,AF'	LD A,R
RETI/N	14			EX DE,HL	LD I,A
RST	11			EXX	LD R,A
				HALT	t=18
				JP (HL)	RLD
				NOP	RRD

BINARY	HEX	BINARY	HEX	BINARY	HEX	BINARY	HEX
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

CHARTS A4 AND A5

DISPLACEMENT (d) RELATIVE TO PC = OPCODE ADDRESS + 2

DJNZ d	16
JR d	24
JR NZ,d	32
JR Z,d	40
JR NC,d	48
JR C,d	56
EX AF,AF'	8
EXX	217

F REGISTER

BIT	FLAG	CONDITION
7	MINUS SIGN	D7 = 1
6	ZERO	A = 0
4	HALF CARRY	AFTER DAA OP
2<	EVEN PARITY OVERFLOW	AFTER LOGIC OP 2's CPL. ARITH
1	CARRY	AFTER ARITH OP

BIT 7, BIT 6, BIT 5, BIT 4, BIT 3, BIT 2, BIT 1, BIT 0, — cut out

SET 7, SET 6, SET 5, SET 4, SET 3, SET 2, SET 1, SET 0, — cut out

CY↔D7-D0↔D7 RLC
D0↔D7-D0↔CY RRC
CY↔D7-D0↔CY RL
CY↔D7-D0↔CY RR
CY↔D7-D0↔0 SLA
D7↔D7-D0↔CY SRA
0↔D7-D0↔CY SRL
— cut out

PREFIX(P): 203

-128 ≤ d ≤ 127
0 ≤ N ≤ 255
M = (HL)
X*__ = 221 P d __
Y*__ = 253 P d __

RES 7, RES 6, RES 5, RES 4, RES 3, RES 2, RES 1, RES 0, — cut out

cut out

BLOCK TRANSFER: (HL) = SOURCE, (DE) = DESTINATION

{ BC = DOWNCOUNT }

LDI	160	CPI	161
LDD	168	CPD	169
LDIR	176	CPIR	177
LDDR	184	CPDR	185

{ B = DOWNCOUNT C = PORT }

INI	162	OUTI	163
IND	170	OUTD	171
INIR	178	OTIR	179
INDR	186	OTDR	187

IN _,(C)
OUT (C),
— cut out

IM0	70	RETN	69	LD I,A	71
IM1	86	RETI	77	LD R,A	79
IM2	94			LD A,I	87
				LD A,R	95

NEG	68
RRD	103
RLD	111

ADC HL,
SBC HL,
LD (NN),
LD __,(NN)
— cut out

```
Y*126X*126M 126L 125H 124E 123D 122C 121B 120A 127Y*  6X*  6M   6L   5H   4E   3D   2C   1B   0A   7
Y*118X*118M 118L 117H 116E 115D 114C 113B 112A 119Y* 14X* 14M  14L  13H  12E  11D  10C   9B   8A  15
Y*110X*110M 110L 109H 108E 107D 106C 105B 104A 111Y* 22X* 22M  22L  21H  20E  19D  18C  17B  16A  23
Y*102X*102M 102L 101H 100E  99D  98C  97B  96A 103Y* 30X* 30M  30L  29H  28E  27D  26C  25B  24A  31
Y* 94X* 94M  94L  93H  92E  91D  90C  89B  88A  95Y* 38X* 38M  38L  37H  36E  35D  34C  33B  32A  39
Y* 86X* 86M  86L  85H  84E  83D  82C  81B  80A  87Y* 46X* 46M  46L  45H  44E  43D  42C  41B  40A  47
Y* 78X* 78M  78L  77H  76E  75D  74C  73B  72A  79Y* 62X* 62M  62L  61H  60E  59D  58C  57B  56A  63
Y* 70X* 70M  70L  69H  68E  67D  66C  65B  64A  71

Y*254X*254M 254L 253H 252E 251D 250C 249B 248A 255Y*190X*190M 190L 189H 188E 187D 186C 185B 184A 191
Y*246X*246M 246L 245H 244E 243D 242C 241B 240A 247Y*182X*182M 182L 181H 180E 179D 178C 177B 176A 183
Y*238X*238M 238L 237H 236E 235D 234C 233B 232A 239Y*174X*174M 174L 173H 172E 171D 170C 169B 168A 175
Y*230X*230M 230L 229H 228E 227D 226C 225B 224A 231Y*166X*166M 166L 165H 164E 163D 162C 161B 160A 167
Y*222X*222M 222L 221H 220E 219D 218C 217B 216A 223Y*158X*158M 158L 157H 156E 155D 154C 153B 152A 159
Y*214X*214M 214L 213H 212E 211D 210C 209B 208A 215Y*150X*150M 150L 149H 148E 147D 146C 145B 144A 151
Y*206X*206M 206L 205H 204E 203D 202C 201B 200A 207Y*142X*142M 142L 141H 140E 139D 138C 137B 136A 143
Y*198X*198M 198L 197H 196E 195D 194C 193B 192A 199Y*134X*134M 134L 133H 132E 131D 130C 129B 128A 135


                                                                    L 104H  96E  88D  80C  72B  64A 120
                                                                    L 105H  97E  89D  81C  73B  65A 121


                                                                    SP122    HL106    DE 90    BC 74
                                                                    SP114    HL 98    DE 82    BC 66
                                                                    SP115    HL 99    DE 83    BC 67
                                                                    SP123    HL107    DE 91    BC 75
```

CHART A6

ASCII CODE CHART

SECOND HEX DIGIT	FIRST HEX DIGIT 0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

HEXADECIMAL - DECIMAL CONVERSIONS

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
00	0	00	0	0000	0	0000	0
01	1	10	16	0100	256	1000	4096
02	2	20	32	0200	512	2000	8192
03	3	30	48	0300	768	3000	12288
04	4	40	64	0400	1024	4000	16384
05	5	50	80	0500	1280	5000	20480
06	6	60	96	0600	1536	6000	24576
07	7	70	112	0700	1792	7000	28672
08	8	80	128	0800	2048	8000	32768
09	9	90	144	0900	2304	9000	36864
0A	10	A0	160	0A00	2560	A000	40960
0B	11	B0	176	0B00	2816	B000	45056
0C	12	C0	192	0C00	3072	C000	49152
0D	13	D0	208	0D00	3328	D000	53248
0E	14	E0	224	0E00	3584	E000	57344
0F	15	F0	240	0F00	3840	F000	61440

□ appendix b □

component list*

	EXPERIMENT NUMBER			
COMPONENT	CHAPTER 2 123456	CHAPTER 4 123456	CHAPTER 5 123456	CHAPTER 6 123456
INTEGRATED CIRCUITS				
74LS00	1-111-	-1-11-	-----1	
74LS02	1-----	-1-11-	---11-	
74LS08	1-----			
74LS20	-1----	-----1		
74LS32	1-----	-11---		1----1
74LS74		--1---		1----1
74LS75			---1--	
74LS90	---1--			
74LS93	---11-			
74121		11----		
74LS125			1-----	
74LS138	-----1			
74LS244			-11---	
74LS373		---12-	----1-	
8255		-----1		
8251			-----1	
556	----1-		1----1	
AD558			-1----	1-----
ADC0804				111111
LM358			-1----	--11--
MM58167			----1-	
CRYSTAL				
32768 Hz			----1-	
RESISTORS				
120 ohm (matched)				--2---
150 ohm			-1----	
330 ohm			--3--2	
470 ohm	111541	111-88	-----1	-----1
1.0 Kohm	--2---		4-----	-1----
3.3 Kohm	24----	---888		
4.8 Kohm			4-----	

COMPONENT	CHAPTER 2 123456	CHAPTER 4 123456	CHAPTER 5 123456	CHAPTER 6 123456
EXPERIMENT NUMBER *continued*				
5.1 Kohm	----2-			1---1-
10 Kohm	----1-		-13---	111111
15 Kohm			-----2	
22 Kohm		11----		
10 Kohm (Potentiometer)				---222
CAPACITORS				
22 pF Polystyrene			----2-	
150 pF Disc				111111
0.01 μF Disc			2----2	
0.10 μF Disc			----12	
0.2 μF Disc			2-----	
0.47 μF Disc				1-1---
1.0 μF Tantalum				---1--
10.0 μF Tantalum				----11
POWER EQUIPMENT				
Transistors/D40K(GE) or MPSU45			-1-4--	
3-Vdc Power supply or batteries			-1----	-1----
Stepper motor power supply			---1--	
Transformer/12-Vac Sec.				-----1
TRANSDUCERS				
LED/MV50	111548	111-48	-----1	-----1
Switch/8 DIP		---111		
Switch/SPDT	--111-			
Joystick/4*100 Kohm			1-----	
Optical limit switch/TRW OPB861(Arrow)			-13---	
DC motor/3-V Perm. magnet (Radio Shack)			-1----	---1--
Stepper motor/e.g., 8AU0705 (Septor)			---1--	
Photoresistor/CdS 3-Megohm dark (Radio Shack)				-1----
Strain Gage/e.g., CEA06125UW120 (Meas. Group)				--2---
Temperature Sensor/AD590 (Analog Devices)				----11
Solid State Relay/e.g., Sigma 226 (Allied)				-----1
Lamp/12 V, 50 watt				-----1
HARDWARE				
Wirewrap IC sockets/16 pin	---122			
Opaque disc, 5-10 cm diam.			-1----	
Transparent disc, 5-10 cm diam.			--3---	
Hacksaw blade				--1---
C Clamp				--1---

* Number listed is the quantity required for the particular experiment indicated by the chapter and experiment number column. Total required for all experiments is the largest number entered in the row for the particular item. Refer to experiment description for any special comments concerning a component.

□ appendix c □

suppliers

GENERAL ELECTRONIC CONPONENTS AND INTEGRATED CIRCUITS

DIGI-KEY CORP.
P.O. Box 677
Thief River Falls, MN 56701
800-346-5144

ADVANCED COMPUTER PRODUCTS
P.O. Box 17329
Irvine, CA 92713-7329
800-854-8230

ALLIED ELECTRONICS
4235 28th Ave.
Marlow Heights, MD 20748

JRD MICRODEVICES
1224 S. Bascom Ave.
San Jose, CA 95128
800-538-5000

JAMECO ELECTRONICS
1355 Shoreway Road
Belmont, CA 94002

ARROW ELECTRONICS
4801 Benson Ave.
Baltimore, MD 21227

SPECIALTY MANUFACTURERS

ANALOG DEVICES
P.O. Box 280
Norwood, MA 02062

SEPTOR
4605 Ripley Dr.
El Paso, TX 79922

MEASUREMENTS GROUP
P.O. Box 27777
Raleigh, NC 27611

****GROUP TECHNOLOGY, LTD.**
P.O. Box 87
Check, VA 24072

**TIMEX/SINCLAIR interface buffer and experiment kits

□ appendix d □
glossary

A Accumulator register.
ACCESS TIME The time required to read a memory location.
ACCUMULATOR A special-purpose register in which the results of arithmetic and logic operations are stored.
ACIA Asynchronous Communications Interface Adapter. See **UART.**
A/D ("A to D") Analog to Digital: the process where an analog signal is converted to its digital representation.
ADC Analog-to-Digital Converter: a device which performs an A/D.
ADDRESS The number representing a specific memory location.
ADDRESS BUS The set of parallel signal lines used by the processor to select the source or destination for data transfer.
ALGORITHM A step-by-step specification for the solution of a problem.
ALPHANUMERIC Relating to the alphabetic, numeric, and symbolic printable characters as distinguished from graphics and control characters.
ALU Arithmetic Logic Unit: a device which performs the fundamental mathematical manipulations of a digital processor.
ANALOG A type of signal capable of assuming any value within its operating range.
AND The logic operation defined by the condition that only if all premises (inputs) are true is the conclusion (output) true.
ARCHITECTURE The composition, function, and relationship of the logic elements of a device.
ASCII ("as-key") American Standard Code for Information Interchange: a seven-bit code for alphanumeric and control characters.
ASSEMBLER A computer program which converts the assembly language of the microprocessor into machine code.
ASSEMBLY LANGUAGE The word-oriented mnemonic form of the microprocessor instruction set.
ASYNCHRONOUS An event or device which is not synchronized with the processor clock.
BAUD RATE Serial transmission rate in bits per second assuming the bit duration is constant.
BCD Binary-Coded Decimal: the four bit binary representation of the decimal numerals 0–9.
BENCHMARK PROGRAM A program used to measure the performance of a computer under well-defined conditions.
BIDIRECTIONAL Signal transmission in either direction in a wire.

BIDIRECTIONAL PRINTING Capable of printing lines either forward or backward.
BINARY COUNTER A device having outputs of assigned binary weight (1, 2, 4, etc.) which provides a sequential count of an input signal transition.
BINARY NUMBER A number whose digits double in value; composed of the numerals 0 and 1.
BIT Contraction of BINARY DIGIT. The positional value of a bit is a power of 2 (i.e., 2**n for n = 0, 1, 2, etc.) counting from right to left.
BREADBOARD Any device used to mount and interconnect electronic components for prototype circuit development.
BOOLEAN LOGIC An algebra named after George Boole using quantities that take values of TRUE and FALSE and consisting of the operations AND, OR, NOT, etc.
BOOTSTRAP A program used for starting the computer which usually sets up the I/O devices and loads the operating system.
BOUNCING The mechanical vibration of switch contacts on closure causing many pulses to be transmitted on the signal line.
BRANCH To select between alternate routes, as in the flow (sequence of instructions) in a program.
BUFFER An intermediate signal conditioner between the transmitter and receiver of a signal.
BUG An error in either a circuit (hardware) or a program (software). See **DEBUG.**
BUS A common signal carrier, typically applied collectively to a set of functional signals such as the Data Bus, Address Bus, or Control Bus.
BYTE A group of eight contiguous bits. A byte can represent 256 different values.
C The abbreviation for the Carry flag.
CALL The instruction mnemonic to transfer the program to a subroutine.
CARRY A status bit in the FLAGS register of the microprocessor which indicates whether a carry (borrow) has been created in an arithmetic operation.
CHANNEL A nonspecific term denoting a data path between devices.
CHIP A small rectangular silicon die cut from a wafer. Integrated circuit packages are commonly called chips.
CHIP SELECT The ENABLE control input line on an integrated circuit.
CLEAR The control input line or the corresponding signal which places the output of a device in the logic 0 state. Compare to **SET** and **RESET.**
CLOCK A square wave generator (oscillator) used as a reference timing source. Also the signal derived from such a device.
CLOCK PULSE A complete signal transition from one logic state to the other and back again, either Positive (0-1-0) or Negative (1-0-1).
CLOCK RATE The frequency of a clock.
CMOS ("sea-moss") Complementary MOS, a fabrication technology for chips having very low power consumption.
CODE A representation of one set of symbols by another set, one set typically being a binary number representation.
COMPILER A program which translates high level language commands (source code) into machine code (object code). The object code is then capable of being executed. Compare to **INTERPRETER.**
COMPLEMENT To change the state of a bit.
CONTROL BUS The set of individual signal lines used by a processor to implement the means and timing of data transfer.
CPU Central Processing Unit, the computer module in charge of fetching, decoding, and executing machine code instructions.

CR Carriage Return, the printer action that brings the print position to the left margin; the corresponding ASCII control character.
CROSSTALK Interference between two signals.
CRT Cathode Ray Tube, the phosphor display tube used in video equipment.
CRYSTAL The quartz crystal whose piezoelectric properties provide very accurate frequency generation for clock timing.
CURRENT LOOP A serial communication technique using the presence or absence of current flow in a twisted pair of wires for digital data.
D The abbreviation for a Data input or output line or signal.
D LATCH A type of flip-flop which functions as a one-bit memory device.
D/A ("D to A") Digital to Analog, the conversion of a digital number into a signal level proportional to its binary value.
DAC Digital-to-Analog Converter, a device which performs a D/A.
DATA BUS The set of parallel lines that carry the information being processed by the microprocessor.
DEBOUNCE To eliminate the signal fluctuations generated in mechanical switching.
DEBUG To seek and eliminate the errors in a circuit or a program.
DEC Decrement, the instruction mnemonic to decrease the contents of a register by one.
DECODE To convert an n bit parallel input to select (activate) one of a maximum of 2**n independent outputs.
DEMULTIPLEX The technique in which a common source can be selected to supply one of many destinations.
DEVICE CODE The eight-bit address of an I/O port or the corresponding decoded clock pulse.
DEVICE SELECT PULSE The clock pulse generated from the device code and an input or output control pulse that is used to activate an I/O device.
DIGITAL Having discrete states. Compare to **ANALOG.**
DIGITAL ANALYZER An instrument used to troubleshoot digital circuits by detecting logic states and timing characteristics.
DIODE An electronic device which allows current to flow in only one direction.
DIP ("dip") Dual In-line Package, an integrated circuit casing characterized by two parallel rows of leads (pins) on 0.1 inch spacing.
DIP SWITCHES A set of switches which can be inserted into a DIP socket.
DISABLE To prevent from functioning. Compare with **ENABLE.**
DISK Any disc-shaped magnetic storage medium.
DISKETTE Any small flexible disk contained in a protective jacket and commonly used with personal computers.
DISPLAY A computer output device used to display information.
DMA Direct Memory Access, a technique for very fast data transfer in which a processor temporarily relinquishes control of its memory to another processor.
DOS ("doss") Disk Operating System, an operating system program which implements a disk system for off-line storage.
DOT MATRIX The technique for printing characters using a rectangular array of dots.
DOUBLE PRECISION Program implemented arithmetic in which numbers are stored using twice as many bits as usual.
DRIVE Any electromechanical device used to access different segments of off-line memory storage, such as tapes or disks.
DRIVER A current amplifier used to increase the power of a signal. Also a program which controls a peripheral device.
DUMP To transfer the contents of memory to an off-line storage device.

DUPLEX A bidirectional serial communications link between two terminals. See **FULL DUPLEX** and **HALF DUPLEX.**
DYNAMIC MEMORY A type of R/W memory IC characterized by high density which requires recurrent addressing to be maintained. See **REFRESH.**
ECHO To send the code of a received character back to the device that transmitted it.
EDITOR A program designed to facilitate entry and modification of text, especially in reference to programming.
EMULATE To simulate in real time.
ENABLE To permit to occur. Also the corresponding circuit input functioning either as a Clocked or Gated control.
ENCODE To output a unique n bit value based on which one of 2**n independent input lines is activated.
EPROM ("E-prom") Erasable Programmable Read Only Memory, nonvolatile static memory stored in an integrated circuit which can be erased and rewritten.
EXECUTE To perform or carry out.
F The FLAGS register of the 80 family processors consisting of independent bits which reflect the status of the most recent arithmetic/logic operation.
FETCH To retrieve, hence to obtain an instruction from memory; the first step in a computer instruction cycle.
FLAG A one-bit status indicator signifying one of two possible states: plus/minus, ready/busy, etc. Used by microprocessors to make branching decisions.
FLIP-FLOP A digital electronic device having one or more inputs plus a clock input and one independent output which reflects the status of the input(s) at the time of the last clock input signal.
FLOPPY DISK See **DISKETTE.**
FLOWCHART A diagrammatic representation of a program.
FORTH An intermediate level programming language characterized by the use of a parameter stack and a return stack.
FULL DUPLEX A serial communications link between two terminals in which both can simultaneously receive and transmit information.
F/V Frequency-to-Voltage, the conversion of an analog signal to a voltage proportional to its frequency.
GATE A digital electronic circuit or corresponding input signal which controls the flow of information between the input and output of a device.
GND Ground, the zero voltage potential to which all other voltage levels in a circuit are referenced; not necessarily earth potential.
H A suffix used with numbers to signify hexadecimal base.
HALF DUPLEX A serial communications link between two terminals in which either can transmit but not simultaneously.
HALT The state of a computer in which program execution is suspended. Recovery is possible either by Reset or Interrupt.
HANDSHAKING A synchronizing technique for data transfer between two devices using request and acknowledge control signals.
HARDWARE The circuitry and physical components of a device.
HEX Shortened form of hexadecimal, the base 16 representation of four-bit numbers using the numerals 0–9 and the letters A–F for the decimal values of 0–15.
HIGH BYTE The more significant byte of a 16-bit number corresponding to bit positions 8–15.
HIGH LEVEL LANGUAGE Any programming language having easily invoked commands which perform complex tasks, such as ALGOL, APL, BASIC, C, COBOL, FORTRAN, PASCAL, and PL/1.

IC Integrated Circuit, an electronic circuit etched on a silicon chip. Also the package containing the chip.

IEEE ("I triple E") Institute of Electrical and Electronic Engineers, a professional society active in establishing standards of signal assignment and tolerance.

IMMEDIATE ADDRESS A mode for accessing memory in which the memory location is explicitly specified.

INC Increment, the instruction mnemonic to increase the contents of a register by one.

INDEXED ADDRESS A mode for accessing memory in which the memory location is specified by a displacement from a base (index) address.

INDIRECT ADDRESS A mode of accessing memory in which the memory location is specified as the contents of a register pair.

INDEX REGISTER A 16-bit register whose contents can be added to a displacement specified by an instruction to form an address. The Z80 has two index registers denoted IX and IY.

INITIALIZE To specify the conditions and start-up values of all relevant parameters at the beginning of a process or program.

I/O Input/Output, the signals, devices, or programs associated with connecting a computer system to its surroundings.

INSTRUCTION The simplest single task a microprocessor can perform as represented by a one-byte operation code. The basic classes of instructions include mathematical, transfer, and test operations.

INSTRUCTION CYCLE The sequence of events performed by a computer in carrying out an instruction. The cycle consists of a Fetch (memory read) step followed by Decode and Execute steps.

INT Interrupt, a control input to the microprocessor used by some peripheral devices to asynchronously request service.

INTERFACE The hardware and software required to connect a computer to another device.

INTERPRETER A program which translates high level language commands into machine code as it executes them. Compare to **COMPILER.**

INTERVAL TIMER An electronic device which outputs a signal at specified time intervals.

IR Instruction Register, the eight-bit register in the processor which stores the instruction after it is fetched from memory.

J-K FLIP-FLOP A type of flip-flop which has two control inputs which are interdependent.

JP Jump, the instruction mnemonic to branch either conditionally or unconditionally by changing the value of the Program Counter register.

KEYBOARD The group of switches encoded as alphanumeric symbols and used as the primary input port in microcomputers.

LATCH See D LATCH.

LCD Liquid Crystal Display.

LD Load, the instruction mnemonic for the transfer of data to a destination register from a source register.

LED Light Emitting Diode, a diode which emits colored light when current flows through it.

LF Line Feed, the printer action that advances the print position vertically without changing its horizontal position; the corresponding ASCII control character.

LOOP A sequence of instructions which are repeated more than once before the linear flow of the program is resumed.

LOW BYTE The less significant byte of a 16-bit number corresponding to bit positions 0–7.

LSB Least Significant Bit, the binary digit having a weight (positional value) of 2**0 or 1.

LSI Large Scale Integration, used in referring to integrated circuits having between 500 and 5000 transitors.

M Any memory register whose address is held in the HL register pair of an 80 family processor, also denoted by "(HL)." The abbreviation for the Sign flag (Minus).

MACHINE LANGUAGE The set of binary codes that form the instruction set of a microprocessor.

MAINFRAME A very large computer supporting many terminals.

MASK To obscure. Also a binary code used as a pattern to selectively set or clear individual bits in a binary number.

MASKABLE INTERRUPT Interrupt request control input line which must be enabled via an instruction and may be disabled with an instruction.

MASS STORAGE Off-line storage media characterized by very large capacity and relatively slow access times and typified by tapes and disks.

MEMORY The digital devices that store binary information in registers.

MEMORY MAP A table showing the allocation of regions of system memory for various programming functions in terms of the limiting addresses.

MEMORY MAPPED I/O An addressing technique in which I/O devices are accessed as memory registers.

MICROCOMPUTER A computer system consisting of a microprocessor, memory, supporting digital logic circuitry, and I/O interfaces.

MICROPROCESSOR An LSI circuit which functions as a CPU.

MINUS FLAG The sign bit in the Flags register of the processor used to indicate (by a logic 1) a negative value (MSB = 1) resulting from an arithmetic operation.

MNEMONIC A memory aid, the shorthand notation of a word describing the action of a machine code instruction. Examples include **LD, JP, INC,** etc.

MONITOR A program implementing the fundamental set of commands required to operate a computer system.

MOS ("moss") Metal Oxide Semiconductor, a fabrication technology used in producing most LSI and VLSI chips.

MOSFET ("moss-fet") MOS Field Effect Transistor, a type of transistor having a Gate, Source, and Drain rather than a Base, Collector, and Emitter.

MSB Most Significant Bit, the bit in the leftmost position of an n bit number and having the weight (positional value) of 2**(n−1).

MSI Medium Scale Integration, used in referring to integrated circuits having between 50 and 500 transistors.

MULTIPLEX The technique where many sources share a common destination. To select one from many.

MUX Abbreviation for **MULTIPLEX.**

NAND ("nand") The NOT AND logic operation where the result is the negation (complement) of that obtained by the AND operation.

NC Non-carry, the abbreviation for the complement of the Carry flag.

NESTED One routine contained within another routine.

NIBBLE Usually four bits, the lower or upper half of a byte.

NMI Non-Maskable Interrupt, an interrupt request control input line which is permanently enabled and cannot be disabled by software.

NMOS ("N-moss") The negative channel MOS technology introduced after positive channel MOS.

NOISE Random transients or other interference on a signal.

NOP ("no-op") No Operation, the instruction mnemonic that alters no registers.

NOR The NOT OR logic operation in which the results of an OR operation are complemented.

NOT The logic operation of complement in which the state of each bit is changed.
NUMBER CRUNCHING Slang expression for performing arithmetic intensive operations.
NZ Nonzero, the abbreviation for the complement of the Zero flag.
OCTAL The base 8 representation of three bit numbers using the numerals 0–7.
OP AMP Operational Amplifier, an electronic circuit which functions as a very high gain dc amplifier.
OPCODE Operation Code, the byte of machine code that distinguishes the instruction from prefixed or suffixed bytes used as operands.
OPEN COLLECTOR An older circuit technique used to connect outputs together for bussing signals, now replaced by three-state devices.
OPERAND The byte(s) in a machine code instruction following the opcode that provide data or address information for the proper execution of the instruction.
OPTO-COUPLER See **OPTO-ISOLATOR.**
OPTO-ISOLATOR A device which converts current pulses to light flashes and then converts the light back to current so that two systems can remain optically coupled but electrically isolated from each other.
OR The logic operation defined by the condition that only if all premises (inputs) are false is the conclusion (output) false.
OVERFLOW A flag bit used to indicate that an arithmetic result is too large.
OVERVOLTAGE PROTECTION Circuitry to protect a device from undesirable surges in the ac power line voltage.
P Abbreviation for the parity flag bit. Also the abbreviation for the complement of the Minus (sign) flag (i.e., Plus).
PACKED BCD Storage of two four-bit binary-coded decimal digits into one eight-bit register.
PARALLEL The processing, transmission, or storage of two or more bits or signals simultaneously.
PC Program Counter, the 16-bit processor register which holds the address of the next byte to be fetched by the processor. Also an abbreviation for Printed Circuit.
PERIPHERAL Any device connected to a computer that functions as a data source or sink.
PIN COMPATIBLE Describes connectors whose leads (or pins) have identical functions, especially in reference to ICs.
PIO Programmable Input/Output device, an interface IC which multiplexes the data bus to two or more eight-bit ports that can be configured as input or output by commands from the processor.
POINTER An address held in a 16-bit register that defines a unique location in memory.
POLLING A scheduling technique for I/O devices where the computer interrogates each device in turn to determine if servicing is required.
POP The instruction mnemonic to load a register pair with two bytes from the stack.
PORT An input or output device identified by a specific address or device code.
PPI Programmable Peripheral Interface, see **PIO.**
PROGRAM A sequence of instructions or commands which results in the execution of an algorithm or task.
PROGRAMMING LANGUAGE The set of commands, functions, and statements that can be used to write a program. See **HIGH LEVEL LANGUAGE.**
PROM Programmable Read Only Memory, strictly a read only memory that can be programmed by a user only one time; commonly an **EPROM.**
PROM PROGRAMMER An addressing device used to write binary data into a PROM or EPROM; may or may not be a computer peripheral.
PROPAGATION DELAY The time required for an input signal to translate into an output signal.

PULL UP RESISTOR A circuit technique to hold a line at a specific voltage while still limiting the amount of current drawn.
PULSE A change in voltage or current level which lasts for a short period of time.
PULSER A switching device used to transmit debounced (clock) pulses.
PUSH The instruction mnemonic to load the two bytes from a register pair into the next available locations on the stack.
R The dynamic memory Refresh pointer register in the Z80 microprocessor.
RAM Random Access Memory, an addressing method where the contents of any location can be read from or written to independent of any other location. Contrast to Serial Access as with tape. Also the conventional reference to R/W memory.
REAL TIME A simulation of any activity in a time scale commensurate with the time of occurrence of the real process.
REFRESH To restore the memory contents by addressing, a requirement of dynamic R/W memory ICs on a period of about 2 msec.
REGISTER A set of parallel latches having a common clock input and forming an n bit storage location; the storage locations in a microprocessor and memory.
RELATIVE ADDRESSING A method of memory addressing which adds a two's complement displacement to the current PC address to determine the particular location.
RELATIVE DECODING A method of address decoding which does not use one or more of the more significant address bits.
RELOCATABLE CODE A machine code routine which uses only relative addressing and holds no absolute addresses and therefore is independent of the segment it occupies in memory.
RESET To restore conditions to their initial values.
RET Return, the instruction mnemonic to terminate a subroutine.
RISE TIME The time required to complete the low-to-high transition of a pulse usually measured between the 10% and 90% levels of the waveform.
ROM Read Only Memory, nonvolatile static memory ICs programmed during manufacture; cannot be programmed by the user.
ROTATE An operation that shifts the bits of a number one position to the left or right, passing the MSB to the LSB or vice versa.
ROUTINE A self-contained portion of a program forming part of the main program.
RS-232 A serial communications Standard defining the signals of a 25-pin connector and bipolar voltage signal levels.
RUN To execute a program.
R/W Read/Write, the type of volatile random access memory ICs. Also the processes that transfer data from or to memory, respectively.
SCRATCHPAD A block of R/W memory set aside to hold temporary or intermediate data.
SERIAL The processing, transmission, or storage of data in time sequential order.
SET The control input line or the corresponding signal which places the output of a device in the logic 1 state, sometimes referred to as Preset. Compare to **CLEAR** and **RESET.**
SHIFT An operation that shifts the bits of a number one position to the left or right without exchange between the MSB and LSB as in ROTATE.
SIGN BIT The MSB of a binary number. See **TWO'S COMPLEMENT.**
SIMPLEX A one-way serial communications link between a transmitter and receiver.
SP Stack Pointer, the 16-bit register in the processor that holds the address of the next available location on the stack.
SSI Small Scale Integration, used in referring to integrated circuits having fewer than 50 transistors.
STACK The region in R/W memory used by the microprocessor to store the return

addresses of subroutines and data PUSHed from the register pairs. The Stack is a last-in first-out build-down list.

SUBROUTINE A self-contained portion of a program not situated in the main program but accessible from any point in the main program.

THREE-STATE The digital electronic logic that utilizes the ordinary logic states of 0 and 1 and the additional high impedance output state that functions like an unconnected output.

TRANSIENT A spurious indeterminate signal.

TRANSISTOR A solid state electronic device having three terminals (Base, Collector, and Emitter) capable of amplifying current.

TRUTH TABLE A table listing the output values of a circuit as a function of all possible combinations of input values.

TWO'S COMPLEMENT A mathematical method of expressing positive and negative binary numbers in which the negative of a number is formed by complementing the number and adding 1.

UART Universal Asynchronous Receiver Transmitter, an IC used in serial data communications consisting of parallel-to-serial and serial-to-parallel shift registers and supporting control logic signals.

VARIABLE A symbolically named quantity which may assume assigned values.

VECTORED INTERRUPT A mode of interrupt servicing in which the device passes information to the processor specifying the address of its service routine.

V/F Voltage-to-Frequency conversion. See **F/V.**

VOLATILE MEMORY Memory circuits which lose their contents when power is removed.

VLSI Very Large Scale Integration, used in referring to integrated circuits having in excess of 5000 transistors.

WAIT An internal state entered by a processor in the absence of a synchronizing control signal.

WARM Write And Read Memory, the preferred acronym for volatile R/W memory because sensible data must be written prior to being read.

WIREWRAP A circuit construction technique in which connections of leads are made by spiral windings of wire on square posts.

Z The electrical symbol for impedance measured in ohms, the ac counterpart to dc resistance. Also the abbreviation for the Zero flag.

□ index □

N

O

P

R

S

T

U

V

W

Z