PRACTICAL INTERFACING PROJECTS WITH THE COMMODORE COMPUTERS

ROBERT H. LUETZOW

PRACTICAL

INTERFACING PROJECTS

WITH THE

COMMODORE COMPUTERS

This book is dedicated to my family:
Peggy
Laura and Jennifer

Also by the Author from TAD BOOKS, Inc.

No. 1583 *Interfacing Test Circuits with Single-Board Computers*

No. 1983
$24.95

PRACTICAL INTERFACING PROJECTS WITH THE COMMODORE COMPUTERS

ROBERT H. LUETZOW

TAB BOOKS Inc.
Blue Ridge Summit, PA 17214

NOTICES

The VIC-20, PLUS/4, Commodore 64, Commodore 16, Commodore 128, SIMON'S BASIC, and VICMON are trademarks of Commodore Business Machines, Inc.

A special thanks to:

Dennis Klepper-who photographed and developed all of the black and white pictures in this book.

FIRST EDITION
FIRST PRINTING

Library of Congress Cataloging in Publication Data

Luetzow, Robert H., 1944-
Practical interfacing projects with the Commodore computers.

Includes index.
1. Computer interfaces. 2. Commodore computers.
I. Title.
TK7887.5.L84 1985 004.6'165 85-22231
ISBN 0-8306-0983-0
ISBN 0-8306-1983-6 (pbk.)

Contents

Introduction vii

1 Controlling Hobby Projects with the Commodore Computers 1

User Port I/O Operation-The User Port Experimenter's Boards-Interfacing Circuits and Experiments-DiscreteTransistor Interface Circuits-Using Integrated Circuits for I/O Interfacing-Example I/O Circuits Using TIL and CMOS Chips-Program Experiments-Conclusion

2 Computer Control with Machine Language 41

The Machine-Language Monitor-A Short Machine-Language Program-A Time-Delay Subroutine-A Final Note on Checking Switches and Pushbuttons

3 Special Projects 58

Project 3-1-Timing Programs-Project 3-2-Converting Analog Signals into Digital Data-Project 3-3-A Universal Op-Amp Circuit for Temperature Measurements and Other Applications Using the *NO* Converter in Project 3-2-ProJect 3-4-An Analog Waveform Recorder

4 VIC-20 Interface Circuits for 110 Projects 91

The 1K Machine-Language Memory Circuit-Adding an Extra 6522 VIA I/O Chip-Adding an Analog-to-Digital Converter Circuit-An I/O System on a Single Plug-in Card-Conclusion

5 A Digital-to-Analog Converter Circuit for the VIC-20 and C-64 100

***6 1/0* Circuits for the Commodore 64** 104

Interfacing I/O Circuits to the C-64- The 6522 VIA Circuit-The *NO* Circuit

7 An 110 System for the C-16 and PLUSl4 110

The Card Cage-The Address Decoder Board-The I/O Port Board-The Analog-to-Digital Converter Circuit-Conclusion

8 Graph Plotting Routines 128

Low-Resolution Plotting Routines-High-Resolution Plotting-Summary

9 A Practical Waveform Recording Program 133

A Practical Application-Conclusion

10 Elementary Signal Analysis 143

Curve-Fitting Program-Conclusion

11 Power Supplies 154

Commercially Manufactured Power SupplieS-Power Supply Construction Projects

12 Computer-Controlled Electronic Measurements 161

Continuity Checks-An Eight-Circuit Continuity Test Method-Window Comparators-Interfacing to a Bridge Circuit-Ac Bridge Circuits and Ac Sine-Wave Sources--Ac Bridge Circuit Amplifier and Level Detector-Conclusion

13 The 6502 Instruction Set-An Alphabetical Presentation 177

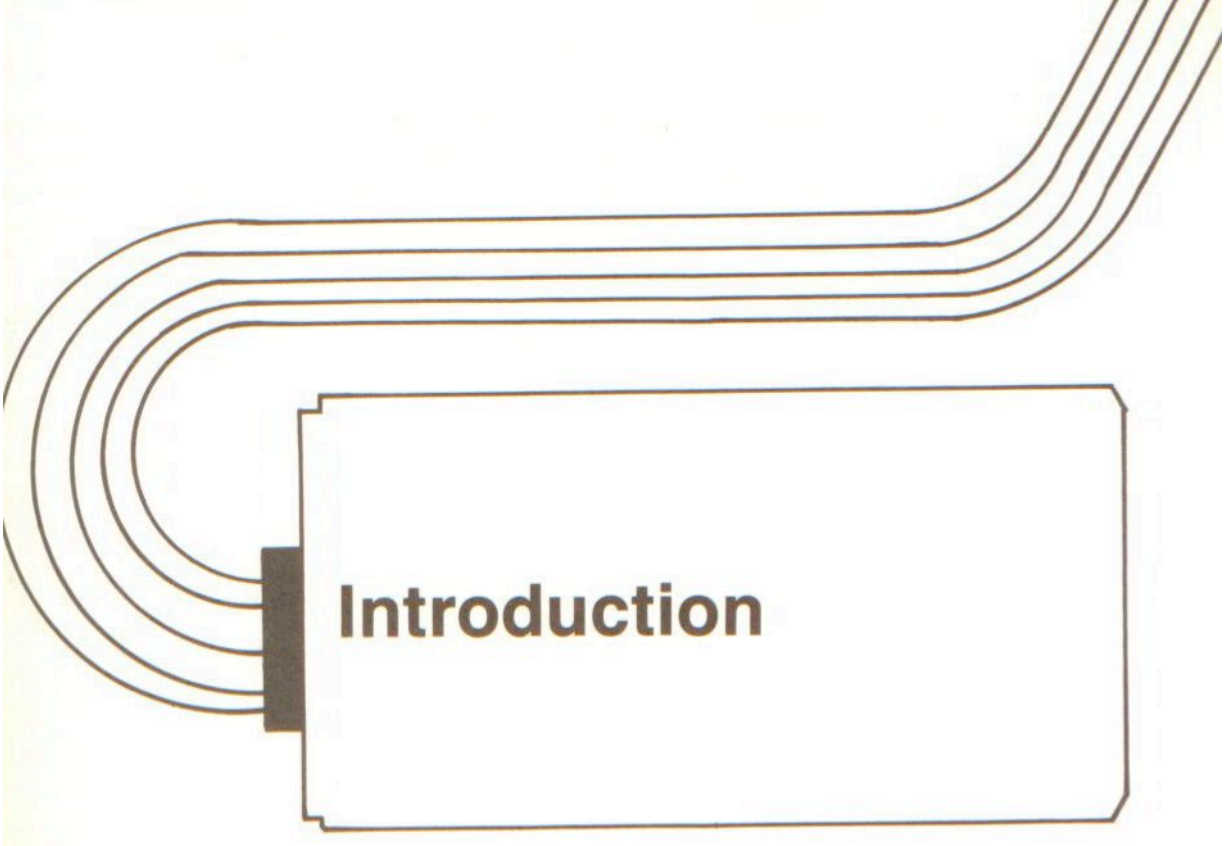
14 Analog Control ProJects 206

Open-Loop Motor-Speed Control-Closed-Loop Motor-Speed Control-Closed-Loop Servo-Control System-Conclusion

15 Useful Hobby and Control Circuits 217

Appendix Circuit Board Layouts for the C-64 and C-128 231

Index 241



Introduction

In the past three years, the Commodore computers have become very popular. There have been quite a few books published about these computers that tell you how to write software programs for home, educational, business, and game applications. In this book, you will be introduced to another use for these computers that has been left almost untouched by all of the other books. This "other use" is *controlling* hobby projects such as toy trains or school science experiments that are performed in physics and chemistry courses. Using the computer control principles that are described in this book, *you* can easily control a hobby project such as a toy train set-up. If you have taken a physics course in high school or college, you most likely performed an experiment in which you placed an ice cube in a container of hot water and charted the temperature decline of the water. Using one of the technical projects in this book, you can let the computer keep track of the temperature of the water and then print out the recorded data in graphical form.

Don't think for one minute that the Commodore computers and the interfacing projects in this book are limited to just hobby projects. The picture on the cover of this book and two others in Chapter 1 show three very sophisticated test systems that are being controlled by Commodore computers using interfacing circuits that are similar to the ones described in this book.

The level of electronic technology that is required for using this book has been kept as simple as possible. The projects have been designed to be safe and practical for both you and your computer. Anyone who has an understanding of fundamental dc electronics and can construct a simple transistor-switch circuit can use this book. If you can not build a simple transistor switch circuit, this book will tell you how. All you need to do is to buy the correct electronic part from a given parts list and follow the project instructions in the book.

Every project in this book has a computer program written for it. Each computer program is written for a specific Commodore computer to keep the programs as simple as possible. If a project is intended to be used for all four computers, there are

four programs presented for that project. All of the programs were printed out while the project and program were up and running together in order to keep the mistakes as few as possible.

If you do every project that is presented in this book, you will be able to impress the best of the computer programmers. The average programmer can easily understand the software concepts of the computer but has no idea what the electrons inside the computer are doing. When you have your computer controlling a hobby project or recording an analog waveform, even the best computer hacker will be impressed whether they admit it or not.

Note: The projects and programs in this book will all work on the new Commodore 128 microcomputer. Put the C-128 in the 40-column mode and use the programs that were written for the Commodore 64.

Chapter 1

Controlling Hobby Projects with the Commodore Computers

THE COMMODORE VIC-20, THE COMMODORE 64, the Commodore 128, and the PLUS/4 personal computers are well known as a group of versatile computers that can be used to run a great variety of personal and business software plus a large number of games. In this chapter, we will introduce you to another use for the computers, which is how to control the operation of hobby projects. Along with controlling things like toy trains, these simple principles of computer control can be used for engineering, science, and educational experiments. The VIC-20, the Commodore 64 (C-64), the Commodore 128 (C-128), and the PLUS/4 computers as purchased from the computer store are equipped to easily function as a small stand-alone control system that can perform highly complicated tasks. The Commodore 16 does not have a USER PORT and requires a little extra hardware help to perform the same 110 functions. The purpose of this this book is to show you how to use your computer, no matter which one you have, as an accurate control system.

As an example of what one can do with a VIC-20 or a C-64 computer, Fig. 1-1 shows a test system controlled by a C-64 that will test all electrical parameters of an ignition sensor used in the automotive industry. Figure 1-2 shows a VIC-20 controlling a complete test system that checks the coil resistance and break-down voltage of an automotive solenoid coil while, at the same time, running all of the test system robotic control functions. Figures 1-3, 1-4, and 1-5 present the hard-copy print-out data from a waveform recording system that is built around a C-64 and described in Chapter 9.

A series of starter projects will be presented in this chapter that use the experimenter's board concept with the computer's USER PORT. These projects and their supporting software programs are designed to teach the basic concept of hardware 110 operation and interfacing. The VIC-20 and the Commodore 64 both have similar USER PORT 110 pinouts on the back of each computer. This USER PORT has an eight bit 110 port available that can function under software control as eight output control lines, eight input data lines, or any combina-

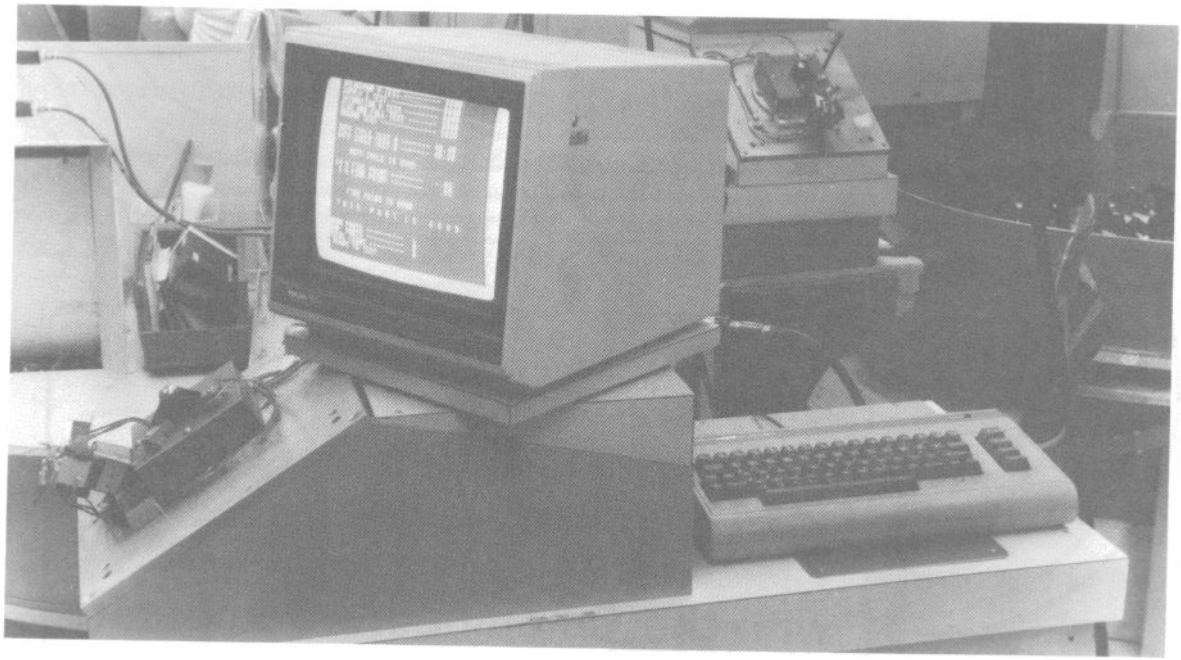


Fig. 1-1. This test system is controlled by a Commodore 64. The test system checks the functional parameters of a Hall-effect Ignition sensor which are: fire-point position, duty cycle, saturation voltage, Vee current, leakage current, and ground path continuity.

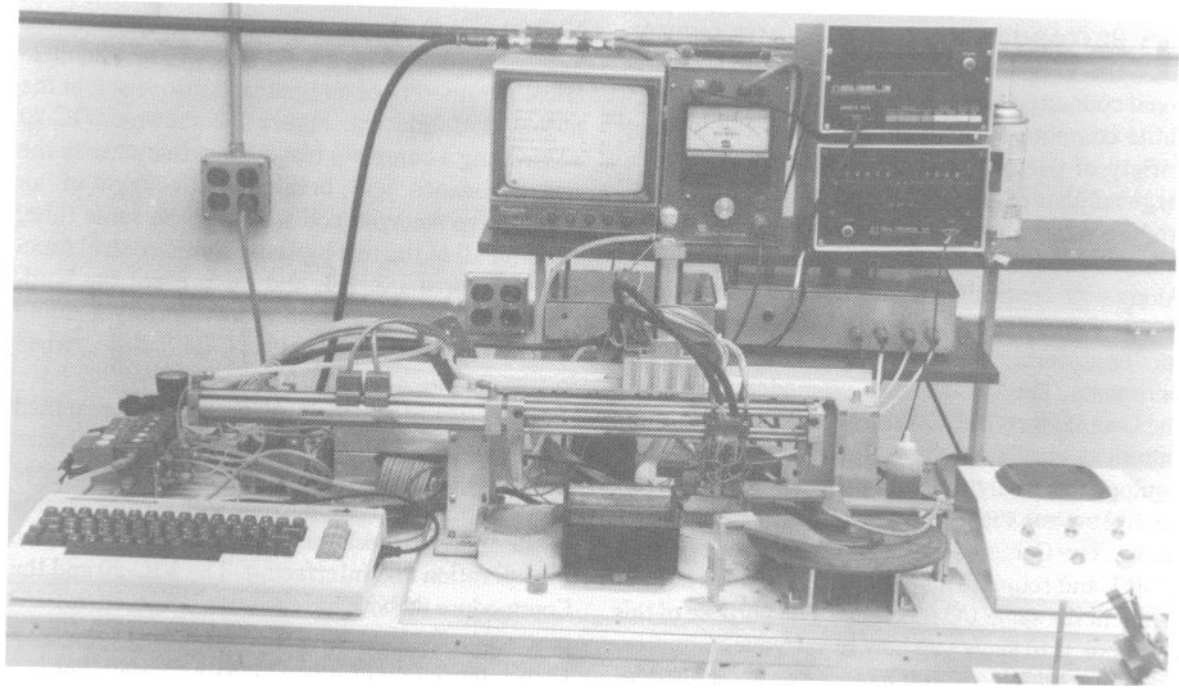


Fig. 1-2. This test system is controlled by a VIC-20 computer. The test system checks an automotive solenoid coil for coil resistance and break-down voltage, while at the same time controlling all of the system's robotic functions.

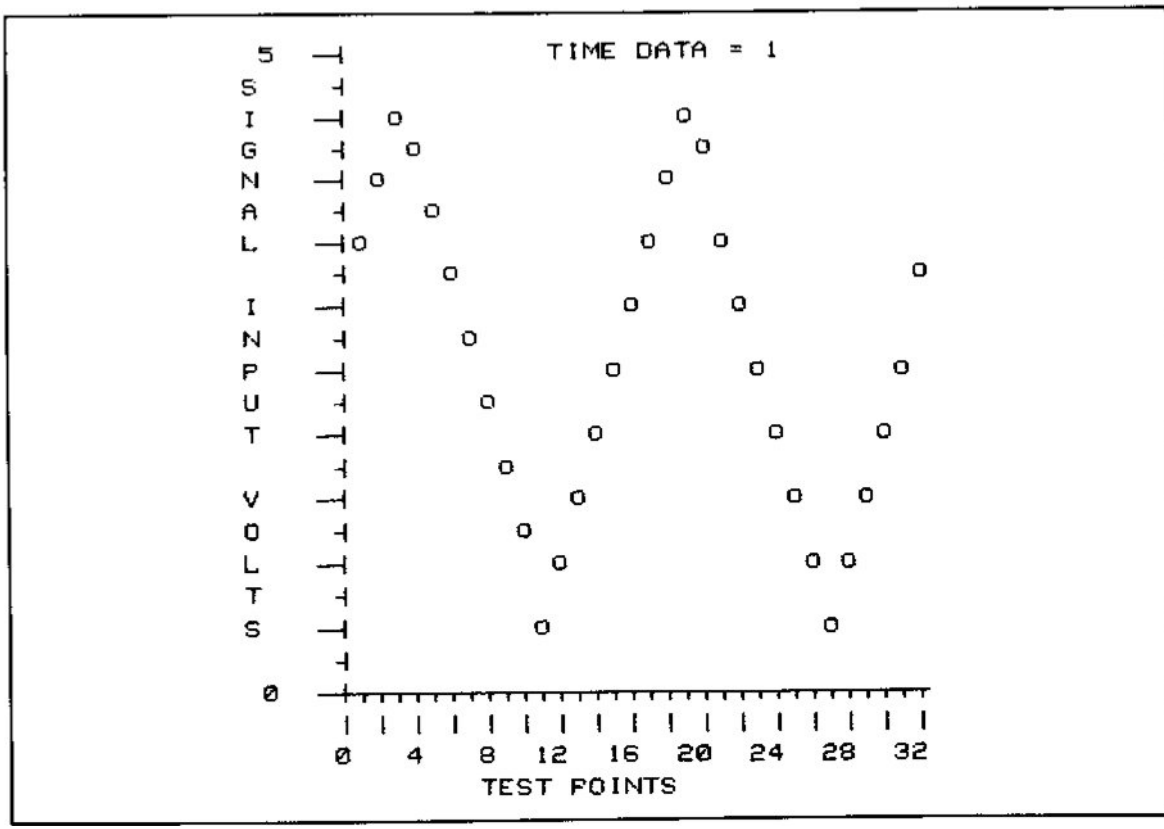


Fig. 1-3. This is a hardcopy printout from a waveform recorder system built around a Commodore 64. This graph displays the 32 recorded waveform points and shows that the recorded waveform is a triangular-wave.

tion of eight input-output lines, Both computers also have a variety of on-board timers which can easily perform timing functions anywhere between microseconds to hours. The PLUS/4 does not have the same USER PORT configuration as the VIC-20 or the C-64, but it can be used in a similar fashion. When one has a computer with a language like BASIC, an 110port, and on-board timers, you have all of the needed requirements for a control system. If the computer's BASIC program language does

```
THE 32 SAMPLE POINTS ARE -

 3.574      4.16       4.57       4.375
 3.867      3.359      2.812      2.304
 1.816      1.25        .683      1.074
 1.64       2.07       2.656      3.203
 3.632      4.199      4.707      4.296
 3.73       3. 183     2.734      2.207
 1.679      1. 171      .703      1.152
 1.718      2.226      2.714      3.261
```

Fig. 1-4. This shows the actual recorded data points that were used to generate the graph in Fig. 1-3.

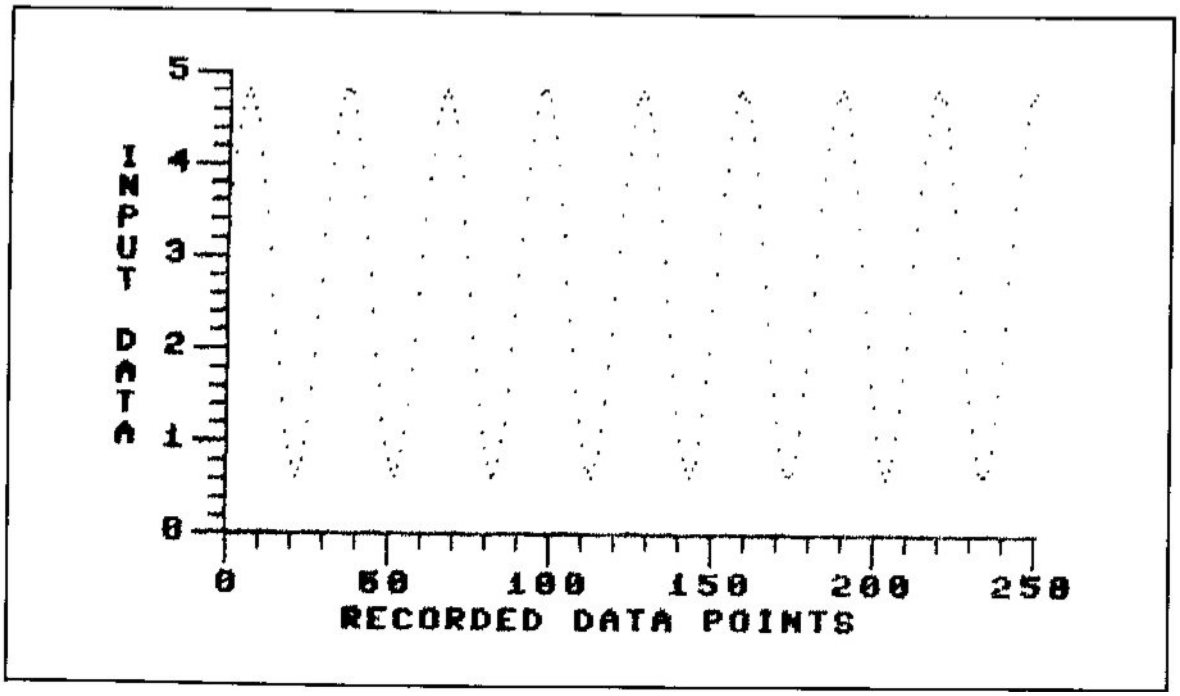


Fig. 1-5. This shows a waveform display that can be secured from the high-resolution waveform recording programs that are presented in Chapter 3, Project 3-4, and in Chapter 9.

not run fast enough for your project's functions, you can use machine language subroutines to perform all of your tasks at lightning speed.

In Chapter 7 of this book, a slide card *110* system for the C-16 (and PLUS/4) will be presented that will increase the I/O capabilities of the C-16 computer to control projects that require TTL compatible I/O lines. But for now, you will be shown how to build an experimenter's board that will operate from the USER PORT of the C-64, the VIC-20, or the PLUS/4.

USER PORT I/O OPERATION

Using the USER PORTS of the VIC-20, the C-64, or the PLUS/4 for 110 experiments is not too hard once you understand how the 110 ports function. Each of the computers have a specific 110 port circuit chips that controls all of the 110 port functions. These I/O circuit chip in the VIC-20 and C-64 have several programmable registers that you must learn how to control in order to use the USER PORTS.

Table 1-1 shows that the VIC-20 and the C-64 have USER PORTS that have a DATA REGISTER (DR) and a DATA DIRECTION REGISTER (DOR). The registers are like any other memory location in that they are addressed as memory locations and have eight bits. The data direction register controls the operation of each of the data register port bits by making them either an input or output bit. The eight bits of the port B data register are connected to the eight circuit board pins that make up the 110 USER PORT. As far as the computer is concerned, an input bit is used to receive data from the outside world and an output bit is used to send data to the outside world. Placing a logic ZERO in a bit location in the DDR makes the corresponding bit in the DR an input bit or input line. Placing a logic ONE in a DDR bit location makes the corresponding DR bit an output bit or as it is better known, an output line. Table 1-1 also presents the decimal numbers that can be poked into the data direction registers with the POKE command to configure the data register bits as an

110port. When the computer is turned on, all data register bits are set up as inputs. If you need an output line, you must reconfigure the DDR by POKING a logic ONE into the DDR bit corresponding to the DR bit which is to be an output line.

The PLUS/4 computer has an *110* chip that does not use a data direction register. Simply putting it, if you PEEK the 110 memory location

Table 1·1. A Short Relerence Outline of the Important Data that Is Needed to Use the USER PORTs to Control the Experimental Programs and Circuits In Chapter 1.

	PEEK and POKE Memory Locations Decimal - - Hex
VIC-20 USER PORT DATA	
1/O Chip - - 6522	
Port B Data Direction Register -	37138 9112
Port B Data Register - - - - - -	37136 9110
COMMODORE 64 USER PORT DATA	
1/0 Chip - - 6526	
Port B Data Direction Register -	56579 D003
Port B Data Register - - - - - -	56577 0001
PLUS/4 USER PORT DATA	
1/0 Chip - - 6529	
Bidrictional Data Port - - - - -	64784 FD10

REGISTER INFORMATION

A Register contains Eight Bits
Most Significant Bit Least Significant Bit

B7	B6	B5	B4	B3	B2	81	80

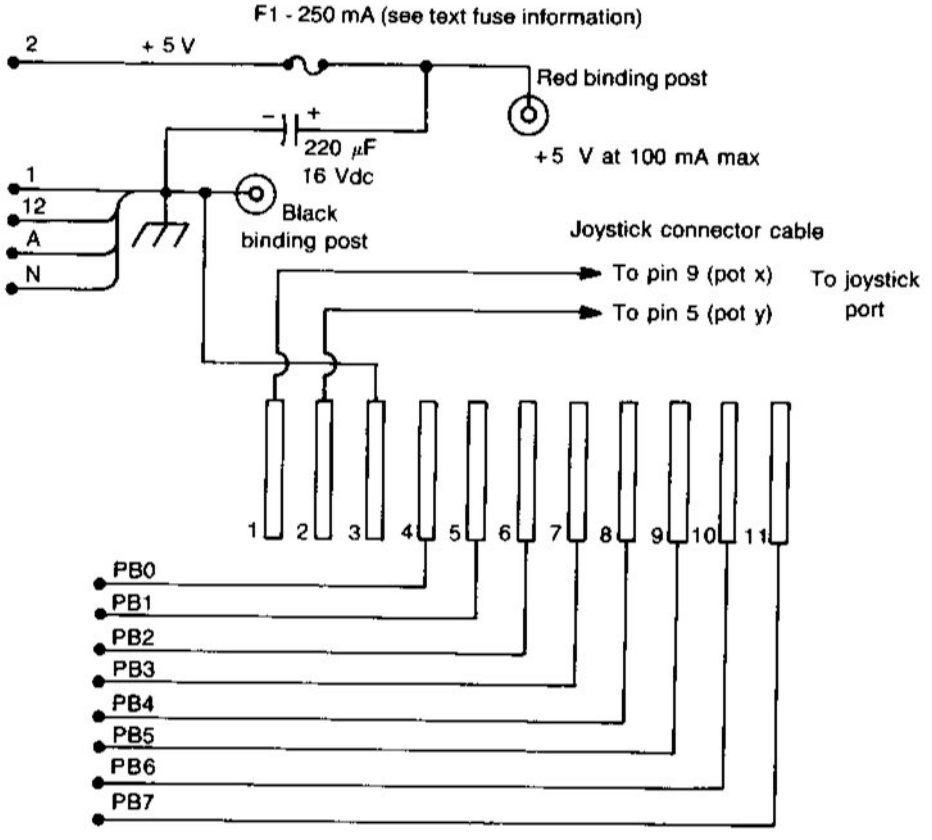
POKE Data For Registar Bits:

Decimal POKE Number	Logic bit Format							
00121 -	0	0	0	0	0	0	0	0
12101 -	0	0	0	0	0	0	0	1
002 -	0	0	0	III	0	0	1	0
004 -	0	0	0	III	III	1	0	0
008 -	0	0	0	111	1	III	0	0
016 -	0	0	0	1	0	0	0	0
032 -	0	0	1	0	121	0	0	0
064 -	0	1	0	0	0	0	0	0
128 -	1	0	0	0	0	0	III	0
255 -	1	1	1	1	1	1	1	1

EXAMPLE: A logic ONE in Bit 4 of a data direction registar makes BIT 4 of the data registar an output bit. A logic ZERO in that location makes that BIT an input BIT.

(64784), the port acts like an input port, and when you POKE data into it, it acts like an output port. A special note should be made at this time. In order to use a port bit as an input bit, you must first POKE that bit location to a logic ONE. Then, when you PEEK the *110* location, that bit will be read as a logic ONE unless a logic ZERO signal from the outside world has pulled that bit (or input line) low,

This USER *1/0* PORT explanation has been kind of short, so you should also read the program-



Note: Use the right side experimenter bOard plug-in holes 'OF" to "J" as shown in pictorials 1.2 and 1.3. See text for connection details. A Radio Shack experimenter 300 board was used for this project. See page 283 in the VIC-20 programmer's reference guide of page 397 in the C-64 guide. For user port pin-out data

Fig. 1-6. This is the connection diagram for the UEB-1.

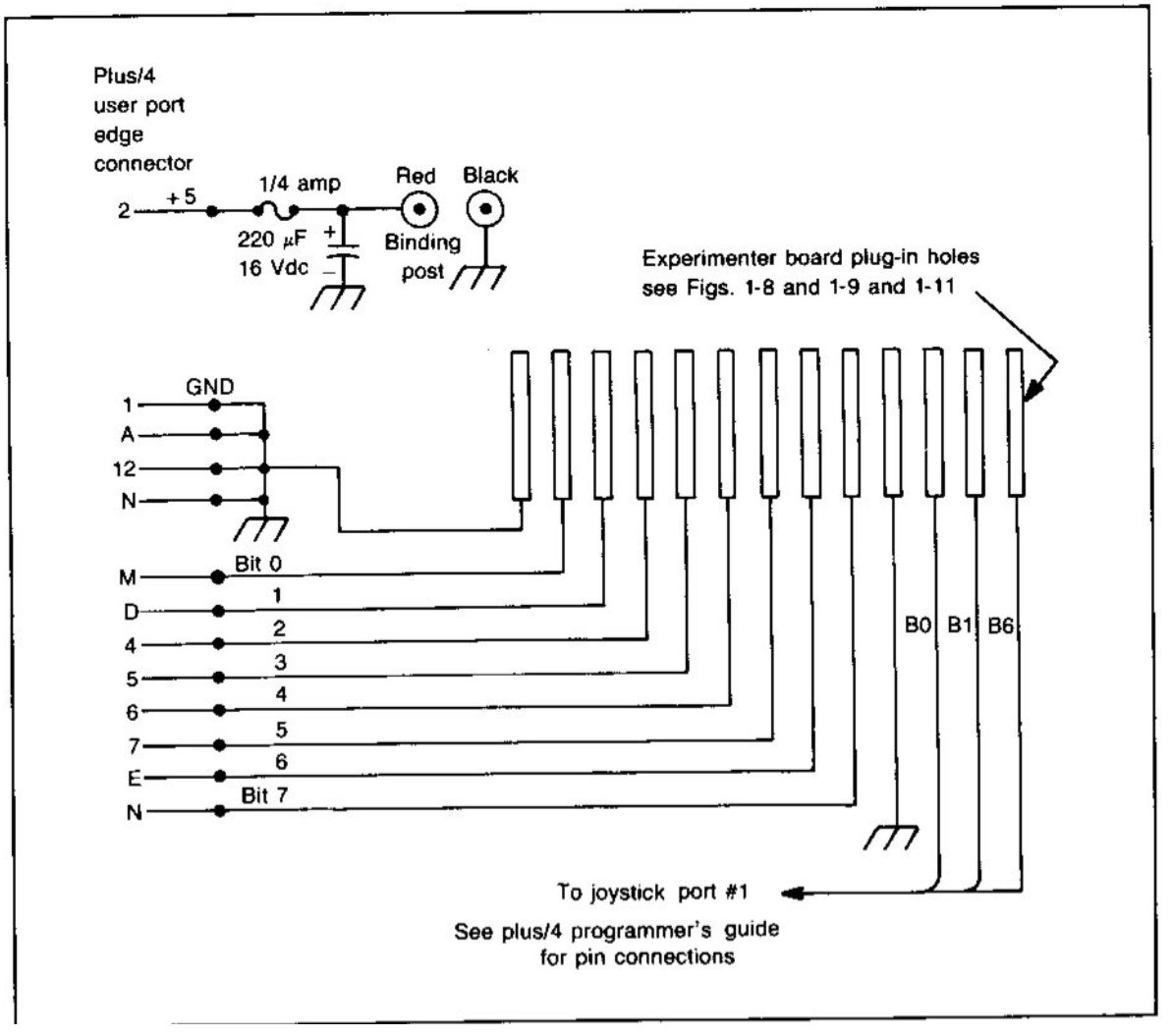


Fig. 1-7. This is the connection diagram for the UEB-2.

mers guide for the computer you are using to secure a complete understanding of the USER PORT *110* functions. A good understanding of the 110 operation will be gained after you have built and run some of the experimenter's projects in this chapter.

A final USER PORT note will now be presented. Some of the experimenter's circuits use the plus 5 volts from the USER PORT pin 3. This plus 5 volts can supply only 100 milliamps of dc current. This is enough current to operate TTL circuit chips and LEDs, but it can not be used to operate dc motors, lights, or bells.

THE USER PORT EXPERIMENTER'S BOARDS

Now that you have a general idea of the operational functions of the USER PORTs, two experimenter's boards will be presented that will make experimenting with different USER PORT 110 circuits very easy. The completed experimenter's board will be called the *Users Experimenter Board* (UEB). The UEB will be given a dash number with the UEB-1 being used on the VIC-20 and the C-64 and the UEB-2 used on the PLUS/4. The C-16 does not have a USER PORT,

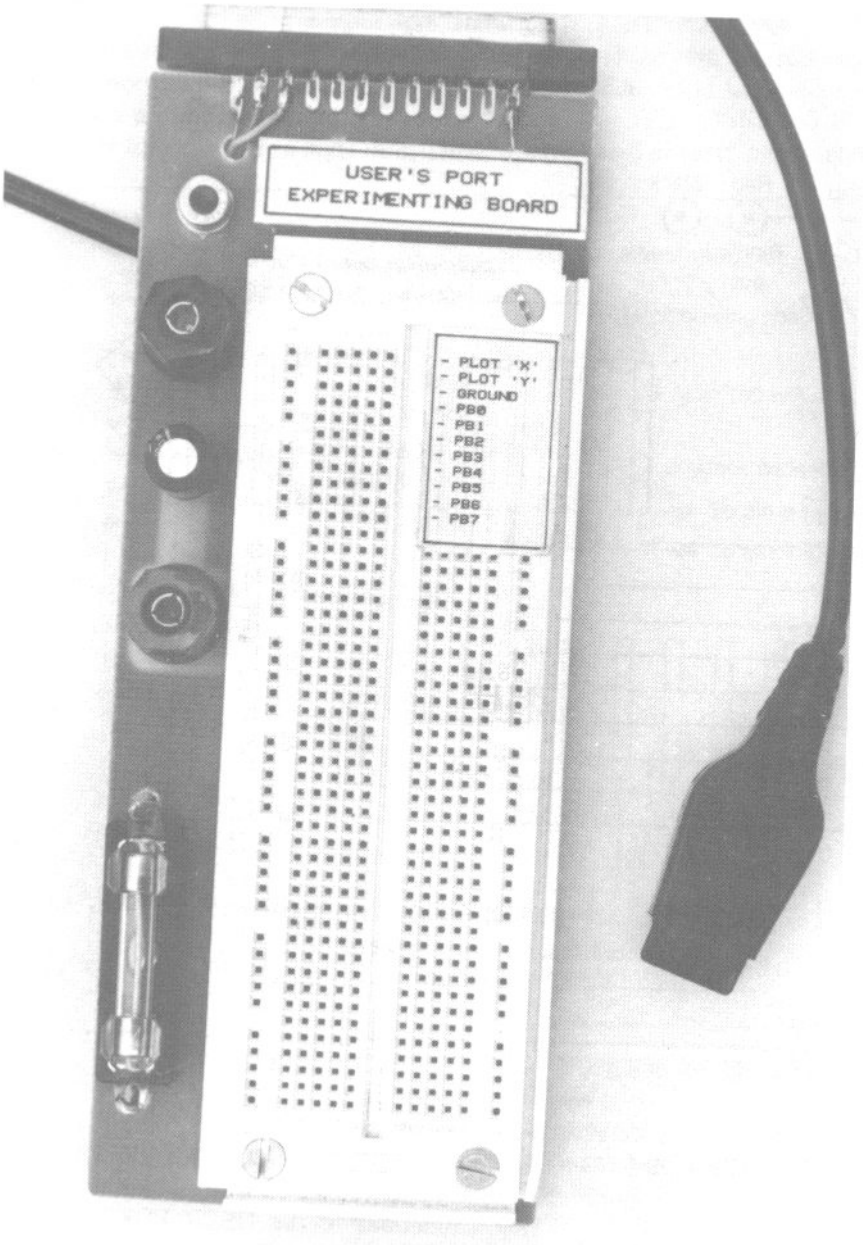


Fig. 1-8. This is a top view of the UEB-1.

but this problem is solved in Chapter 8. This experimenter's board method is not a new idea since a different version was presented in another book for other single-board computers. (1)

(1) Luetzow R. H. *Interfacing Test Circuits with Single-Board Computers,* TAB Books: Blue Ridge Summit, Pa., 1983, p41.

The two UEB units are built on a supporting board made from double sided copper clad circuit board material. The UEB-1 connection diagram is presented in Fig. 1-6 and the UEB-2 is shown in Fig. 1-7. The top and bottom views of the UEB-1 are shown in Figs. 1-8 and 1-9 and a top view of the UEB-2 is shown in Fig. 1-10. If you study these

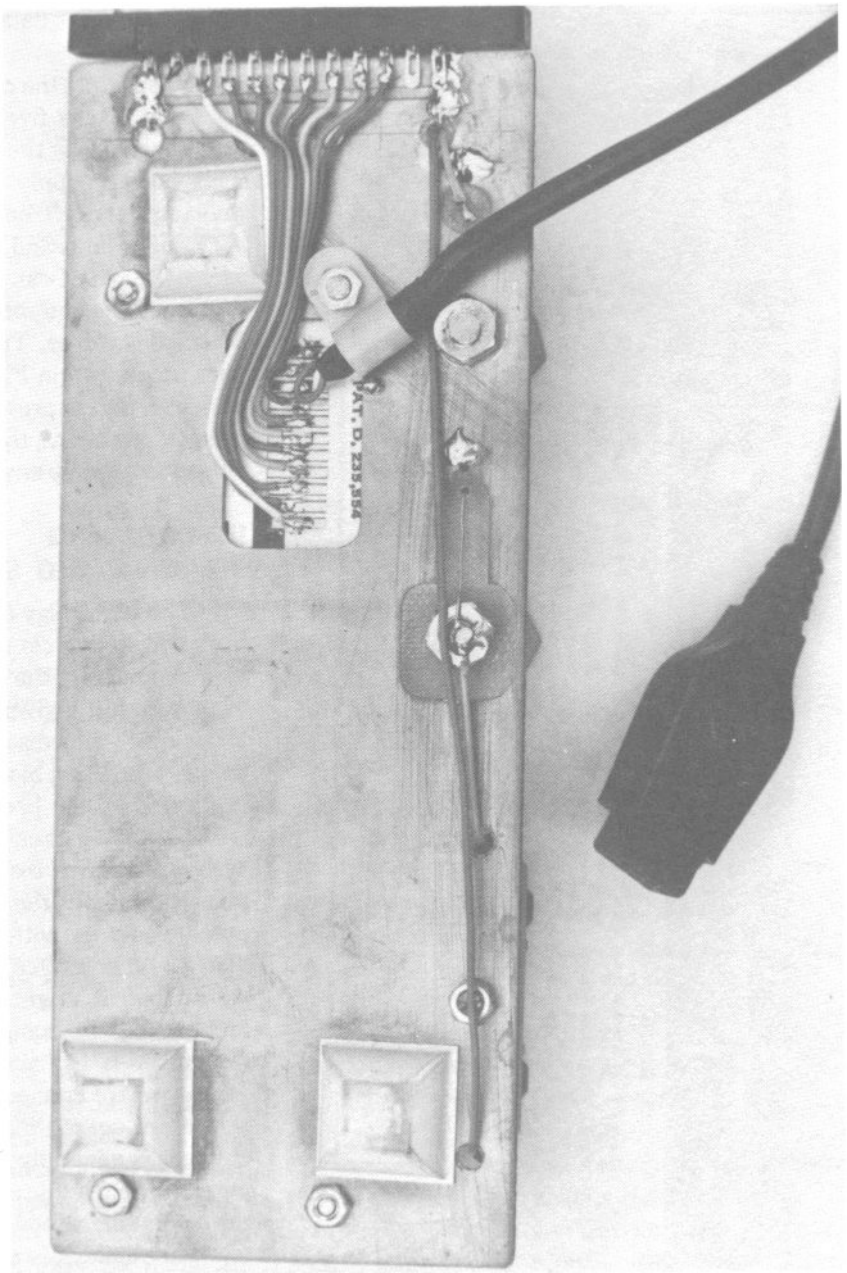


Fig. 1-9. This is a bottom view of the UEB-1. Note how the hard-wire connections are made to the experimenter's circuit board.

figures closely, you will see that a twenty-four pin edge connector was connected to the supporting copper circuit board by soldering the four end pins to the copper foil. Since the four outside pins of the USER PORTs are ground pins, this is an easy way of connecting the connector to the support board while at the same time making the copper foil a ground or common circuit point. A layer of tape is placed between the copper foil and the other connector pins to prevent them from shorting to the copper foil. Looking at Fig. 1-9 you will see that the connections to the experimenter's board plug-

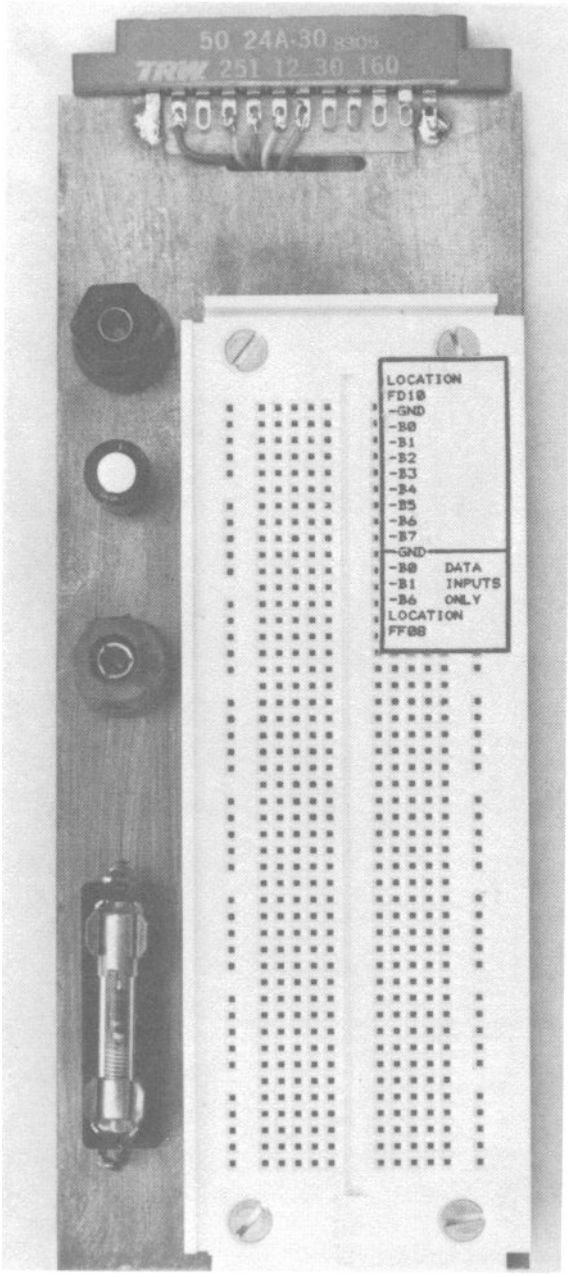


Fig. 1-10. This is a top view of the UEB-2.

in pins are made by cutting a hole in the support board and then removing the backing from the experimenter's board, which lets you make solder connections directly to the circuit plug-in clip pins. A label can be made and placed on top of the UEB to indicate what data is available from a specific pin location.

If you study the connection diagrams, you will note that a plus five volts from the computer is made available at the *red binding* post on the UEB. This power supply point can only supply 100 milliamps of dc. This is enough current to operate TTL logic chips and other circuits elements such as LEDs, but it can not be used to run motors, power relays, and other items that require higher dc current values. Two small battery power supplies are shown in Fig. 1-11 that will run any experiment that is presented in this chapter. Under average conditions, these two power supply circuits will not smoke up anything if you make a mistake.

INTERFACING CIRCUITS AND EXPERIMENTS

The terminolgy of "interfacing circuits" in this book will mean a circuit that is designed to connect your computer to the outside world for the purpose of performing a given technical function. The interfacing circuits discussed in this chapter are fundamental building blocks for the projects presented in the rest of the book.

Interfacing circuits are the parts of the control system that gives the computer a form of physical muscle. Without the interface circuits, The computer would be nothing more than a play thing. When you interface a computer to the outside world, there is a right way, a wrong way, and a hazardous way of doing it. One point you should remember is to "always protect the computer" when you are connecting it to the outside world. **If** you are connecting only to TTL or CMOS IC chips, it is generally pretty hard to hurt the computer. When you are driving power circuits with the computer, a good general rule to follow (unless you don't make mistakes) is to always use a buffer circuit between the computer and the interface circuit. This way, if you smoke an interface circuit, you will only torch the buffer circuit and not the computer 110 port chip. The 110 port chips in the Commodore Computers are constructed in such a way that you must really abuse them in order to burn out a port bit when you are using 5-volt TTL

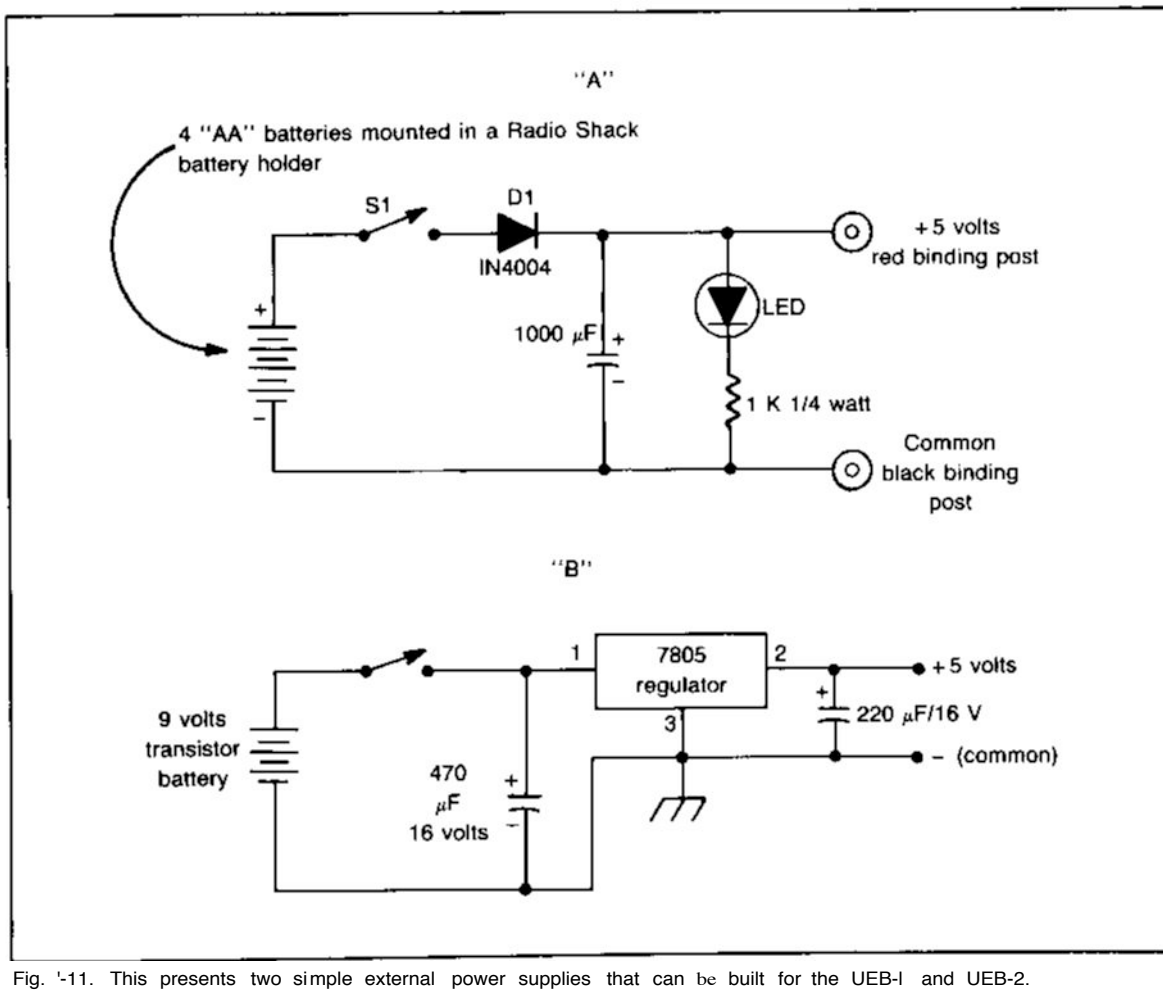


Fig. '-11. This presents two simple external power supplies that can be built for the UEB-I and UEB-2.

or CMOS logic circuits.

Before we start describing the different experimenter's circuits, it will be assumed that the reader has a basic background in electronics and can understand how a npn transistor can be turned off and on and knows a little about the logic families of TTL and CMOS. There are a lot of good electronic books at the locallibrary; so, if you are strong in programming and weak in electronics, you should study some of these books. Learning how to interlace 110circuits to a computer is somewhat comparable to learning how to program a computer because a little hands on experience can be better than two thousand words and a whole bunch of flowcharts.

DISCRETE TRANSISTOR INTERFACE CIRCUITS

The first of a number of different computer interlace circuits that are described in this chapter will be the transistor output circuits. The transistor output circuits will enable the programmer to operate a wide variety of dc components with the speed, accuracy, and repeatability that is inherent in a computer. The transistor circuits will be divided into three categories: low-power, medium-power, and high-power driver circuits. All of the presented circuits can easily be built on the experimenter's board so you can learn how to use them with your computer.

In all of the transistor circuits, the transistors

will be specified as Q1 for low-power applications, Q2 for medium-power applications, and Q3 for high-power applications. The transistor specification for each of the three power levels are described in Table 1-2 for low-power applications, Table 1-3 for medium-power applications, and Table 1-4 for high-power applications along with the transistors that fulfill the specified requirements. These transistors were selected because one or more of them will usually be available at an electronic supply store close to your location.

The transistor circuit in Fig. 1-12 is the basic npn transistor switch circuit. The operation of the circuit is very simple in that when the push button is pressed, a positive voltage is applied to the base circuit of transistor Q1. This positive voltage starts a current that flows into the base of transistor Q1, which causes the transistor to switch on. When there is no positive voltage applied to the base circuit of Q1, the transistor will remain in the off-state and no current can flow in the collector circuit. With no current flowing in the collector circuit, there will be no voltage drop across resistor R2 and the voltage level at the collector of Q1 (point "X") will be at plus six volts. If there is a positive voltage applied to the base circuit of Q1 (6 volts when PBI is pushed), a base current will flow and Q1 will switch on. When Q1 is switched on, a collector cur-

Table 1-2. Specifications for the Low-Power Switching Transistor Q1 and Some Selected Transistors that Match these Specifications.

The low power transistor for circuit application as 01 in the transistor output interface circuits.

Required Transistor Specifications	
Type	NPN
Material	Silicon
Power Dissipation	.5 Watts
Collector Current	.5 Amp
Maximum Usable Frequency	30 MHz
Base to Emitter Voltage	5 Volts
Collector to Emitter Voltage	40 Volts
Input Base Current	4 mA
Transistor Gain	150

Some transistors which meet or exceed the low power transistor specifications and are available at most retail electronics stores are given below.

Manufacturer	Part Number
Calectro (CG Electronics)	J4-162B
Radio Shack	276.2009
Sylvania	ECG.123
RCA	SK-3444
GE	GES6004
2N Type Number	2N2222A

Table 1-3, Specifications for the Medium-Power Switching Transistor Q2 and Some Selected Transistors that Match these Specificationa.

The medium power transistor for circuit application as 02 in the transistor output interface circuits.

Required Transistor Specifications	
Type	NPN
Material	Silicon
Power Dissipation	5 Watts
Collector Current	1 Amp
Maximum Usable Frequency	3MHz
Base to Emitter Voltage	4 Volts
Collector to Emitter Voltage	40 Volts
Input Base Current	400 mA
Transistor Gain	15

Some transistors which meet or exceed the medium power transistor specifications and are available at most retail electronics stores are given below.

Manufacturer	Part Number
Calectro (C G Electronics)	J4-1649
Radio Shack	276-2018
Texas Instruments	TIP 29
Sylvania	ECG-188
RCA	SK-3893
GE	044 C4
2N Type Number	2N 1701

Table 1-4. Specifications lor the High-Power Switching Transistor 03 and Some Selected Transistora that Match these Specifications.

The high power transistor for circuit application as 03 in the transistor output interface circuits.

Required Transistor Specifications	
Type	NPN
Material	Silicon
Power Dissipation	40 Watts
Collector Current	3 Amp
Maximum Usable Frequency	3 MHz
Base to Emitter Voltage	5 Volts
Collector to Emitter Voltage	40 Volts
Input Base Current	1 Amp
Transistor Gain	20

Some transistors which meet or exceed the high power transistor specifications and are available at most retail electronics stores are given below.

Manufacturer	Part Number
Calectro (C G Electronics)	J4-1654
Radio Shack	276-2017
Texas Instruments	TIP 31
Sylvania	ECG-152
RCA	SK-3054
GE	D44C5
2N Type Number	2N148S

rent will flow that is limited by the resistance of R2. Transistor theory will tell you that a voltage drop across a silicon npn transistor is about. 7 volts when the transistor is turned on. This fact will cause a voltage drop across R2 of 5.3 volts which leaves only.7 volts at point "~X."~Now if point "Z" is connected to an 110 port line and not the push button, the transistor switch can be turned off and on by the computer.

The above explanation of the transistor switching circuit brings out a point which should be discussed further, This point is the fact that the computer 110 port can in theory drive the transistor switch circuit. If a logic ONE signal is placed on an 110 port line that is connected to the switch circuit, the positive voltage from the logic ONE signal will tum on the transistor switch. When the transistor is turned "on", the voltage at the collector of the transistor goes to .7 volts or a 10gic ZERO. This means that a logic ONE into the transistor switch circuit generates a logic ZERO out of the switch circuit. This shows that the transistor switch functions as a logic inverter circuit. The output circuit characteristics of the 110 port really prevents any useful operation with a straight transistor switch circuit because the port is designed to work with one TTL load. TTL circuits operate on a principle that is known as current sinking which means that the port is not designed to output a voltage, but it is designed to pull down a voltage through a resistive network as long as the current does not

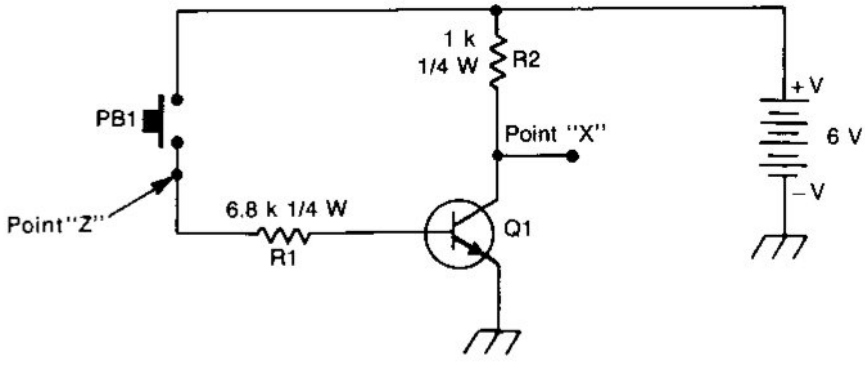
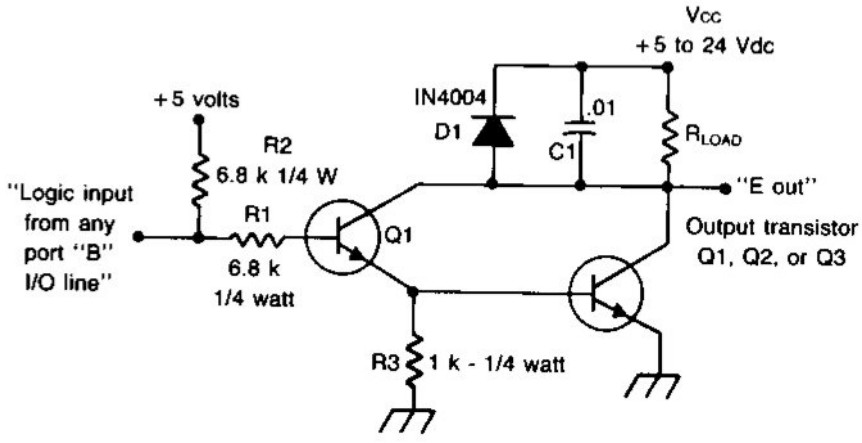


Fig. 1-12. This is the basic schematic 01 the simple transistor switching circuit.

exceed more than one milliamp (check your own computer specifications).

The Darlington transistor circuit that is shown in Fig. 1-13 will work nicely with any of the computer *1/0* port B lines. Resistor R2 is a pull-up resistor that supplies the transistor turn-on drive current when the 1IO port line is in the logic ONE state. When the 110 port line is in the logic ZERO state, the current through resistor Rl flows into the computer port circuit (remember the port is designed to sink a dc current) and cannot turn on the transistors. The current gain of the Darlington circuit gives one the ability to switch on and off a dc current of up to at least 150 milliamps with a computer 110 port line, This switching capability now gives our personal computer the ability to do work. Diode Dl and capacitor Cl are placed across the dc load to suppress any inductive noise that might be generated when the load current is switched on and off. The problem of e1ectricalnoise being generated in this manner can cause noise problems inside of the computer and much care must be used to prevent this noise problem.

Figure 1-14 shows a Darlington transistor circuit being used to switch a number 47lamp on and off. This circuit does not use the noise suppression diode and capacitor because the number 47 lamp is a resistive load and does not generate any inductive noise. One point should be noted at this time. If the lamp is located far away from the computer, such as 50 feet, the wires going to the lamp can generate the inductive noise. In this situation, the diode and capacitor are still needed at the point in the circuit where the Darlington transistor circuit



Notes:

1. The output transistor is selected so it will safely handle the required output current through RLoAD'

2. "E OUT" will switch from a Vee level for a logic zero input to about .7 volts for a logic one input.

3. The load current in RLOADs about equalto:

$$\text{CurrentRL} = \frac{\text{Vee} - .7}{\text{Resistance of RLOAO}}$$

Fig. 1-13. This circuit is a Darlington transistor circuit that is designed for USER PORT interface applications.

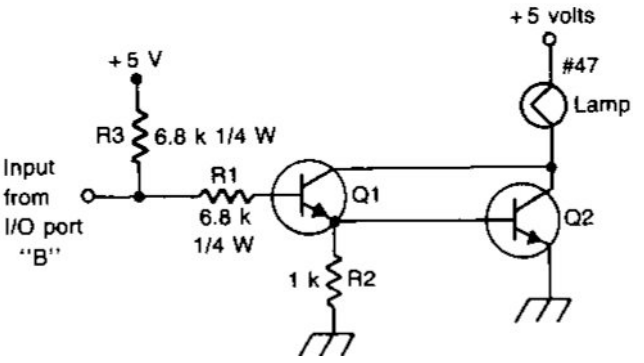


Fig. 1-14. A simple Darlington lamp-driver circuit.

is located. Figure 1-15 shows how to drive a low-power relay with a Darlington circuit.

The circuit described next is an ac power-line control circuit and it should not be built by a beginner. Also, this circuit should not be built on an experimenter's board because, if a 120 volt power line wire should slip out of a plug-in hole and touch the wrong part, a disaster in the form of smoke, fire, and tears could result. Remember, the power available from the ac power line can easily bum up (smoke-up) an experimenter's board if a short circuit occurs. The circuit in Fig. 1-16 is an extended application of the relay circuit of Fig. 1-15. When the relay closes, triac TR-l will turn on supplying power to the ac load.

USING INTEGRATED CIRCUITS FOR I/O INTERFACING

One can say in a general type of statement that the Port B 110 circuits in the computer's 110 interface chip were designed for TTL operation. If you would POKE memory locations 37138 and 37136

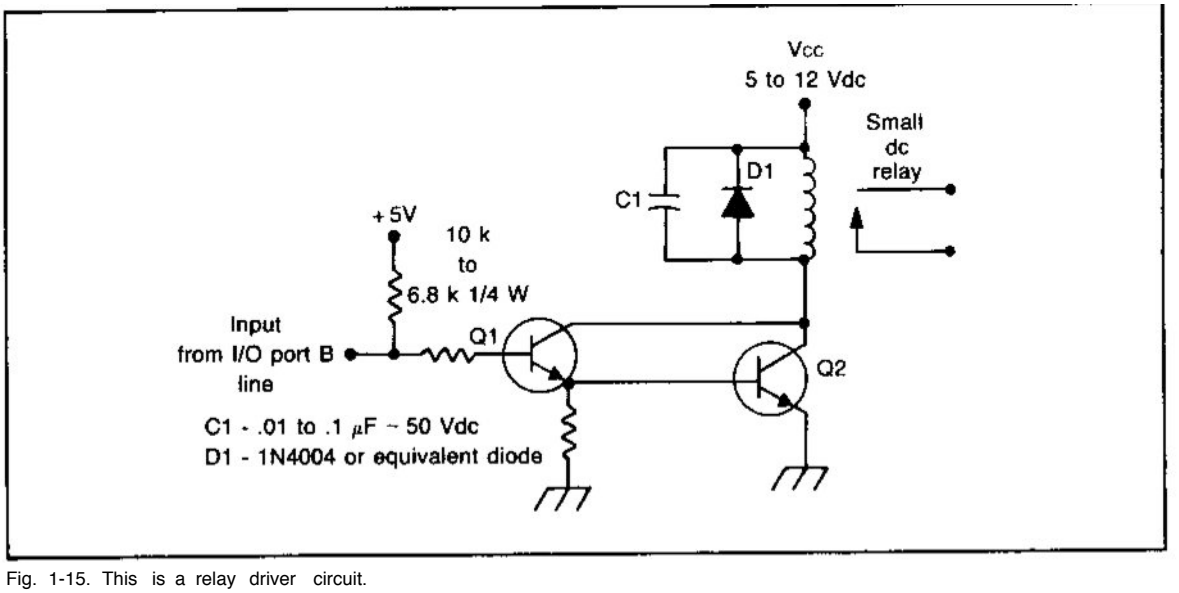


Fig. 1-15. This is a relay driver circuit.

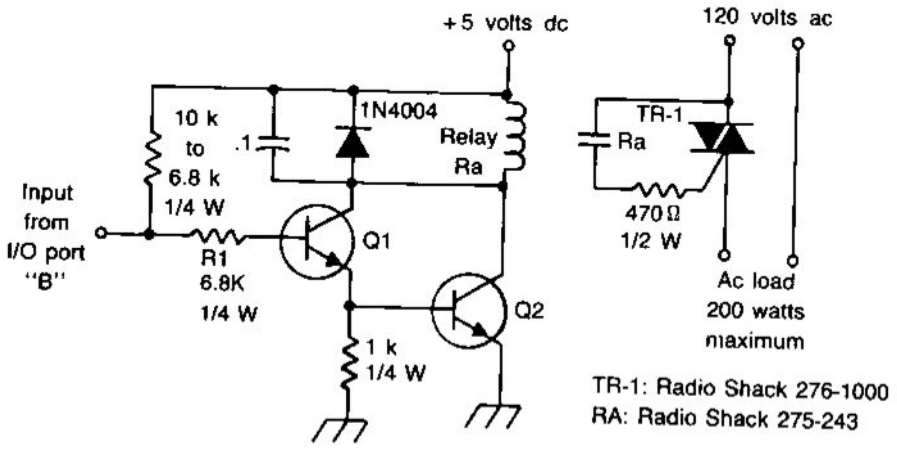


Fig. 1-16. This circuit shows one method that could be used to safely control ac loads with a computer. Don't build this circuit on UEB-I or UEB-2.

with 255 in the VIC-20 and then measure any output port B line voltage, you would find that it is about 2.7 volts or so for a 10glc ONE signal. This is because the port B line is TTL compatible. If you want a full five volt swing from the port B line, you must use a 6.8K pull-up resistor between the port line and the computer's five-volt supply. The VIC-20, the C-64, and the PLUS/4 all have different I/O port interface output chips and it would be wise to read the section of the programmer's guide which explains the I/O port operation. But, as long as you do not exceed the requirements of the one TTL load specification, you will not damage any of the port lines,

The two main logic families that are used in this book are TTL and CMOS. The majority of all computer interfacing requirements can be done with only a few of the various TTL and CMOS circuit chips. CMOS chips require a full five volts logic swing for their operation and you must use a pull-up resistor as described in the preceding paragraph when you are connecting a CMOS chip to a port B I/O line for an output function. Tables 1-5 and 1-6 describe some of the general TTL and CMOS characteristics. Tables 1-7 to 1-11 describe some specific TTL and CMOS Ie circuit chips that can be used for interfacing projects,

Table 1-5. General TTL Operating Characteristics.

A. Dc power connections: +5 volts and ground or common.
B. TIL circuits should be bypassed by a .1 μF capacitor at the +5 volts input pin.
C. Unused inputs should be tied to +5 volts.
D. Know what type of output cjrcuit is in the TIL chip you are using; standard, open collector, or 3 state output (High-low-open circuit).
E. Input conditions: A "One"" Logic level must be more than about 2.5 volts. A "Zero" logic level must be below about .8 volts.
F. The operating speed of the TIL gates are generally between 10 and 30 nanoseconds, which js far faster than any low priced single board computer can operate.
G. Output Orjve (Fan-Out) The 7400 series will drive ten 7400 serjes inputs. The 74LS .. series will drive two 7400 series or ten 74LS series inputs.

Table 1-6. General CMOS Operating Characteristics.

A. Dc operating current requirements are very small compared to TTL.
B. All inputs are high impedance inputs which require no driving power.
C. The design characteristics give CMOS chips good noise immunity.
D, The CMOS "B" series chips will function from 3 volts to over 15 volts with some operating up to 20 volts.
E, The outout voltage will go from the positive voltage supply level to ground level. (complete + to - voltage swing.)
F. All inputs "must" be connected to an input line or to the + voltage line or ground line, CMOS is a high impedance input logic family and unconnected inputs can easily pick up noise.
G. The operating speed of CMOS is not as fast as **TTL** at the 5-volt supply level. The best supply voltage for CMOS circuits is between 9 and **12** volts.

Table 1-6 describes a Schmitt-trigger logic circuit which is one of the most versatile interfacing chips you can buy that can be used for both input and output projects. A Schmitt-trigger logic circuit has the special ability to turn on at a specific positive-goingvoltage and then turn off at a specific negative-going voltage. These voltage trip points are referred to as the positive and negative going threshold voltages. The threshold voltages are set so the positive threshold trip point is a little higher than the negative threshold trip point giving what is called an area of hysteresis. The hysteresis area between the positive threshold point and the negative threshold point is generally about .6 volt to 1 volt for logic circuits with 5-volts supply voltages. The Schmitt-trigger IC can convert a slowlyrising or falling logic signal to neat and clean logic signal with fast rise and fall characteristics. Schmitt-trigger IC chips are especially useful for computer interfacing because they can be used with great success in preventing noise from getting into an 110 port line from the outside world. As you read on in this book, you will find that Schmitt-trigger circuits are used in many different circuit applications other than just interfacing.

The IC circuits that are described in Tables 1-7 to 1-10 are either inverting or noninverting buffer circuits. These circuits can be used to interface the computer to other logic families and logic voltage levels than the standard five-volts logic. We first introduced you to the transistor switch as a method of computer interfacing, but it is wise for a beginner to use one of these buffer circuits between the computer and the circuit you are interfacing, because if you make a mistake and connect to the wrong signal or voltage level you will only blowout a cheap buffer circuit and not a computer port line circuit.

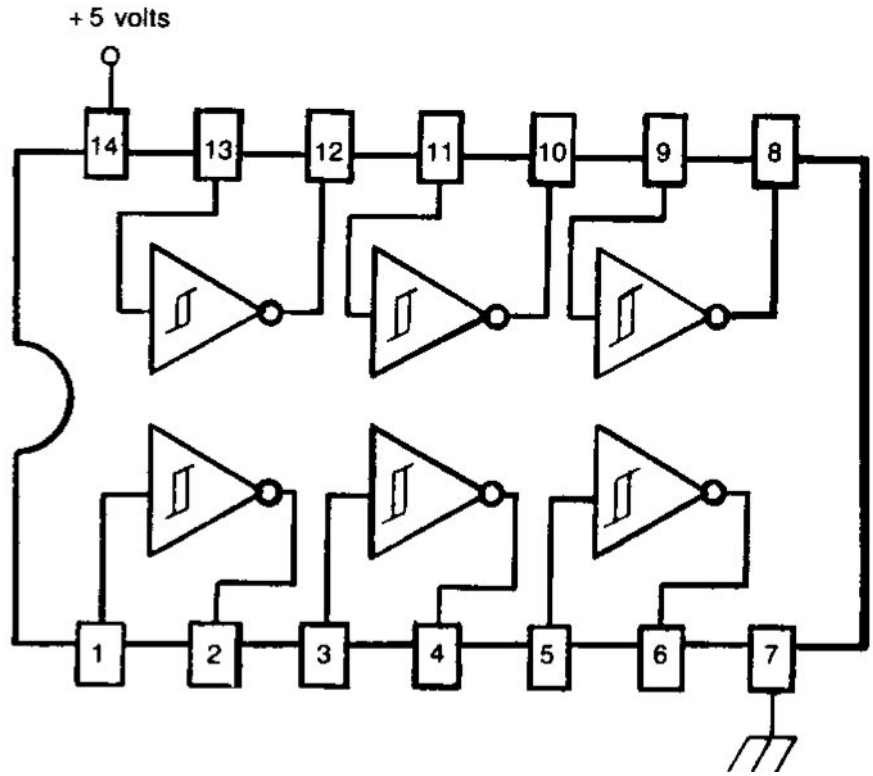
EXAMPLE *1/0* CIRCUITS USING TTL AND CMOS CHIPS

The next group of I/O circuits that will be presented are examples of how you can use the TTL and CMOS IC chips to perform input and output operations with a computer. Actually, the IC circuits that are presented in Figs. 1-17, 1-18, and 1-19 can be used to perform most of the needed *I/O* functions that will be encountered in normal control system tasks. Computer programs that will work with these IC circuits will be presented later in this chapter.

Figure 1-17 shows a CMOS MC14584 Hex Schmitt-trigger IC connected in a four function operation. Pins 10, 11, 12, and 13 are connected in a logic input circuit. Pins 8 and 9 are connected in a 12-volt pilot lamp driver. Pins 1, 2, 3, and 4 are connected as LED drivers. Remember that CMOS circuits are high impedance inputs so any unused inverter circuits must have their inputs tied to ground so the inverter circuit will not oscillate. An oscillating inverter circuit can cause the circuit chip to overheat or generate unwanted signal noise. One more general note about Fig. 1-17. CMOS inputs are designed to function with logic levels that switch between *zero* and five volts. The port B computer outputs need a pull-up resistor to pull their

Tabte 1-7. A CMOS Hex Schmitt Trtgger tnverting Buffer IC.

MC 14584 MM 74C14



The IC package contains six separate inverter circuits. Each inverter circuit has a Schmitt trigger which gives an internal switching hysteresis characteristic that makes this IC very ideal for interfacing noisy circuits to a computer input port.

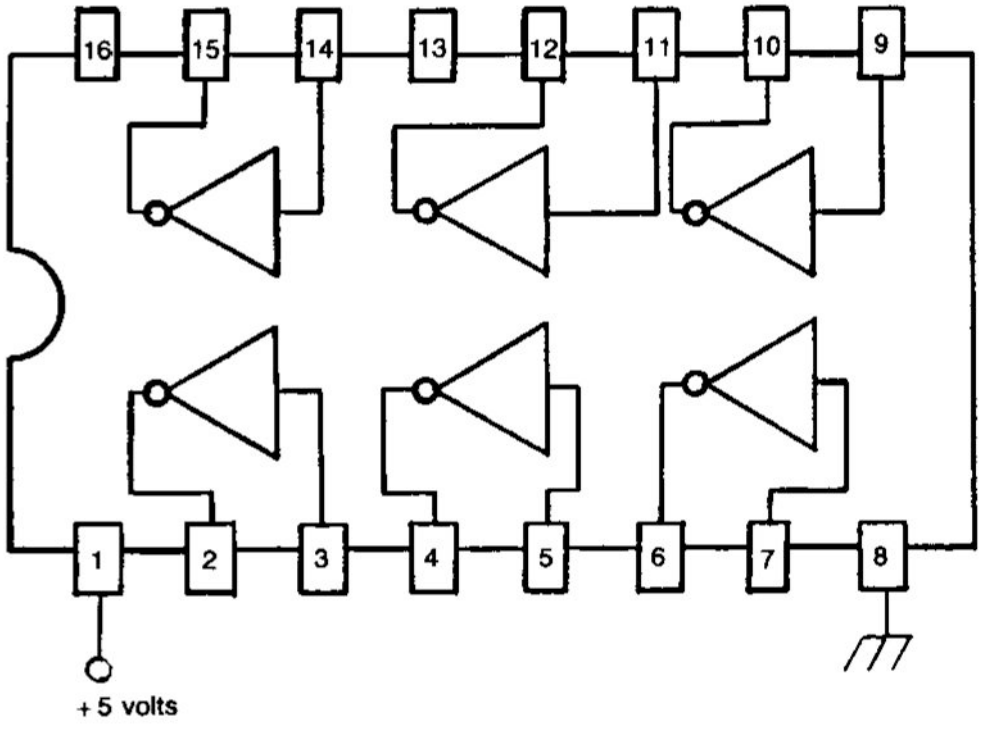
On a positive going input signal, the output state will change when the input signal is about 58 percent of the supply voltage. On a negative going signal, the output state will change when the input signal, the output state will change when the input signal is about 24 percent of the supply voltage. The actual dead band "hysteresis" is about .61 volts for a pin 14 supply voltage of 5 volts. Five volts is the required supply voltage when using this IC to interface input data into the USER's port of a VIC-20, C-64, or Plus/4 computer.

output up to the five-volts logic ONE level for CMOS operation. Resistors R1, R2, R3, and R4 function as pull-up resistors in this figure.

Looking at the circuit of Fig. 1-17 shows that the circuit connected to pins 12 and 13 is an "input" push-button circuit and the circuit connected to pins 10 and 11 is an "open-closed" switch circuit. The main purpose of this circuit is to show you how to

Table 1-8_This CMOS Inverting Buffer Can be Used lor Input Interlacing to Logic Voltage Levels Other thsn the Standard Five-Volt Computer Logic Signals.

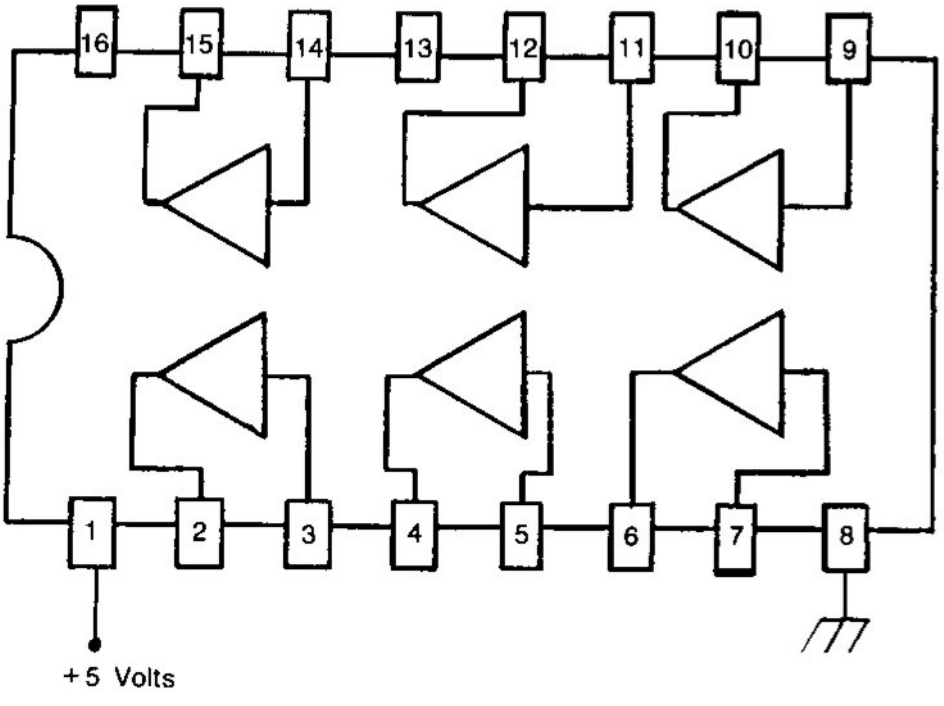
MC14049 CD4049



This IC package contains six separate inverter circuits. The main application of this inverting buffer IC is to give the 110input line the capability of interfacing to other voltage levels than just five volts. With a supply voltage of five volts, any input logic level between 5 and 15 volts can be safely applied 10 the inverter inputs. One should note the unusual supply connections with the + V being connected to pin 1 and dc common being connected to pin 8.

Table 1-9. Thia CMOS NonInverting Buffer Can be Used for Input Interfacing to logic Voltage levels Other than tIM Standerd Five-Volt Computer logic Signala.

MC 14050 CD 4050



ThiS le package contains six separate noninverting buffer circuits. The main application of this noninverting buffer le is to give the 110 input line lhe capability of interfacing to other voltage levels than just five valls. With a supply voltage of five volts, any input logic voltage level between 5 and 15 volts can be safely applied 10 the inverter inputs. One should note the unusual supply connections with the + V being connected to pin 1 and dc common being connected to pin 8.

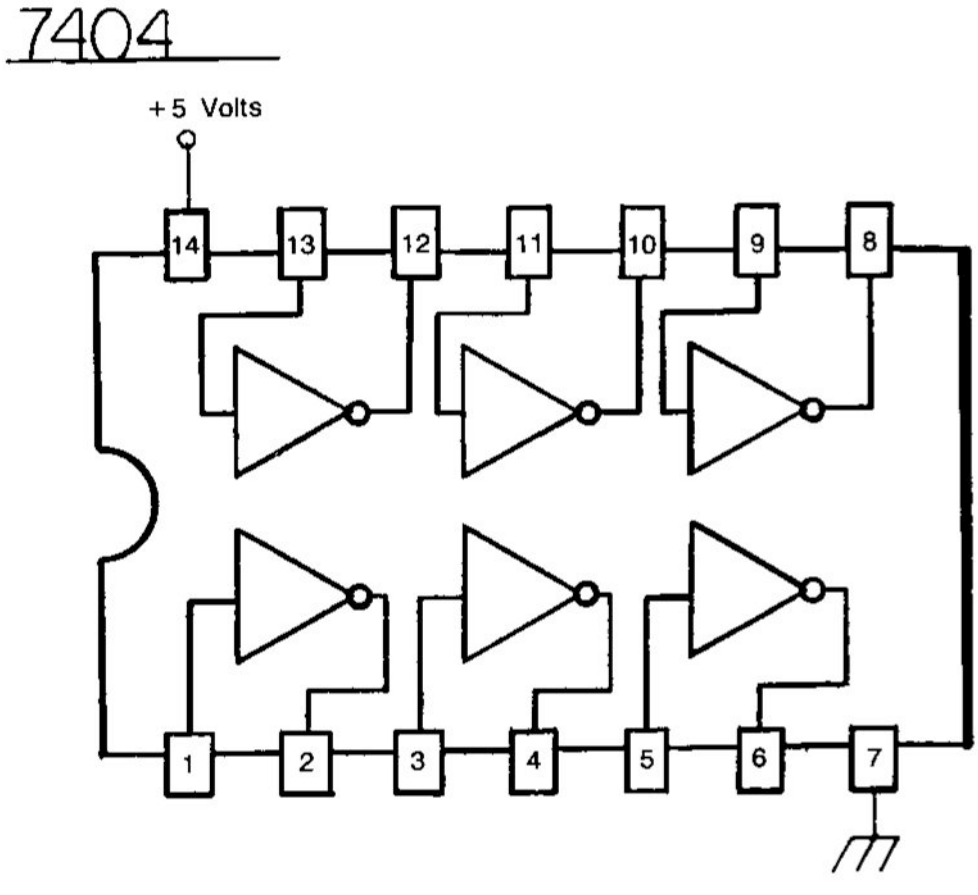
connect a pushbutton or switch to your computer to start or stop the program operation. If the switch is turned on or the pushbutton pressed, a logic ONE signal will appear at output pins 10 or 12. The logic signal output of pins 10 and 12 should be connected to either port bit 6 or 7 because there is a special machine language BIT instruction which makes it very easy to read the logic level of input bits 6 and 7.

The circuit that is connected to pins 8 and 9 is used to turn a pilot light on and off. A logic ZERO at pin 9 will generate a logic ONE at pin 8 that will

turn on the lamp. The lamp could actually be any dc load within the power ratings of transistor Q2. Remember to use a noise-suppression diode and capacitor across the load if it has any inductive components. The circuit that is connected to pins 1 and 2 is a simple LED circuit that is designed to turn on LED 1 when a logic ONE is at pin one. The LED circuit at pins 3 and 4 is designed to turn on when a logic ZERO is at pin 3. Pin 5 is grounded because this inverter circuit is not being used.

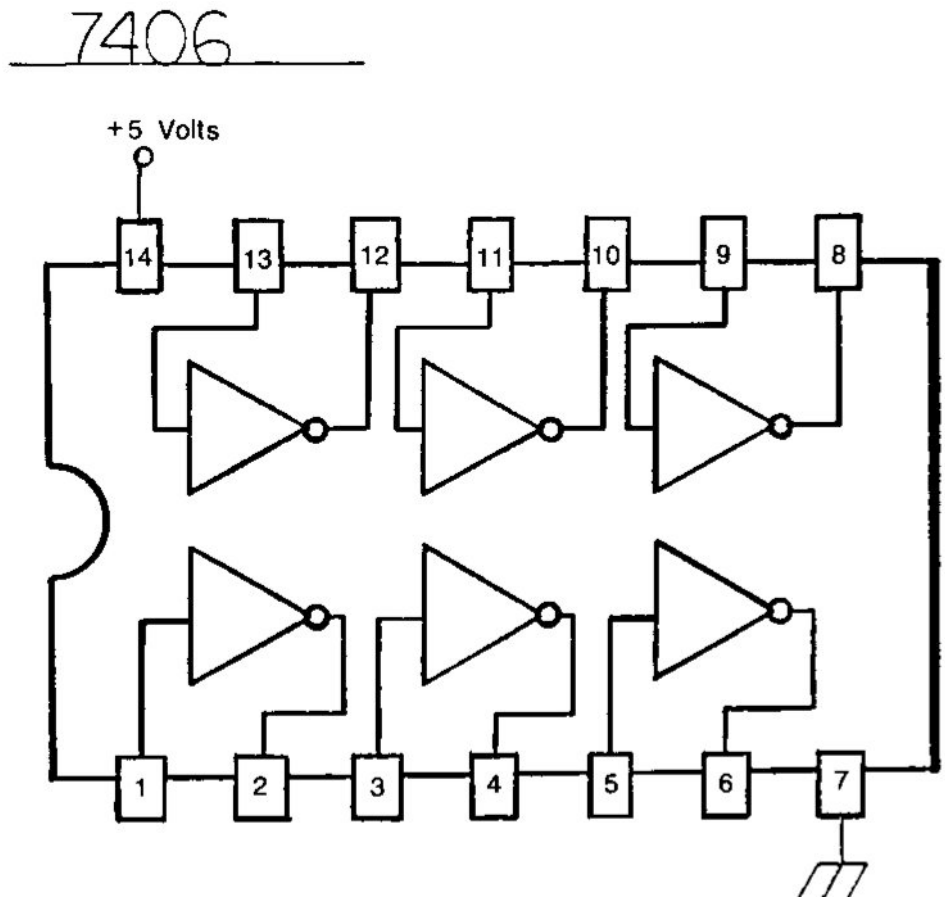
Figure 1-18 shows a CMOS MC14050 Hex noninverting buffer circuit that is being used to interface logic input signal voltages that are higher than the normal 5-volts computer levels. If a logic signal inversion is needed, the MC14049 can also be used for the same voltage level interfacing since it is an inverting buffer IC. The IC has a supply voltage of 5 volts applied to pin one, which sets-up the output logic voltage swing of 0 to 5 volts. With a supply voltage of 5 volts, an input logic ONE level

Table 1-10. Standard TTL Inverting Buffer Chip.



This Ie package contains six separate inverter circuits. The main application of this IC is to increase the TIL driving capability 01 the computer flO port. Using this IC as an inverting buffer in an output line circuit will give the I/O line the capability to drive to TIL loads.

Table 1-11. This Inverting Buffer TTL Chip Can Drive Output Loads at Voltage Levels Up to 30 Volta.



This Ie package contains six separate open collector inverter circuits. The main application of this Ie is 10 increase the TTL driving capability of the computer *110* port. Using Ihis Ie as an inverting buffer in an output line circuit will give the *110* line the capability of interfacing 10 other voltage levels than just five volts. When the inverter output is low, it can sink 30 milliamperes per gate, and when the output is high, it can handle circuit voltages up to 30 volts. This gives the computer *110* port line the capability to drive low power circuits operating in the range of 12, 15, and 24 volts. The Vee supply voltage at pin 14 for this chip is five volts. If all six inverters are operated at the maximum gate current of 30 milliamperes per gate, you must make sure that the Ie package does not overheat.

anywhere between 5 and 15 volts can be applied to any of the six buffer circuits in the Ie. In this circuit, a pushbutton (PB-I) that is connected in a 12-volts logic system is being interfaced to PB7, which is operating at the normal 5-volts level. PB-2 is functioning at 14.8 volts and is being interfaced to PB6 at the normal 5-volts level.

Figure 1-19 shows a 7406 TTL chip being used as an output driver to drive various output loads at voltage levels between 5 and 24 volts. This cir-

cuit chip can control a little more power because each of the six gates in this package is an open-collector transistor circuit that can sink 30 milliamperes per gate. If each gate is operated at 30 milliamperes, you must make sure that the Ie chip does not overheat. In this figure, the circuit that is connected to pins 10 and 11 is a simple resistor that can supply 5 milliamperes of dc driving current to an external circuit. The circuits using LED-1 and LED-2 simply show how to connect LEDs in circuits with 12- and 24-volts supply voltages. Note that pin 14 has the standard 5-volts TTL voltage connected to it, but the open collector concept of the chip gives it the ability to switch voltage levels up to 30 volts.

This now concludes our beginning discussion on *1/0* computer operations. The technology that has been presented to you up to this point will give you the ability to do any programmable control tasks that are needed in a control system. The presented 110 circuits along with the software capabilities of the Commodore computers gives one

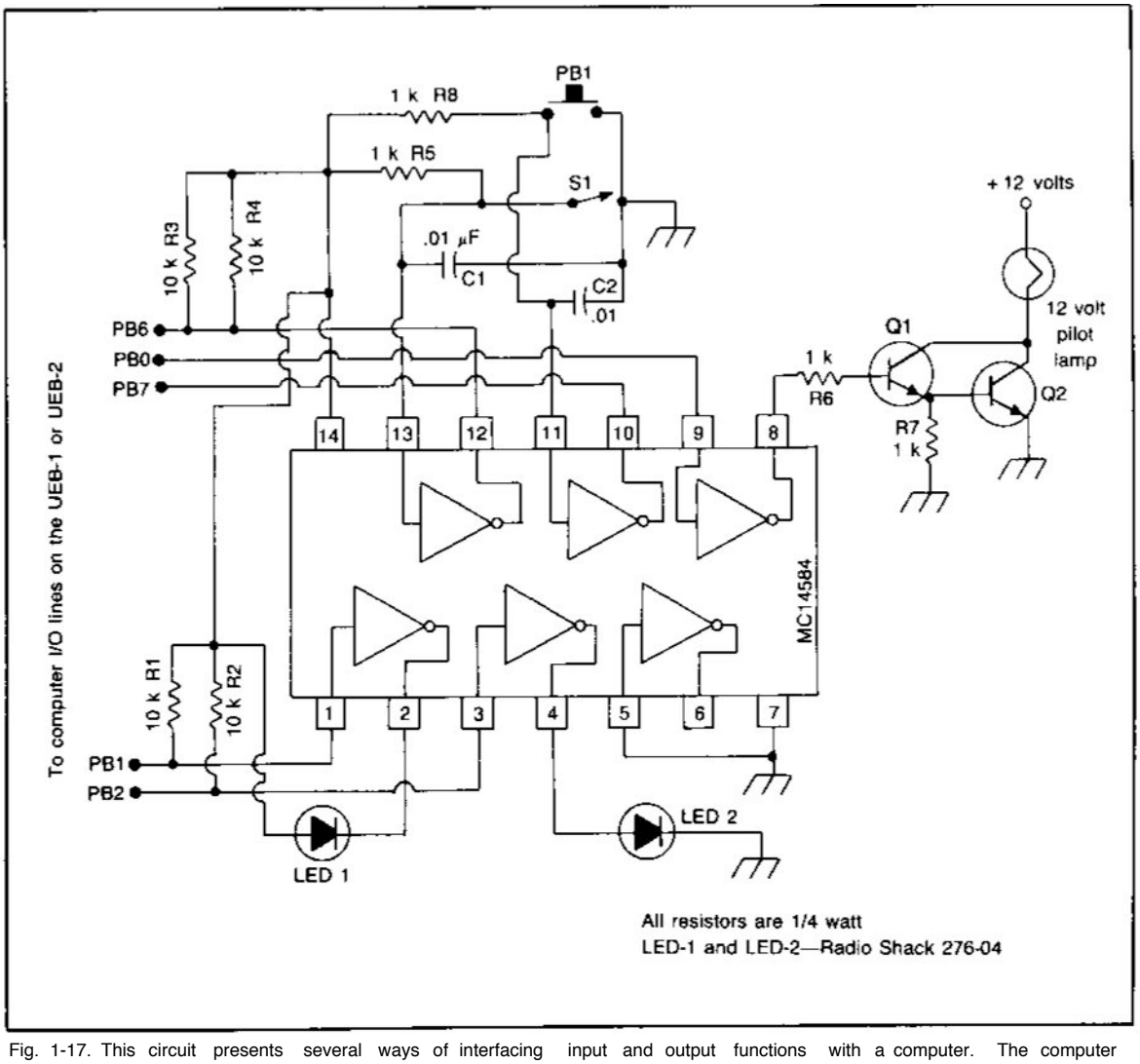


Fig. 1-17. This circuit presents several ways of interfacing input and output functions with a computer. The computer programs that are presented in Programs 1-1, 1-2, and 1-3 use this CMOS IC circuit.

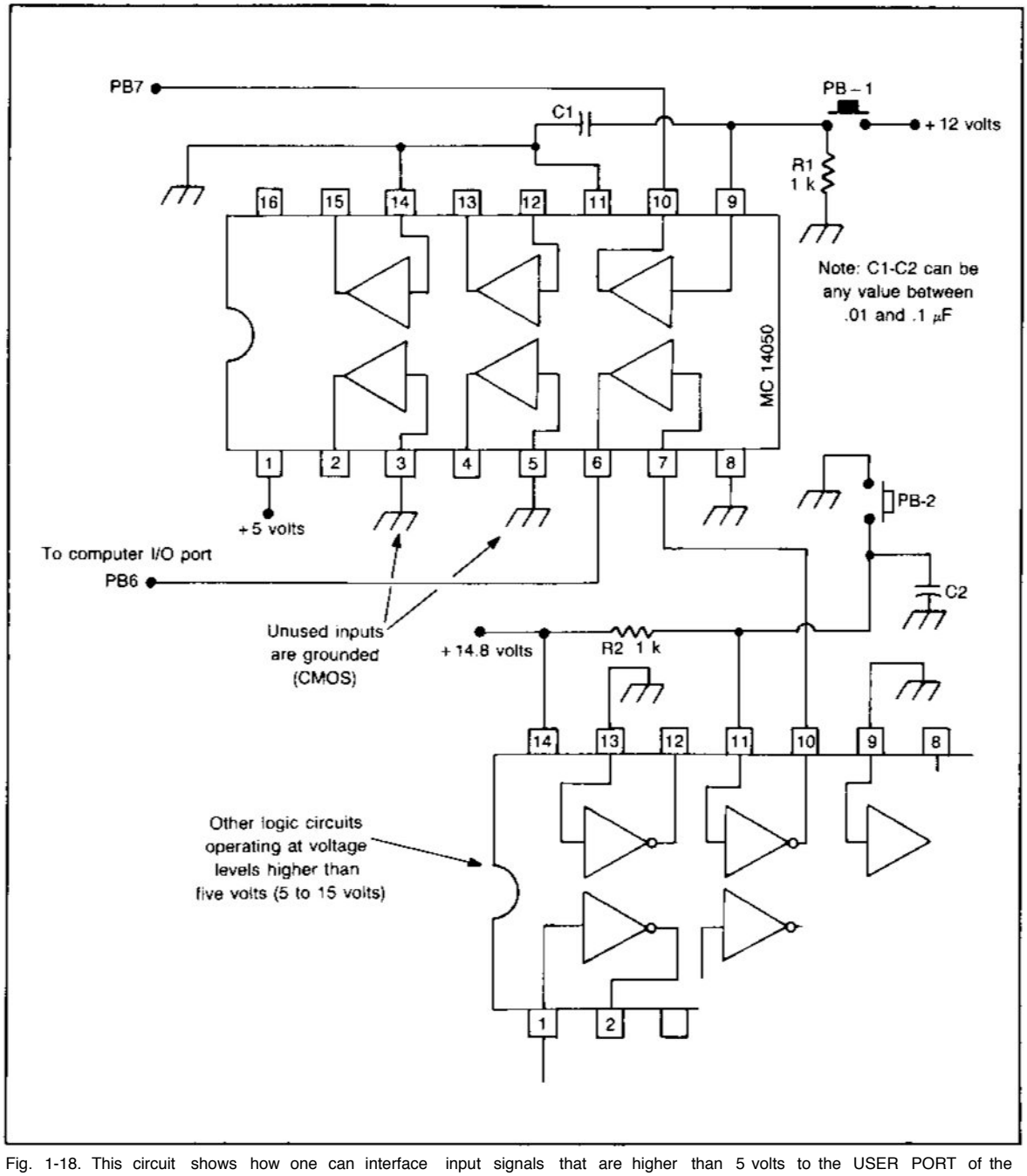


Fig. 1-18. This circuit shows how one can interface input signals that are higher than 5 volts to the USER PORT of the VIC-20, C-64, or PLUS/4.

a programmable controller that can compete and surpass any of the industrial programmable controllers that are currently being sold. A series of experimental programs and circuits will now be presented to give you some hands-on experience with the technical presentations of this chapter.

PROGRAM EXPERIMENTS

Programs 1-1 through 1-21 are presented to give you the fundamental knowledge of computer input-output interfacing. Each of these experimental programs has been designed to work with a specific electronic circuit that can be built on one of the USER PORT Experimenter's Board (UEB-1 or UEB-2), These experiments were designed so they could be performed without injury to the computer under even the worst mistakes. About the worst mistake you could make would be to short out the power supply in the computer. The *VEB-I* and UEB-2 have a 1/4-amp fuse that should blow before the fuse inside of the computer blows. Each of the Commodore computers have an internal fuse that is mounted on the computer circuit board and can only be replaced if you open up the computer case. If you can buy a lower rated fuse of about .1 amp for the *VEB* board, do so, but I believe that most electronic shops only sell a *1/4* fuse as their smallest size fuse, The best protection against a mistake is to always keep the wiring neat and double check all wiring.

Program 1-1 is written to use the electronic circuit of Fig, 1-17. The program and circuit combination represents the basic input-output control

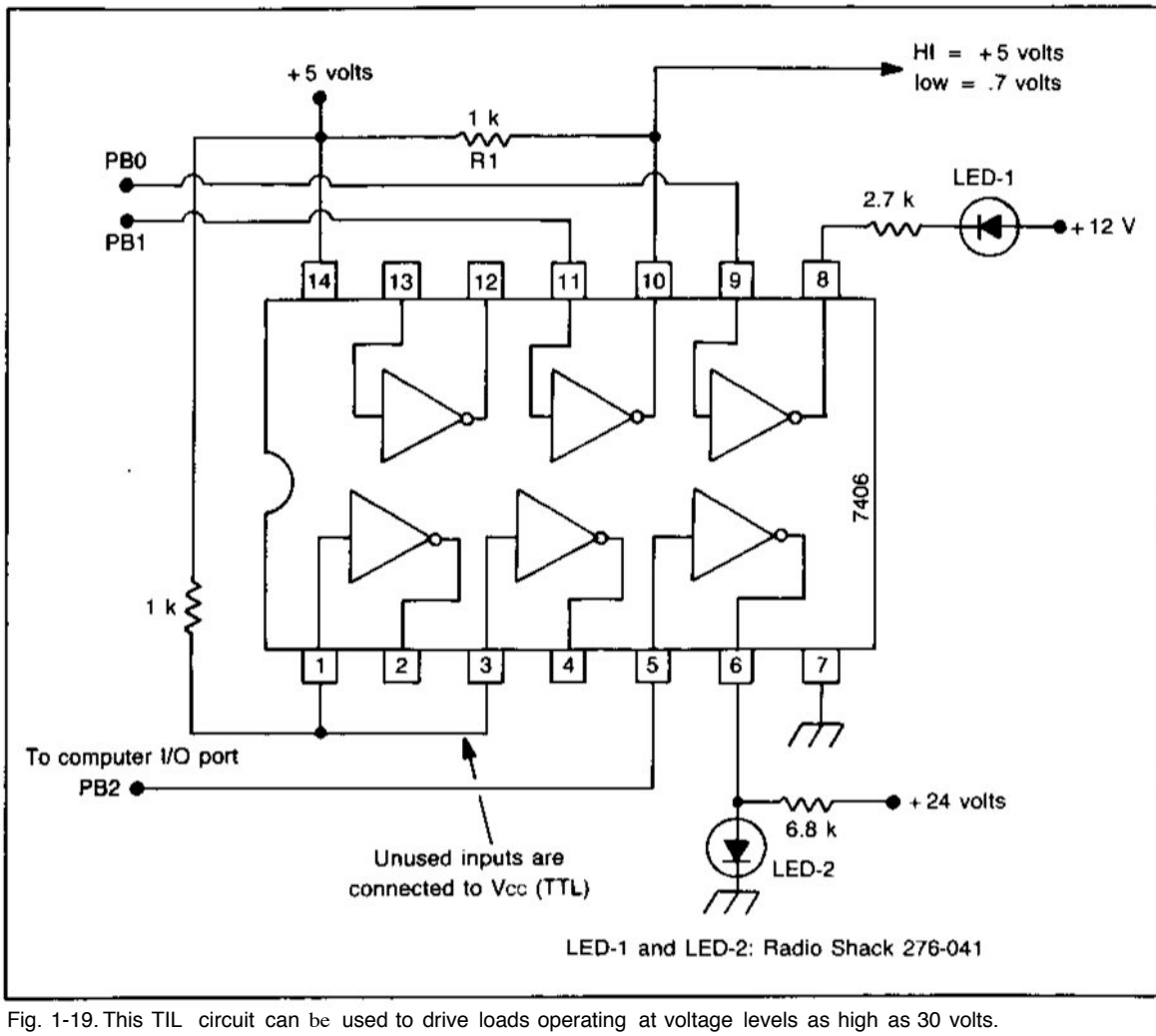


Fig. 1-19. This TIL circuit can be used to drive loads operating at voltage levels as high as 30 volts.

operation of a computer. Program 1-1 is for the C-64, Program 1-2 is for the VIC-20, and Program 1-3 is for the PLUSI4 computer. The general operation of the program causes LED-1 to light when PB-l is pressed, LED-2 lights up when 51 is switched on, and both LED-1 and LED-2 plus the pilot lamp light when PB-1 and 51 are turned on together. The program also presents a video display that tells you what input (PB-l or 51) is on and what outputs (LEDs and light) should be on.

The functioning of Programs 1-1, 1-2, and 1-3 works as follows. Lines 10 and 20 set up the U5ER PORT 110 operation by making bits 6 and 7 inputs and bits 0 to 5 outputs and then setting the output bits so that the LEDs and the light are off. Line 30 checks to see if any of the input devices (PB-1 or S1) have been turned on. Lines 50 to 80 decide which input has been turned on, if any. If an input has been turned on, then lines 90 to 110 light up the proper LED or light. Lines 200 to 225 control the video presentation while lines 300 to 330 check to see if both the pushbutton and the switch are on.

Programs 1-4, 1-5, and 1-6 are used with Fig. 1-20 to demonstrate how a relay can be operated by computer control. In these programs, line 10 sets up the USER PORT 110 operation by making all eight of the 110 lines output lines and then turning all outputs off. Lines 15 and 20 tell you to push '1' to turn on the relay or push '0' to tum off the relay. Lines 30, 40, and 50 check to see if the '1' key or the '0' key have been pushed. Lines 50 and 60 are used to POKE the proper data into the data register to turn the relay on or off. Lines 80 and 90 are used to print a video message that tells you if the relay is off or on.

Programs 1-7, 1-8, and 1-9 along with the circuit in Fig. 1-21 are used to show how you can control the operation of a dc motor with a computer, In Fig. 1-21, transistors Ql and Q2 are connected in a Darlington arrangement to drive the switching transistor Q3. The de motor is used as the load for Q3, and when Q3 is turned on, the motor runs. Pushbuttons PB-l and PB-2 are connected to 110 lines PB6 and PB7 for the start/stop operation of the motor. LED-l should light up when the motor is running.

```
5 REM PROGRAM 1.1 FOR FIGURE 1.17
B REM FOR VIC-2I2J
1121POKE 37 138 ,12183REM TH IS MAY. ES B ITS 121TO 5 OUTPU TS AND B ITS 6 AND 7 INPUTS
2121POKE 3713B,0S:REM THIS TURNS OFF THE TWO LEOS AND THE LIGHT
3121A=PEEK(37136)
4121PRINTCHR$(147):PRINT''PUSH PBI OR SWITCH SI"
5121IF A<7 THEN GOTO 3121
6121 IF 1'1=197 GOTO 9121
7121IF A=69 GOTO 1121121
8121IF 1'1=133 GOTO 11121
3121 POKE 37136, 1212:GOT0200
t0121POKE 37136,01:GOT022121
110 POKE 37136,07:GOT021121
200 PRINT" ":PRINT"B0TH THE SWITCH AND PSI ARE ON"
205 PRINT" ":PRINT" THE l.IGHT AND BOTH LEOS SHOULD BE ON";GOT03121I2J
210 PRINT" ":PRINT"PB-1 IS PRESSED"
215 PRINT" ":PRINT" LED-I SHOULD BE ON":GOT031210
22121PR INT" .:PR INT"S I IS SWITCHED ON"
225 PRINT" ":PRINT" LED-2 SHOULD BE ON":GOT03121121
31210B=PEEK(37136)
310 IF B>150THEN GOTO 30
320 IF B>1217 THEN GOT030I2J
330 GOTOII2I
```

Program 1-1. This program is written to use the Circuitry of Fig. 1-17 and the UEB-1 to demonstrate the basic input and output operation of the USER PORT of the C-64.

```
:5 REM  PROGRAM    1,2  FOR  FIGURE    1.17
6  REM  FOR  C-64
10  POKE   56579,063'REM      THlS   MAKES   BITS   a  TO  5  OUTPUTS    AND  BITS  6  ANO  7  INPUTS
20  POKE   56577,1'I5'REM    THIS   TURNS   OFF   THE   TWO   LEOS   AND   THE   LIGHT
31'1A=PEEK(56577)
41'1PRINTCHRS(147):PRINT"PUSH         PBI   OR   SWITCH    51"
51'1IF  A<7   THEN   GOTO   31'1
60   IF  A=197   GOTa   91'1
71'1IF  A=69    GOTO   11'11'1
81'1IF  A=133   GOTO   111'1
91'1POKE   56577,    1'12:GOT021'11'1
101'1POKE   56577,I'II:GOT0221'1
110  POKE   56577,1'I7:GOT0210
21'10PRINT"    ":PRlNT"BOTH     THE  SWITCH   AND   PBI   ARE   ON"
205   PR INT"   .: PR INT"       THE   L JGHT   AND   BO TH   LE OS   SHOULD    BE   ON": GOT031'11'1
211'1PRINT"    "'PRINT"PB-l      IS  PRESSED"
215   PRINT"    "'PRINT"        LED-l    SHOULD    BE   ON":GOT031'10
220   PR INT"     :PR INT"S  I   IS  SWITCHED     ON"
225   PRINT"    "PRINT"           LEO-2    SHOULD    BE   ON":GOT031'10
31'11B=PEEK(56577)
310   IF  B>15I'1THEN   GOTO   31'1
321'1 IF  B>1'I7   THEN   GOT031'10
331'1GOTO  tel
```

Program 1-2. This program is written to use the circuitry of Fig. 1-17 and the UEB-1 to demonstrate the basic input and output operation of the USER PORT of the VIC-20.

```
5  REM  PROGRAM    1.3  FOR  FIGURE    1.17
6  REM  FOR  THE  PLUS/4
10  POKE   64784,192:REM      THIS   MAKES   BITS   I'! TO  5  OUTPUTS    AND  BITS  6  AND  7  INPUTS
21'1POKE   64784,t97:REM      THIS   TURNS   OFF   THE   TWO   LEOS   AND   THE   LIGHT
25  PRINTCHR$(147)
31'1A=PEEK(64784)
40  PR INT  "iii"; PR INT"PUSH    PB I  OR   SII TCH   S I"
51'1IF  1'1<7THEN   GOTO   31'1
60   IF  A=197   GOTO   91'1
71'1IF  1'1=69   GOTO   11'11'1
81'1IF  1'1=133 GOTO   111'1
91'1POKE   64784,194     'GOT021'11'1
11'11PCKE   64784,t93:GOT0221'1
111'1POKE   64784,199:GOT0210
201'1PRINT"    ':PRINT"BOTH     THE  SWITCH   AND   PBI   ARE   ON"
21'15PRINT"    "'PRINT"        THE   LIGHT   AND   BOTH   LEOS   SHOULD    BE   ON"'GOT031'10
210   PR INT"    •ı PR INT" PB -1   IS  PRESSED"
2 t5  PR INT"     I PR INT"       LEO- 1  SHOULD    BE   ON"  :GOT031'11'1
220   PRINT"    ":PRINT"SI     IS  SWITCHED     ON"
225   PRINT"    ":PRINT'         LED-2    SHOULD    BE   ON":GOT031'11'1
300   B=PEEK(64784)
311'1IF  B)150THEN     GOTO   31'1
320   IF  B>1'I7   THEN   GOT031'11'1
331'1GOT010
```

Program 1-3. This program is written to use the circuitry of Fig. 1-17 and the UEB-2 to demonstrate the basic input and output operation of the USER PORT of the PLUS/4.

```
5 REM PROGRAM 1.4 FOR FIGURE 1.20
6 REM FOR THE C-64
10 POKE56579, 63: POKE56577,00
15 PR INTCHRS( 147): PR INT" PUSH '1' TO TURN ON THE RELAY"
20 PRINT" ":PRINT" PUSH '0' TO TURN THE RELAY OFF"
30 GET A$: IF AS = "1" GOT060
40 IF I'1$ = "0" GOTO 70
50 GOT030
60 POKE 5S577,01:GOT080
70 POKE 56577,00:GOT090
80 PR INT' ":PR HNT" THE RELAY ISON" :GOT030
90 PRINT" ":PRINT" THE RELAY IS OFF":GOT030
READY.
```

Program 1-4. This program, along with the circuit of Fig. 1-20, teaches one how to control a simple relay with a C-64 computer.

```
5 REM PROGRAM 1.5 FOR FIGURE 1_20
6 REM FOR THE VIC-20
10 POKE37138, 63: POKE37136,00
15 PRINTCHR$(147): PRINT' PUSH '1' TO TURN ON THE RELAY"
20 PRINT" ":PRINT" PUSH '0' TO TURN THE RELAY OFF"
30 GET I'1$: IF A$ " "I" GOT060
40 IF A$ = '0" GOTO 70
51"1GOT030
60 POKE 37136,eJl:GOT08eJ
70 POKE 37136,00:GOT090
80 PRINT" ":PRINT" THE RELAY IS ON":GOT030
90 PRINT" ":PRINT" THE RELAY IS OFF":GOT030
```

Program 1-5. This program, along with the circuit of Fig. 1-20, teaches one how to control a simple relay with a VIC-20 computer.

```
5 REM PROGRAM I.S FOR FlGURE 1.20
6 REM FOR THE PLUS/4
t0 POKE64784, 192
15 PR INTCHR$( 147): PR INT" PUSH '1' TO TURN ON THE RELAY"
20 PRINT" ":PRINT" PUSH '0' TO TURN THE RELAY OFF"
30 GET A$: IF A$ = "I" GOT060
40 IF A$ = "0" GOTO 71"1
50 GOT030
60 POKE 64784,193:GOT080
70 POKE 64784,192:GOT090
81"1PRINT" ":PRINT" THE RELAy' IS ON":GOT030
30 PR INT' ":PR INT' THE RELAY IS OFF" :GOT030
```

Program 1-6. This program, along with the cirCUit of Fig. 1-20, teaches one how to control a simple relay with a PLUS/4 computer.

```
5 REM PROGRAM 1.7 FIGURE 1.21
6 REM FOR THE C-64
10 POKE56579,63:POKE56577,01
20 PRINTCHR$(147): PRINT"MOTOR CONTROL PROGRAM'
30 PRINT" ":PRINT"PUSH THE PBS PUSH BUTTON TO START MOTOR"
40 PRINT" ":PRINT"PUSH THE PB7 PUSH BUTTON TO STOP MOTOR"
50 A=PEEK.(56577)
60 IF A = 65 THEN GOTO 200
70 IF A = 128 THEN GOTO 210
80 GOTO 50
100 PRINTPEEK(56577):GOTOI00
200 POKE 56577,00: PR INT "MOTOR ON": GOTO 50
210 POKE 56577,01:PRINT"MOTOR OFF": GOTO 50
```

Program 1-7. This program controls the elementary dc motor-control circuit of Fig. 1-21 with a C-64 computer.

```
5 REM PROGRAM 1.8 FIGURE 1.21
6 REM FOR THE VIC-20
IO POKE37138,63:POKE37136,01
20 PR INTCHR$ (147 ): PR I NT" MO TOR CONTROL PROGRAM"
30 PRINT" ":PRINT"PUSH THE PB6 PUSH BUTTON TO START MOTOR"
40 PRINT" ",PRINT"PUSH THE PB7 PUSH BUTTON TO STOP MOTOR"
50 A=PEEK(37136)
60 IF A = 65 THEN GOTO 200
70 IF A = 128 THEN GOTO 210
80 GOTO 50
l00 PR INTPEEK (37136 ):GOTO 100
201"PPOKE 37136 ,¤IePR INT"MOTOR ON", GOTO 5¤I
210 POKE 37136,01:PRINT"MOTOR OFF": GOTO 50
```

Program 1-8. This program controls the elementary dc motor-control circuit of Fig. 1-21. with a VIC-20 computer.

```
5 REM PROGRAM 1.9 FIGURE 1.21
6 REM FOR THE PLUS/4
10 POKE64784, 192
20 PR INTCHR $ (147 ): PR INT' MO TOR CONTROL PROGRAM"
30 PRINT" ":PRINT"PUSH THE PB6 PUSH BUTTON TO START MOTOR"
40 PR INT" ":PR INT" PUSH THE PB7 PUSH BUTTON TO STOP MO TOR"
50 A=PEEK(64784)
60 IF A = 65 THEN GOTO 200
70 IF A = 128 THEN GOTO 210
80 GOTO 50
100 PRlNTPEEK(S4784):GOTOt00
200 POKE 64784. t92 :PR INT"MOTOR ON": GOTO 50
210 POKE 647B4,193:PRINT"MOTOR OFF": GOTO 50
```

Program 1-9. This program controls the elementary dc motor-control circuit of Fig. 1-21 with a PLUS/4 computer.

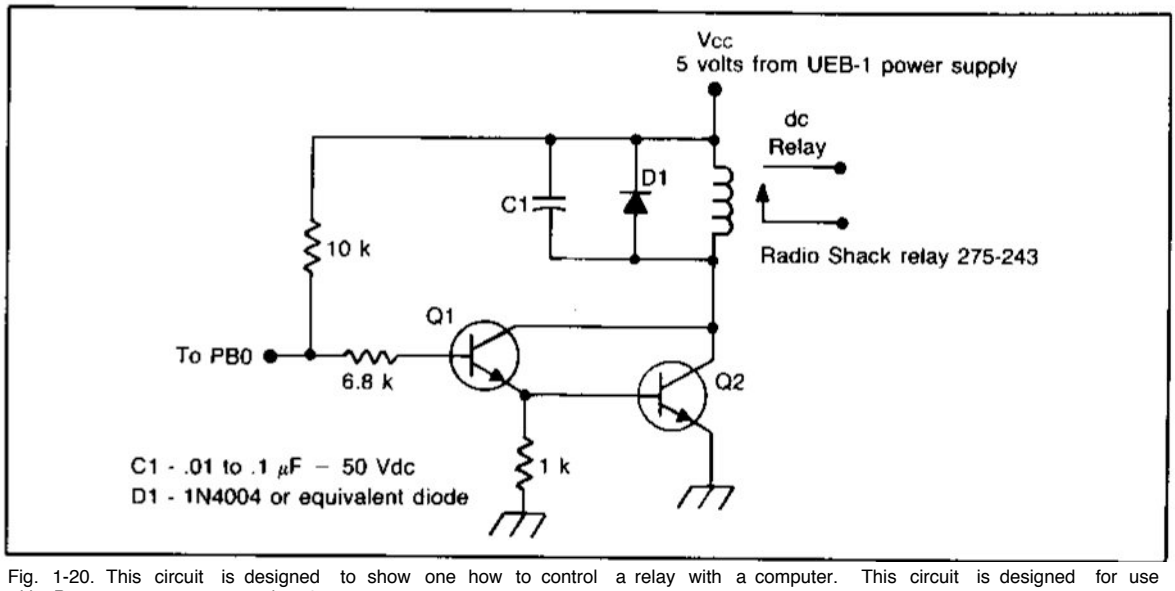


Fig. 1-20. This circuit is designed to show one how to control a relay with a computer. This circuit is designed for use with Programs 1-4, 1-5, and 1-6.

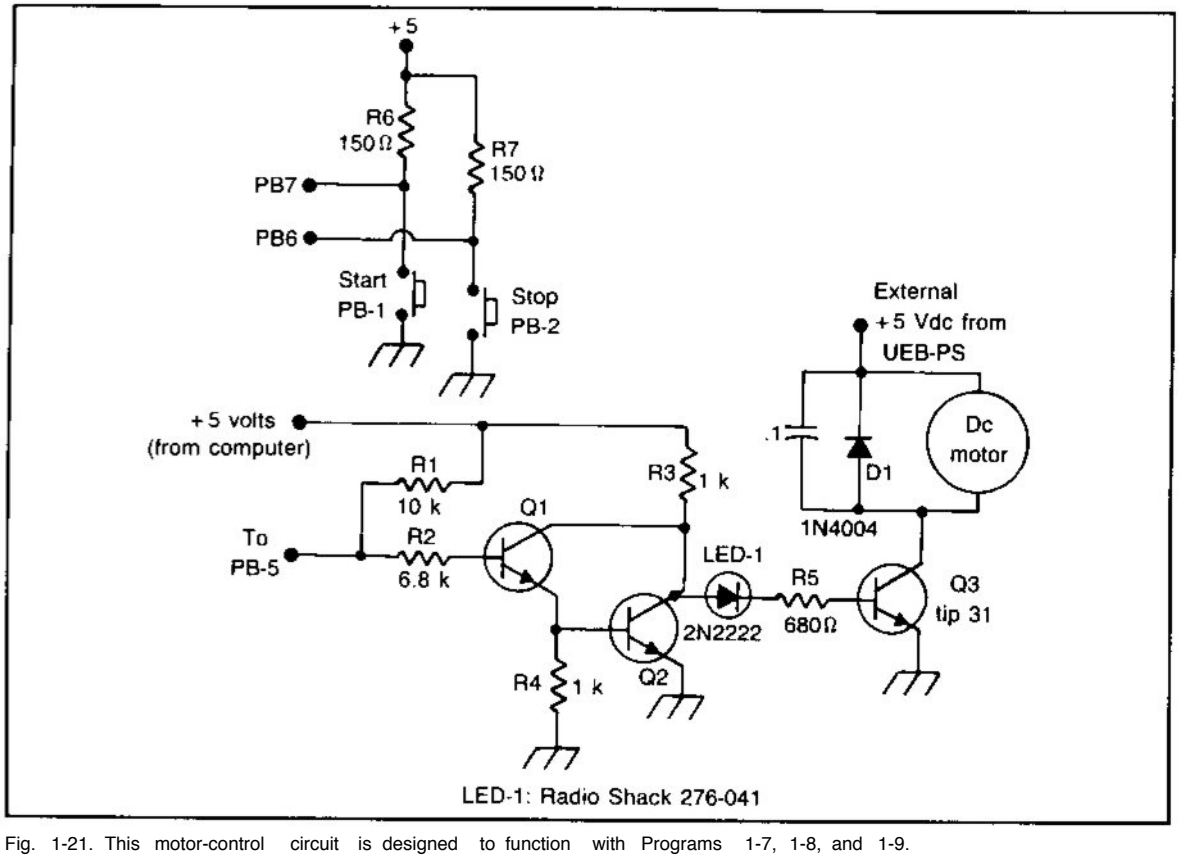


Fig. 1-21. This motor-control circuit is designed to function with Programs 1-7, 1-8, and 1-9.

```
5 REM PROGRAM 1.11"1FIGURE 1.22
6 REM FOR THE C-64
Ie POKE56579,63:POKE56577,00
21"1 A=0
313 B=PEEK(56577)
41"1IF B <128 THEN GOTO 31"1
56
60 IF A = 1"1THEN GOTO 80
71"1IF A = 1 THEN GOTO 91"1
81"1 POKE 56577,63:A=1:GOTOII"I0
90 POKE 56577,01"1:A=0:GOT0101"1
101"1FOR 1=0T0500' NEXT :GOT030
```

Program 1-10. This C-64 program is used to toggle the two LEOS of Fig. 1-22 every time the pushbutton is pressed.

```
5 REM PROGRAM 1.12 FIGURE 1.22
6 REM FOR PLUS/4
Ie POKE64784, 192
21"1A=0
31"1B=PEEK(64784)
41"1IF B <128 THEN GOTO 31"1
50
60 IF A = 1"1 THEN GOTO 80
71'1IF A = I THEN GOTO 90
80 POKE 64784,255:A=1:GOT0100
90 POKE 64784,192:A=0:GOT0100
100 FOR 1=0T0500: NEXT :GOT031'1
```

Program 1-12. This PLUS/4 program is used to toggle the two LEOs of Fig. 1-22 every time the pushbutton is pressed.

Line 10 of Programs 1-7, 1-8, and 1-9 initiates the USER PORT I/O set-up by making port lines PB6 and PB7 input lines and port lines PBO to PB5 output lines. Port line PBO is set to a logic ONE so the motor is turned off at the start. Using a logic ONE to turn the motor off makes sure the motor is off when the computer is turned on and the I/O port is not set-up. Lines 20 to 40 tell you which pushbutton must be pushed to turn the motor on or off. Lines 50, 60, 70, and 80 check to see if a pushbutton has been pressed and if so, what button was pressed. Line 100 is not part of the actual program, but if you 'RUN 100', you can watch what data is generated when a pushbutton is pressed. Lines 200 and 210 are used to POKE the correct data into the USER PORT to turn the motor on or off and also print on the video screen telling you if the motor is on or off.

```
5 REM PROGRAM I. II FIGURE 1.22
6 REM FOR VIC-21"1
Ie POKE37138,63:POKE37136,1"I0
21"1A=0
31"1B=PEEK (37136 )
41"1IF B <128 THEN GOTO 31"1
56
61"1IF A = 1"1 THEN GOTO 81"1
71"1IF A = 1 THEN GOTO 90
81"1 POKE 37136,63:A=I:GOTOII"II"I
91"1POKE 37136,1"I0:A=0'GOT0100
t01"1FOR 1=0T051"11"1:NEXT :GOT031"1
```

Program 1-11. This VIC-20 program is used 10 toggle the two LEOs of Fig. 1-22 every time the pushbutton is pressed.

Programs 1-10, 1-11, and 1-12 are used to control the operation of the circuit in Fig. 1-22. The purpose of the circuit of Fig. 1-22 is simply to show you how LEDs can be turned on and off under different logic conditions. When PB3 and PBS are set to a logic ONE, LED-1 will turn off and LED-2 will turn on. If PB3 and PB5 are set to a logic ZERO, LED-1 will turn on and LED-2 will turn off. The programs along with the pushbutton PB-1 are used to toggle the two LEDs off and on every time the push PB-l is pressed.

Programs 1-13,1-14, and 1-15 and the switch circuit of Fig. 1-23 are used to demonstrate how eight bits of numerical data can be read by an input port. In Fig. 1-23, and eight position 16 pin DIP switch is used to place a logic ONE or ZERO on each one of the 110 lines of the USER PORT. The binary data that is set up by switch Sl is read and displayed in a video presentation by the program that you use. These three programs do not contain a program line to set up the USER PORT because all USER PORT lines are initialized as input lines when the computer is turned on. No other set-up data is needed if no output lines are required. The three programs are written to read the input data of the USER PORT and produce a video display that shows the binary number, the hexadecimal number, and the decimal number that represents the logic state of the port lines (ONES or ZEROS).

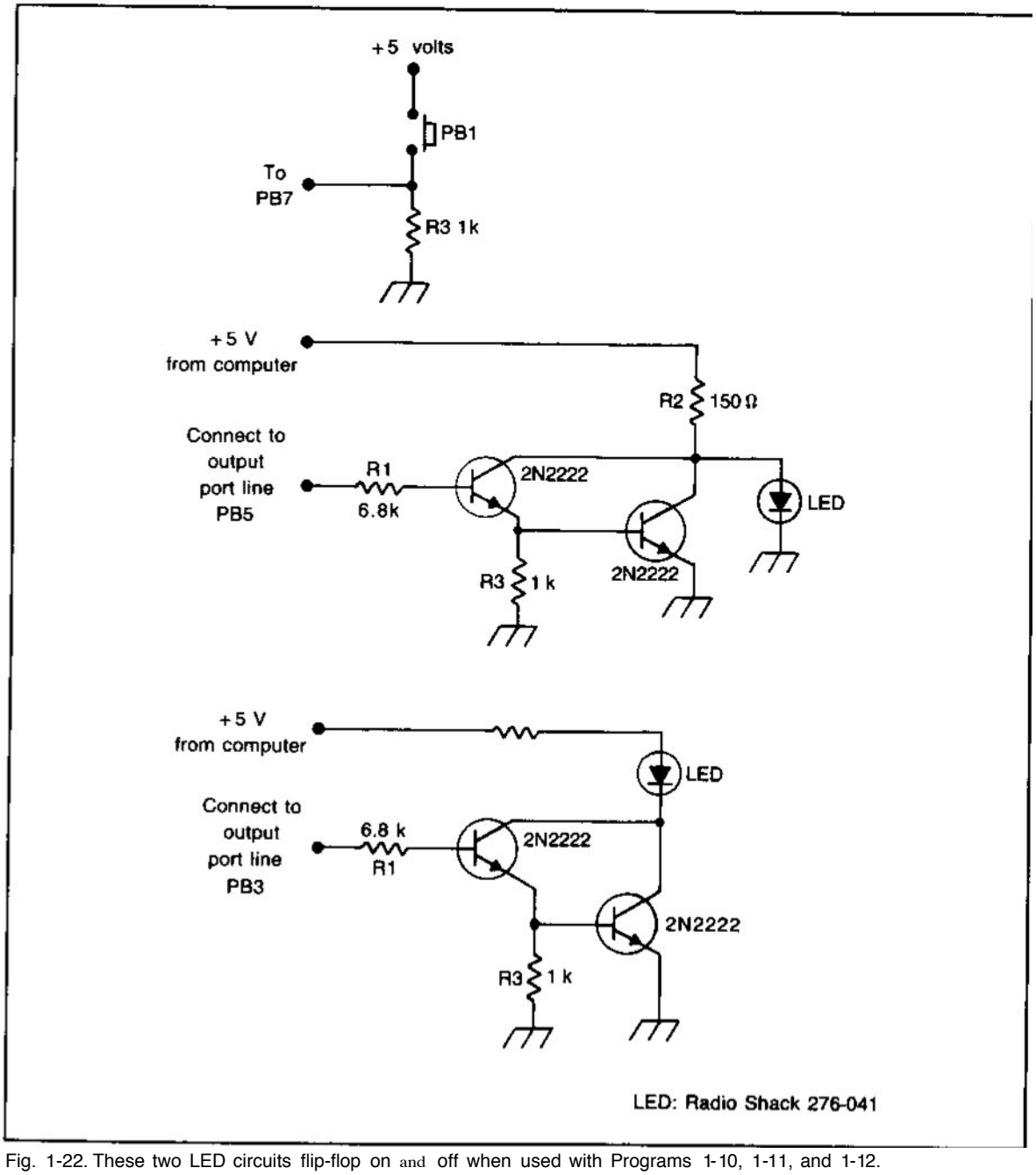


Fig. 1-22. These two LED circuits flip-flop on and off when used with Programs 1-10, 1-11, and 1-12.

Anytime a switch is changed, the programs will update the video display with the correct numerical data. When this experiment is performed, it is very easy to understand why you can only use decimal numbers between 0 and 255 in your BASIC program POKE commands.

Programs 1-16, 1-17, and 1-18 are used with the LED display circuit of Fig. 1-24. The two

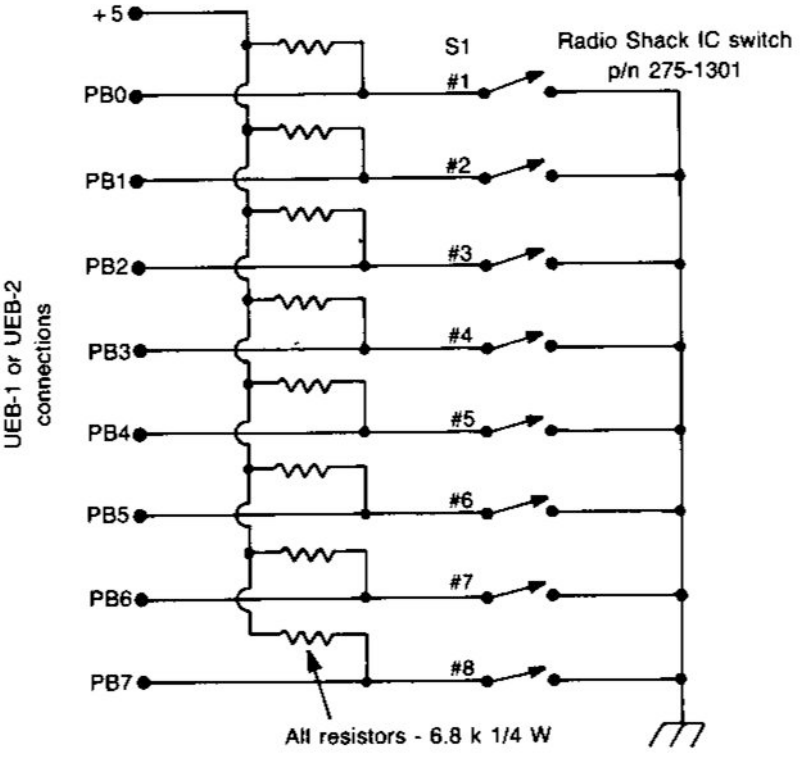


Fig. 1-23. This switch circuit can be used to input an eight-bit binary number into the USER PORT using Programs 1-13, 1-14, and 1-15.

```
5 REM PGM 1_13  FIGURE  1.23
6 REM FOR THE C-64
10 DIM A$(20):DIM  B$(20)
15 E=56577
20 D=PEEK(E)
40 W=INT(D/I6):X=W*16:Y=O-X
400 B$(0)="0":B$(I)='I':B$(2)="2":B$(3)="3":B$(4)="4":B$(5)="5"1B$(6)='6'
405 B$(7)='7'IB$(8)='8':B$(9)='9"
407 B$(10)="A":B$(11)="B"
410 B$(12)='C":B$(13)='D'1B$(I4)="E':B$(t5)='F"
500 A$(0)="0000':A$(1)='0001":A$(2)="0010':A$(3)="0011"1A$(4)="0100"
510 A$(5)="0101":A$(6)="0110":A$(7)="0111":A$(0)="1000":A$(10'="1001"
520 A$(10,="1010":A$(t1'="1011":A$(12)="1100":A$(13)='1101":A$(14)="1110"
530 A$(15)="1111"
600 PRINTCHR$(147)
610 PRINT"    THE DATA  IN ADDRESS  ";1PRINTE
620 PRINT"  ":PRINT"  ";:
630 PRINTA$(W);:PRINT'   ",:PRINTA$(Y);:PRINT"  -
900 PRINTD' DEC  -$   "B$(W)B$(Y)' HEX':GOTOI5
```

Program 1-13. This program displays the digital input data at the C-64 USER PORT, which is set-up by the switch circuit of Fig. 1-23.

```
5 REM PGM 1.14 FIGURE 1.23
6 REM FOR THE V IC-20
10 DIM A$(20):OIM B$(20)
15 E=37136
20D=PEEK(EI
40 W=INT(O/16):X=W*16:Y=0-X
400 B$(0)="0':B$(I)="1":B$(2)="2":B$(3)="3":B$(4)='4":B$(5'."5":S$(6)="S"
405 B$(7'='7":B$(8'="8":B$(9'="9"
407 B$(10)="A":B$(I1)="B"
410 B$( 12)="C" :8$( 13'= '0":B$( 14 )='E' :8$( 15 )="F"
500 A$(0)="0000":A$(I,="0001":A$(2)="0010":A$(3)="0011"IA$(4)="0100'
510 A$(S'="010I':A$(6)="0110":A$(7)="0111":A$(8)="1000":A$(10'="1001"
520 A$(10'="1010":A$(11)="1011":A$(12)="1100':A$(13'.'1101"IA$(14)="1110"
530 A$(15)="1III"
600 PRINTCHR$(147'
6 10 PR I NT" THE DATA IN ADDRESS' :PR INTE
S20 PR 1tN" "
631'1PRINTA$(W); :PRINT' "; :PRINTA$(Y):PRINT" "
900 PRINTO"OEC - $ "B$(W'B$(Y)" HEX':GOT015
```

Program 1-14. This program displays the digital input data at the VIC-20 USER PORT, which is set-up by the switch circuit of Fig. 1-23.

MC14584 CMOS IC circuits of Fig, 1-24 are being used to drive the LED display Ie. The three programs are written to make the LED display count from ZERO to NINE and then reset and start over again. Line ten of each program is used to set-up the USER PORT as an output port since all eight lines are required to drive the LED display circuit. Other technical data about the functioning of these three programs and Fig. 1-24is given in Table 1-12.

Programs 1-19 and 1-20 are used with Figs.

```
5 REl PROGRAM 1.15 FIGURE 1.23
6 REM FOR THE PLU5/4
10 DIM A$(20):DIM B$(20)
15 E=64784
2121D=PEEK(E)
40 W=lNT(D/16) 'X=W*16' Y=D-X
400 B$(0)= "121:e$( 1 I="1":B$(2)= "2":B$(3) ."3": B$(4)= "4":B$(S)= "5":B$(6'="6"
405 B$(7)="7":8$(8)="8":B$(9)="9"
407B$(10)="A":B$(II)="B"
41121B$( 12) = 'C" :B$( 13)="0" :B$( 14)="E' :B$( 15)="F"
50121A$(0)="0000":A$(11=·0001":A$(2)="0010":A$(3)="0011":A$(4)="0100"
510 A$(5)="0101":A$(6)="0tI0":A$(7)="0111":A$(8'="1000":A$(1121'="1001'
520 A$(10)="1010":A$(II)="1011"lA$(12'="tt00":A$(13'="110t':A$(14'="1110"
530 A$(I5'="1111"
600 PRINTCHR$(147)
610 PRINT" THE DATA IN ADDRESS "; :PRINTE
620 PRINT" ":PRINT" ",:
630 PRINTA$(W1; :PRINT" ":PRINTA$(Y,; :PRINT' _ "':
90121PR INTD" DEC - $ "8$(W)B$(Y)" HEX": GOTO 15
```

Program 1-15. This program displays the digital input data at the PLUS/4 USER PORT, which is set-up by the switch circuit of Fig. 1-23.

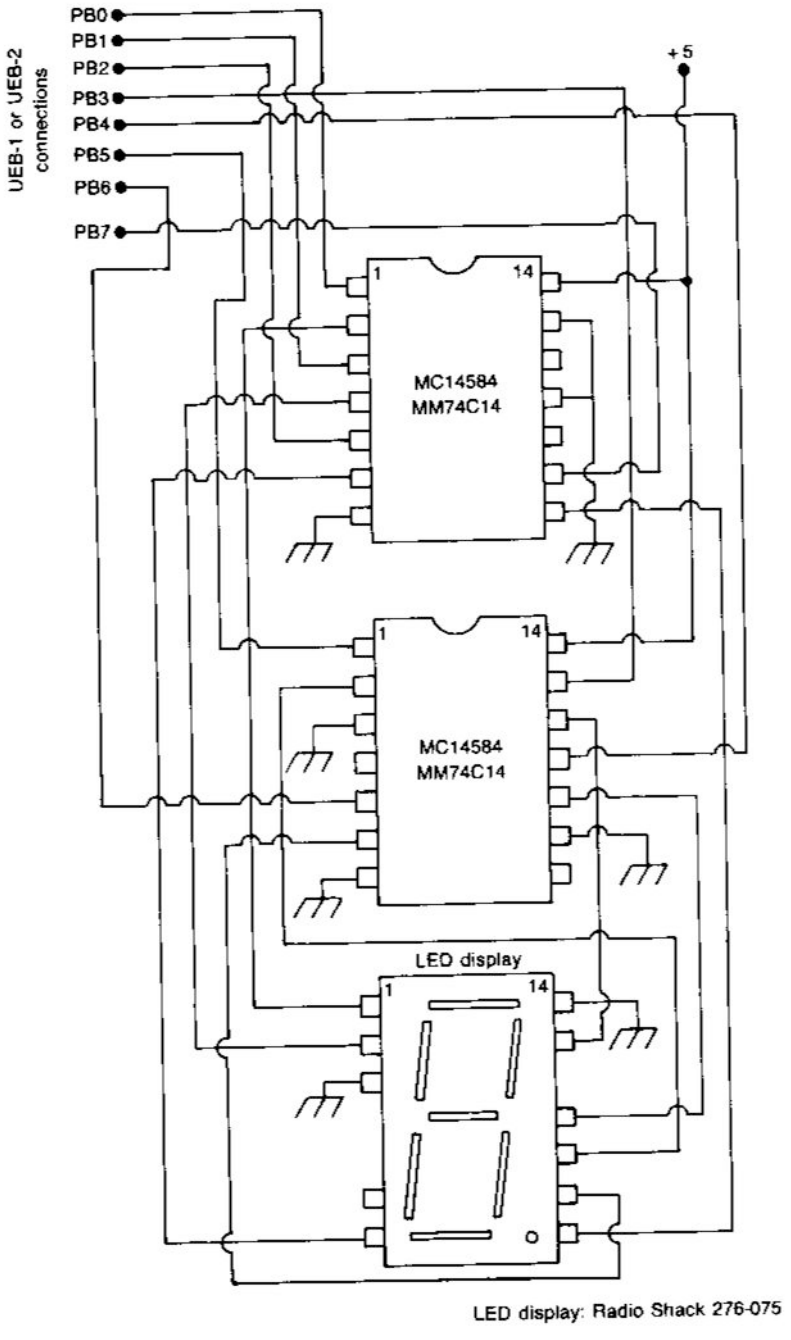
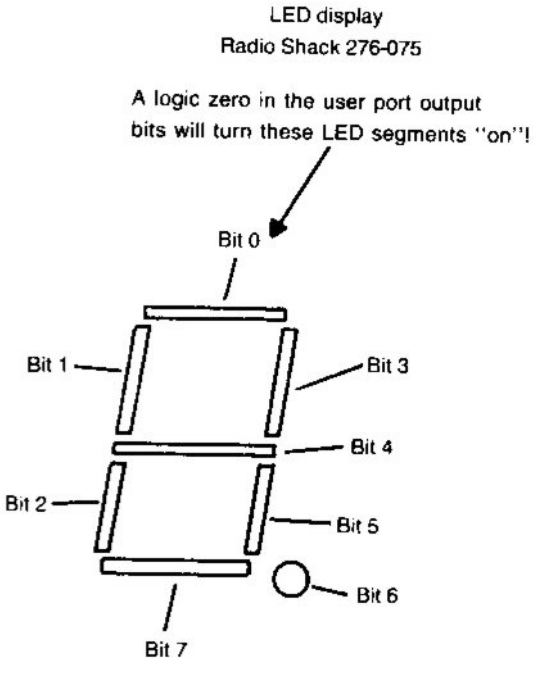


Fig. 1-24. This LED display circuit will count 0 through 9 when used with Programs 1-16, 1-17, and 1-18.

Table 1-12. AddItional Supporting Data for Fig. 1-24.



Number Generation Data

Display	Number-Poke Value
Blank	255
Dot-	·191
1	215
2	98
3	70
4	197
5	76
6	72
7	214
8	64
9	68
0	80

Display Segment-Poke Value Needed to Turn on A LED Segment

Bit 0	= 254
Bit 1	= 253
Bit 2	= 251
Bit 3	= 247
Bit 4	= 239
Bit 5	= 233
Bit 6	= 191
Bit 7	= 127

Note: This data is for Fig. 1-24 only

1-25 and 1-26 to show the functioning of the *A/D* converters in the VIC-20 and the C-64. The PLUS/4 computer is not used in these experiments. Programs 1-19 and 1-20 simplyread the memory location that is assigned to the internal *AID* converter circuit and then displays this data as a decimal number between 0 an 255 on the video screen. Figure 1-25 is a potentiometer circuit and the decimal

```
5 REM PGM 1.IS-FIGURE 1.24-FOR
  THE C-S4
t0 POKE 56S79,255:A=56577
20 POKE A,191:GOSUB500
30 POKE A,215:GOSUB500
40 POKE A,98:GOSUB500
50 POKE A,70:GOSUB500
60 POKE A,197:GOSUB500
70 POKE A,76:GOSUB500
80 POKE A,72:GOSUB500
90 POKE A,214:GOSUB500
100 POKE A,64:GOSUB500
110 POKE A,68:GOSUB500
120 POKE A,80:GOSUB500
130 GOT020
500 FOR 1=1 TO 1000:NEXT:RETURN
```

Program 1-16. This program controls the LED display of Fig. 1-24 with a C-64. When the program runs, the LED display will continually count 0 through 9 until the program execution is halted.

number that is shown on the video screen is a number that corresponds to the positional setting of the pot. Figure 1-26 uses a Cds photocell (light sensitive resistor) in a light-level sensing application. When this circuit is used, the number that is displayed on the video screen is a representation of the level of light in the computer room.

```
5 REM PGM 1.17-FIGURE 1_24-FOR
  THE VIC-20
10 POKE 37138,255:A=37136
20 POKE A,191:GOSUB500
30 POKE A,21S:GOSUB500
40 POKE A,98:GOSUB500
50 POKE A,70:GOSUB500
60 POKE A,197:GOSUB500
70 POKE A,76:GOSUB500
80 POKE A,72:GOSUB500
90 POKE A,214:GOSUB500
100 POKE A,64:GOSUB500
110 POKE A,68:GOSUB500
120 POKE A,80:GOSUB500
130 GOT020
S00 FOR 1=1 TO 1000:NEXT:RETURN
```

Program 1-17. This program controls the LED display of Fig. 1-24 with a VIC-20. When the program runs, the LED display will continually count 0 through 9 until the program execution is halted.

```
5 REM PGM I.IS-FIGURE 1.24-FOR
  THE PLUS/4
10 POKE 64874,000:A=64784
20 POKE A,191:GOSUB500
30 POKE A,215:GOSUB500
40 POKE A,98:GOSUB500
50 POKE A,70:GOSUB500
60 POKE A,197:GOSUB500
70 POKE A,76:GOSUB500
80 POKE A,72'GOSUB500
90 POKE A,214'GOSUB500
100 POKE A,64'GOSUB500
110 POKE A,68:GOSUB500
120 POKE A,80:GOSUB5ee
130 GOT020
500 FOR 1=1 TO 1000:NEXT'RETURN
```

Program 1-18. This program is used to control the LED display of Fig. 1-24 with a PLUS/4. When the program runs, the LED display will continually count 0 through 9 until the program execution is halted.

```
Ie REM PROGRAM 1.19 F!GURES 1.25
   AND 1.2S
15 REM FOR THE C-64
20 B=0
100 FOR I=1 TO 50
110 A = PEEK(54297)
120 B=B+A
130 NEXT I
140 C=B/50
150 PRINTCHR$(147)
160 PRINT INT(C)
170 GOT010
```

Program 1-19. This program demonstrates the built in *NO* converter in the C-64, using the pot circuit of Fig_ 1-25 or the light-sensor circuit of Fig. 1-26.

```
10 REM PROGRAM t.20 FIGURES 1.25 AND
   1.26
15 REM FOR THE VIC-20
20 B=0
100 FOR 1=1 TO 50
110 A = PEEK(3S872)
120 B=B+A
130 NEXT I
140 C=B/50
150 PRINTCHR$(147)
160 PRINT INT(C)
170 GOT010
```

Program 1-20. This program demonstrates the built in *NO* converter in the VIC-20, using the pot circuit of Fig. 1-25 or the light-sensor circuit of Fig. 1-26.

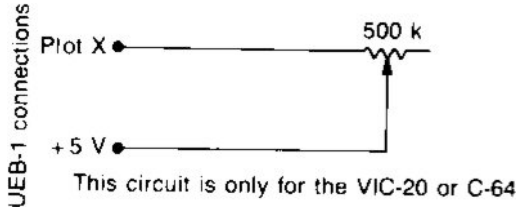


Fig. 1-25. This potentiometer circuit is designed to show the capabilities of the on board *NO* converter in the C-64 or the VIC-2O using Programs 1-19 or 1-20.

Remember that these programs and circuits use the joy stick ports on the computers.

Program 1-21 is the last program in this chapter and it is used with the light level LED display circuit of Fig. 1-27. This experiment is really a combination of Figs. 1-24 and 1-26 and Programs 1-16 and 1-19. The result of the addition of these programs and circuits to each other is a light-level meter that generates a light level reading between 0 and 9. The resistor and capacitor combination of Rl and C1 is used to adjust the linearity of the light meter. The actual values of these two parts will depend on the light sensing characteristics of the Cds photocell. **It** the display readings seem to be either all in the low-light level or high-light level area, try adjusting one of the two components one way or the other. At night, this circuit has enough sensitivity to detect the emitted light beam from a 6-volt flashlight that is 100 yards away.

```
 5 REM PROGRAM 1.21-FIGURE 1,27-FOR
   C-64
 6 PRINTCHR$(147):PRINT"PROGRAM
   RUNNING - CHECK LED DISPLAY"
11'1POKE 56579,255
20 POKE 56577,191:GOTO 51'11'1
31'1POKE 56577,215:GOTO 51'11'1
41'1POKE 56577,98:GOTO 501'1
51'1POKE 56577,70:GOTO 51'10
60 POKE 56577,197:GOTO 501'1
70 POKE 56577,76:GOTO 560
80 POKE 56577,72:GOTO 561'1
90 POKE 56577,214:GOTO 500
11'10POKE 56577,64:GOTO 501'1
111'1POKE 56577,6B:GOTO 500
121'1POKE56577,8a:GOTO 51'10
51'11B=1a:A=a
511'1FOR I=I TO 51'1
515 A=PEEK(54297)
521'1B=B+A
530 NEXT I
540 C=B/5a
550 C=INT(C)
560 D=C/28:E=INT(D)
570 IF E=a THEN GOTO 111'1
580 IF E=I THEN GOTO 11'11'1
590 IF E=2 THEN GOTO 90
600 IF E=3 THEN GOTO 80
611'1IF E=4 THEN GOTO 70
620 IF E=5 THEN GOTO 60
630 IF E=8 THEN GOTO 30
640 IF E=9 THEN GOTO 121'1
650 GOT0500
```

Program 1-21. This program makes an elementary light-level meter, using the LED display and light-sensor circuit of Fig. 1-27.

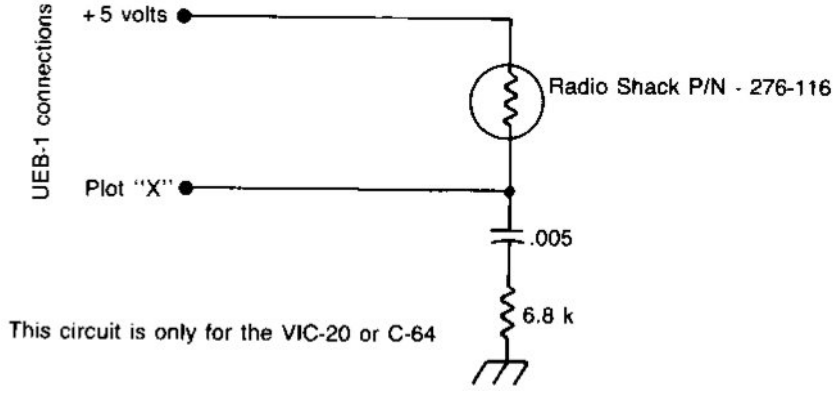


Fig. 1-26. This circuit can be used as a relative light-level meter with Programs 1-19 or 1-20.

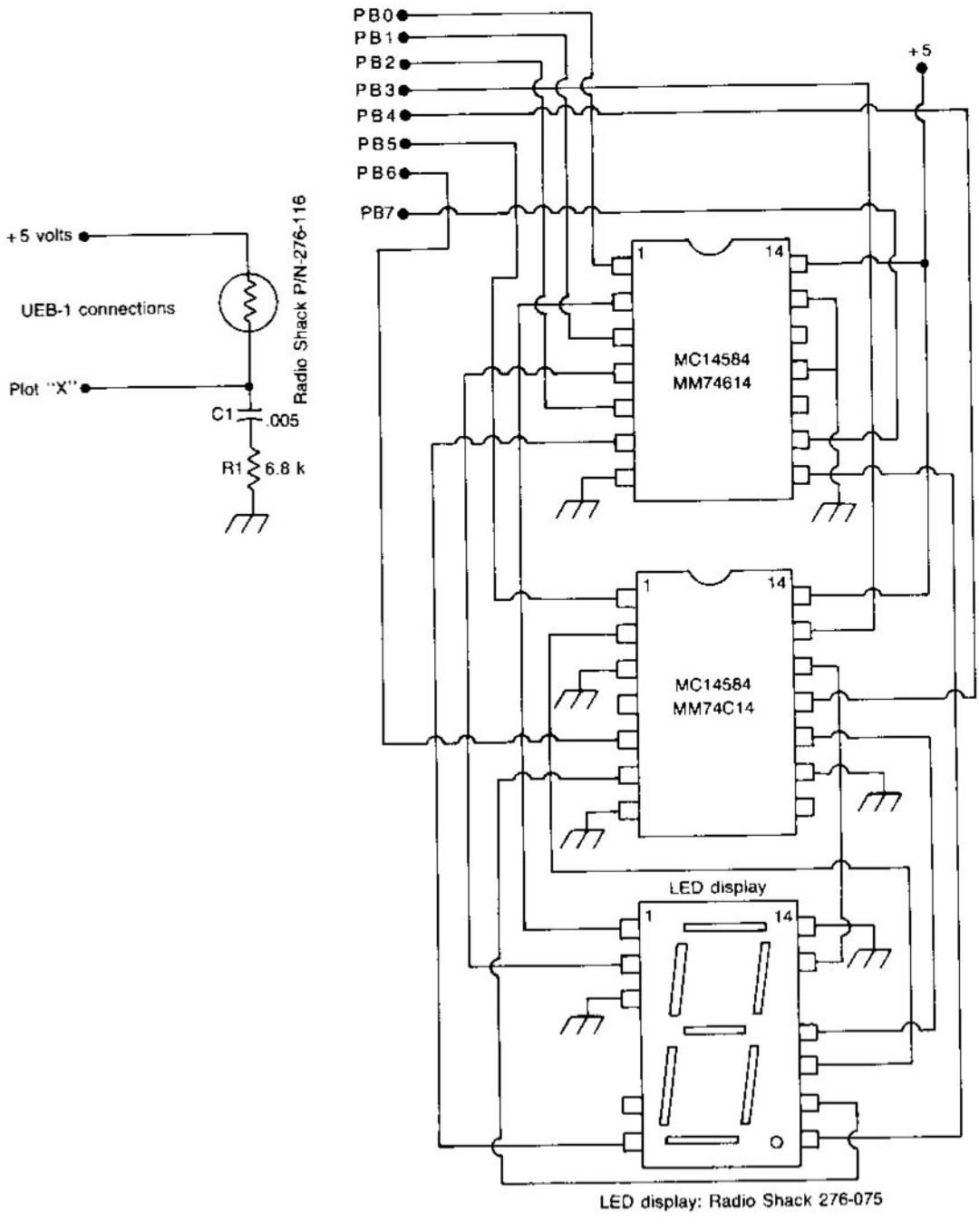
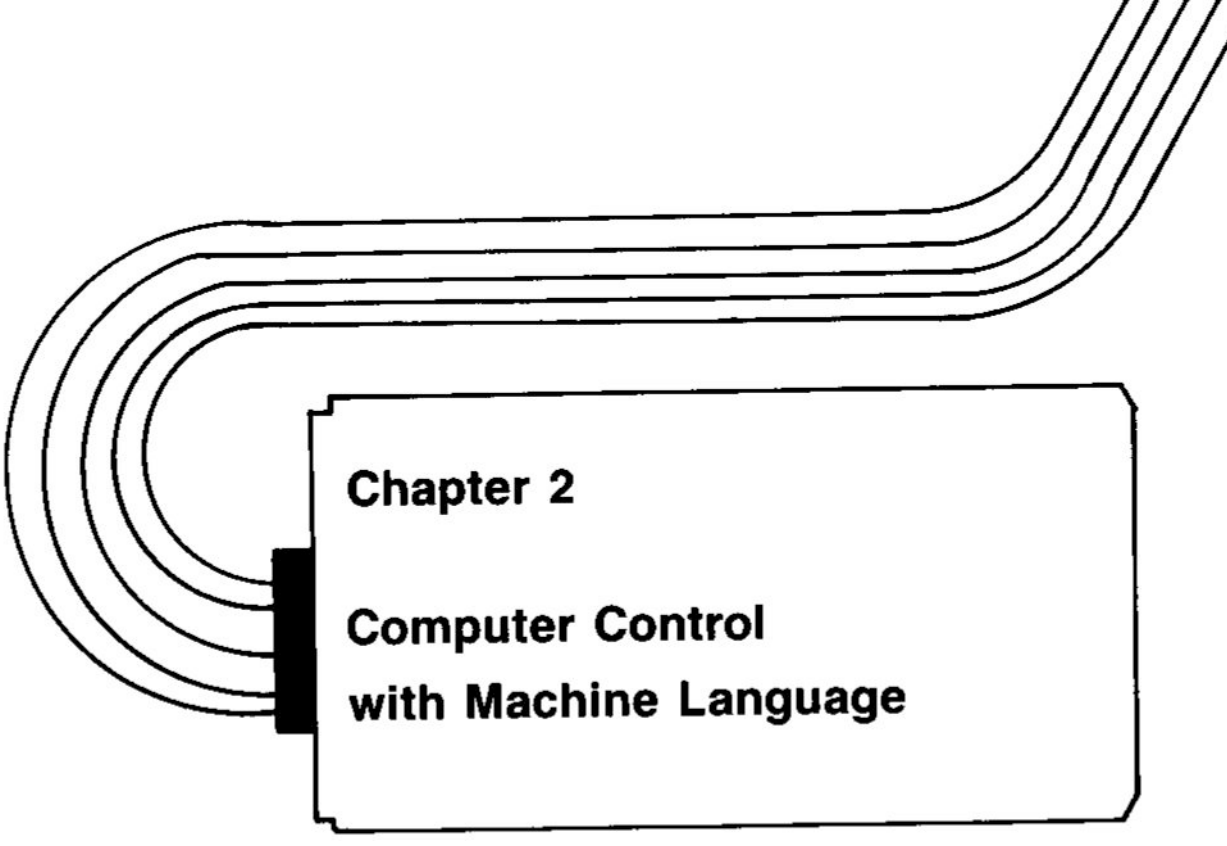


Fig. 1-27. This LED display and photocell circuit is designed to function as a light-level meter when used with Program 1-21.

CONCLUSION

In Chapter 1, you have been introduced to a few elementary computer interfacing techniques to show how easy it can be to use your Commodore computer as a controller for either electronic technology or science experiments, After you have built and run all of the experimental circuits and programs in this chapter, you will have no trouble understanding any of the remaining projects in this book. You should also have a good idea about how easy it would be to control even the most sophisticated experiments with a Commodore computer.



Chapter 2

Computer Control with Machine Language

IN CHAPTER 1, WE PRESENTED A SERIES OF EXperimenter's circuits and programs. All of the programs were written in the BASIC language that all four of the Commodore computers use. You can do a great deal with BASIC because it is a very powerfullanguage. This BASIC language does fall short in one area, though, and that area is speed of operation. **In** very general terms, one could say that it takes "about" 1 millisecond to perform the average BASIC command. This means that if you would PEEK the USER PORT memory location to see if a pushbutton has been pressed, it would take around one millisecond to do that pushbutton check. That would be no problem because the pushbutton will be held down for a second or so most likely. What happens if you are trying to detect a logic pulse that is only .1 milliseconds long? The chances are that the PEEK command would miss the pulse when it came along because the BASIC program would be doing some of its busy work and not checking the USER PORT input line when the pulse appeared. This problem can be easily handled with the machine-language control. The main purpose of this chapter is to rerun one of the Chapter 1 experiments using 6502 machine language and not the standard BASIC language. We will use BASIC as the supporting program, but all of the input and output functions will be done with machine-language subroutines that are part of the main BASIC program, After you have completed this chapter, you will be able to use the powerful BASIC language and still perform any tasks that require lightning fast speed.

Most people have a very hard time understanding machine language. **If** you limit the scope of your machine-language routines to only include the machine instruction codes that are required for control functions, your introduction to machine language will be much easier. In this chapter, we will just use 11 of 6502 machine-language instruction codes (opcodes), A series of machine-language subroutines using these opcodes will be presented that can be substituted for the BASIC PEEK and POKE routines that were used to control the ex-

periments in chapter 1.

THE MACHINE-LANGUAGE MONITOR

A machine-language monitor program is a programming aid type of program that lets you talk to your computer at its own machine-language level. The main function of the monitor program is to let you examine and change various memory locations in the computer, insert machine-language subroutines, and test out these subroutines. Most monitor programs contain some form of a mini-assembler and disassembler that lets you look at any area of the computers memory and a number of monitor commands that lets you perform various machine-language functions. I would strongly suggest that you visit your local computer store and purchase a monitor program for your computer if you do not have one. (Note: The C-16 and the PLUS/4 have their own built-in monitor programs.)

If you do not own a monitor program, you can use Program 2-1 to examine any area of memory in your computer. This program will display the memory contents as shown in Fig. 2-1. This figure displays the 16 highest memory locations in the PLUS/4 memory map. The program first asks you to input the address of the memory location where you want to start the display. The program then prints out the video display line by line. Each line starts out by presenting the decimal address location. Following the address location is a combination of eight ONES or ZEROs that corresponds to the binary data contained in that eight bit memory byte. Following the binary data is the decimal and hexadecimal number equivalent for the binary data. The program display will continue until you press the RUN/STOP key.

All monitor program that you can buy for your computer use the hexadecimal number system. The hexadecimal number system is used in machine language work because it can represent the eight bit computer byte very easily. There are many books and magazine articles on this number system, and so we will not present any long explanations. The data that is presented in Tables 2-1 and 2-2 will give you all of the hexadecimal information that is needed for this book. After studying these two tables, you will understand the funny number data that is displayed by the monitor programs. The

```
INPUT DECIMAL MEMORY ADDRESS ? 65520
 65520  -0000 0100-    4 DEC-$04 HEX
 65521  -0000 0000-    0 DEC-$00 HEX
 65522  -0000 0000-    0 DEC-$00 HEX
 65523  -0000 0000-    0 DEC-$00 HEX
 65524  -0000 0000-    0 DEC-$00 HEX
 65525  -0000 0000-    0 DEC-$00 HEX
 65526  -1000 1101-  141 DEC-$8D HEX
 65527  -0011 1110-   62 DEC-$3E HEX
 65528  -1111 1111-  255 DEC-$FF HEX
 65529  -0100 1100-   76 DEC-$4C HEX
 65530  -1010 0100-  164 DEC-$A4 HEX
 65531  -1111 0010-  242 DEC-$F2 HEX
 65532  -1111 0110-  246 DEC-$F6 HEX
 65533  -1111 1111-  255 DEC-$FF HEX
 65534  -0000 0000-    0 DEC-$00 HEX
 65535  -0100 0100-   68 DEC-$44 HEX
```

Fig. 2-1. This is the video display that is generated by the memory display Program 2-1. This figure shows the top 16 memory locations of the PLUS/4 computer after running this program.

```
5 REM  PROGRAM 2.1  MEMORY DISPLAY
10 DIM A$(20):DIM B$(20):PRINTCHR$(147)
15 INPUT"INPUT DECIMAL MEMORY ADDRESS ";E:PRINT" "
20 D=PEEK(E)
40 W=INT(D/16):X=W*16:Y=D-X
400 B$(0)="0":B$(1)="1":B$(2)="2":B$(3)="3":B$(4)="4":B$(5)="5":B$(6)="6"
405 B$(7)="7":B$(8)="8":B$(9)="9"
407 B$(10)="A":B$(11)="B"
410 B$(12)="C":B$(13)="D":B$(14)="E":B$(15)="F"
500 A$(0)="0000":A$(1)="0001":A$(2)="0010":A$(3)="0011":A$(4)="0100"
510 A$(5)="0101":A$(6)="0110":A$(7)="0111":A$(8)="1000":A$(10)="1001"
520 A$(10)="1010":A$(11)="1011":A$(12)="1100":A$(13)="1101":A$(14)="1110"
530 A$(15)="1111"
600 PRINTE;:PRINT"-";:
630 PRINTA$(W);:PRINT" ";:PRINTA$(Y);:PRINT"-";:
900 PRINTD"DEC-$"B$(W)B$(Y)" HEX"
950 E=E+1 : GOTO20
```

Program 2-1. This program can be used to display the memory contents of your computer as shown in Table 2-1.

numbering display of Table 2-2 was generated by Program 2-2. For those of you who do not own a monitor program at this time, Programs 2-3 and 2-4 are two mini-monitors for the VIC-20 and C-64 that can be used to enter any program in this chapter using the information in Tables 2-3 and 2-4.

A SHORT MACHINE-LANGUAGE PROGRAM

The best way to learn how to use machine language is to use it to do something. Three short machine-language program experiments and their supporting instructions will now be presented to introduce you to the high speed world of machine language. These experiments are written to control the simple pushbutton/LED circuit of Fig. 2-2. The main point of these experiments is to learn how to divide the computer memory into a BASIC area and a machine-language area, enter a machine-language routine, and run a BASIC program and a machine-language subroutine together using the BASIC's SYS command. One may think that this is a lot of trouble to go through to just turn a LED on and off, but you must start with machine language somewhere and in this book this is where it starts. Understanding how a BASIC program and a machine-language subroutine operates together is a very important part of control system programming. In the experiments, the machine-language subroutine program data will be presented in the disassembly format that is common to all of the

Table 2-1. Some of the More Important Computer Address Locations In Decimal, Hexadecimal, and Binary.

Decimal #	— Hexadecimal $ —	Binary Number - - - - - - - - - -	K of Memory
0	0	0000 0000 0000 0000	
1	1	0000 0000 0000 0001	
16	10	0000 0000 0001 0000	
255	FF	0000 0000 1111 1111	
1023	3FF	0000 0011 1111 1111	
1024	400	0000 0100 0000 0000	1K
4096	1000	0001 0000 0000 0000	4K
8192	2000	0010 0000 0000 0000	8K
16384	4000	0100 0000 0000 0000	16K
32768	8000	1000 0000 0000 0000	32K
65535	FFFF	1111 1111 1111 1111	65K

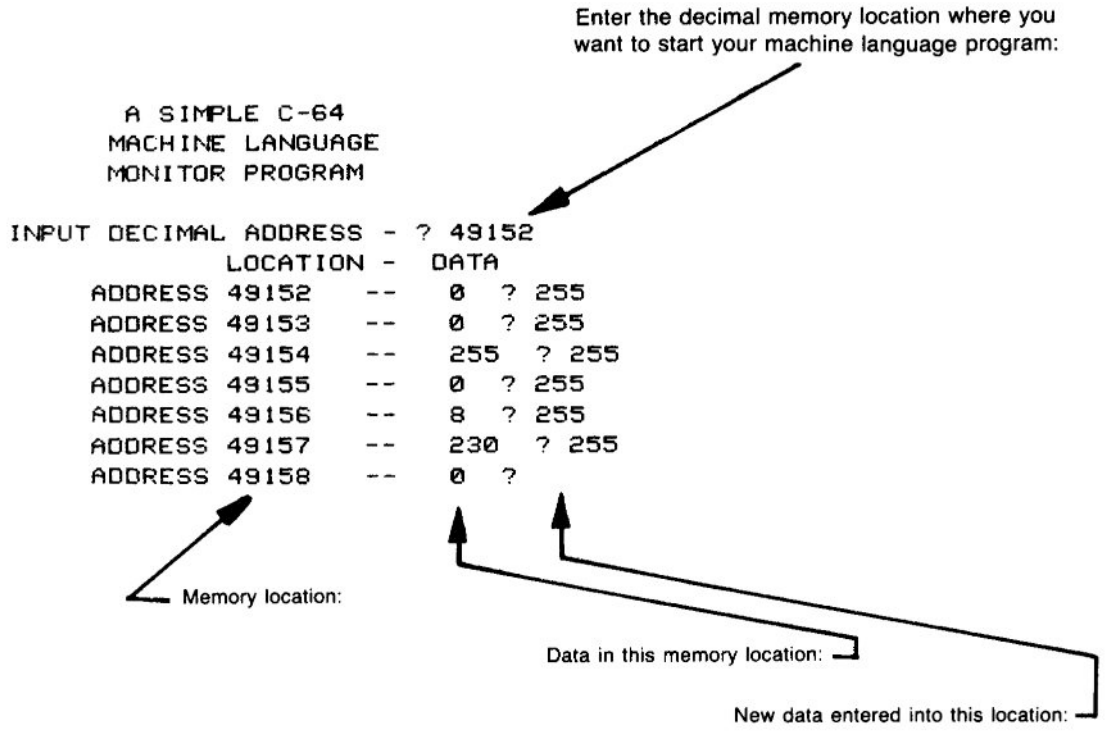
Table 2-2. Decimal, Hexadecimal, and Binary Numbers
from 0 to 255. You Can Use this Table to Look-up Binary Bit Patterns and Poke Data for Control Programs.

DECIMAL	0	=	HEX $ 00	=	0000 0000	DECIMAL	57	=	HEX $ 39	=	0011 1001
DECIMAL	1	=	HEX $ 01	=	0000 0001	DECIMAL	58	=	HEX $ 3A	=	0011 1010
DECIMAL	2	=	HEX $ 02	=	0000 0010	DECIMAL	59	=	HEX $ 3B	=	0011 1011
DECIMAL	3	=	HEX $ 03	=	0000 0011	DECIMAL	60	=	HEX $ 3C	=	0011 1100
DECIMAL	4	=	HEX $ 04	=	0000 0100	DECIMAL	61	=	HEX $ 3D	=	0011 1101
DECIMAL	5	=	HEX $ 05	=	0000 0101	DECIMAL	62	=	HEX $ 3E	=	0011 1110
DECIMAL	6	=	HEX $ 06	=	0000 0110	DECIMAL	63	=	HEX $ 3F	=	0011 1111
DECIMAL	7	=	HEX $ 07	=	0000 0111	DECIMAL	64	=	HEX $ 40	=	0100 0000
DECIMAL	8	=	HEX $ 08	=	0000 1000	DECIMAL	65	=	HEX $ 41	=	0100 0001
DECIMAL	9	=	HEX $ 09	=	0000 1001	DECIMAL	66	=	HEX $ 42	=	0100 0010
DECIMAL	10	=	HEX $ 0A	=	0000 1010	DECIMAL	67	=	HEX $ 43	=	0100 0011
DECIMAL	11	=	HEX $ 0B	=	0000 1011	DECIMAL	68	=	HEX $ 44	=	0100 0100
DECIMAL	12	=	HEX $ 0C	=	0000 1100	DECIMAL	69	=	HEX $ 45	=	0100 0101
DECIMAL	13	=	HEX $ 0D	=	0000 1101	DECIMAL	70	=	HEX $ 46	=	0100 0110
DECIMAL	14	=	HEX $ 0E	=	0000 1110	DECIMAL	71	=	HEX $ 47	=	0100 0111
DECIMAL	15	=	HEX $ 0F	=	0000 1111	DECIMAL	72	=	HEX $ 48	=	0100 1000
DECIMAL	16	=	HEX $ 10	=	0001 0000	DECIMAL	73	=	HEX $ 49	=	0100 1001
DECIMAL	17	=	HEX $ 11	=	0001 0001	DECIMAL	74	=	HEX $ 4A	=	0100 1010
DECIMAL	18	=	HEX $ 12	=	0001 0010	DECIMAL	75	=	HEX $ 4B	=	0100 1011
DECIMAL	19	=	HEX $ 13	=	0001 0011	DECIMAL	76	=	HEX $ 4C	=	0100 1100
DECIMAL	20	=	HEX $ 14	=	0001 0100	DECIMAL	77	=	HEX $ 4D	=	0100 1101
DECIMAL	21	=	HEX $ 15	=	0001 0101	DECIMAL	78	=	HEX $ 4E	=	0100 1110
DECIMAL	22	=	HEX $ 16	=	0001 0110	DECIMAL	79	=	HEX $ 4F	=	0100 1111
DECIMAL	23	=	HEX $ 17	=	0001 0111	DECIMAL	80	=	HEX $ 50	=	0101 0000
DECIMAL	24	=	HEX $ 18	=	0001 1000	DECIMAL	81	=	HEX $ 51	=	0101 0001
DECIMAL	25	=	HEX $ 19	=	0001 1001	DECIMAL	82	=	HEX $ 52	=	0101 0010
DECIMAL	26	=	HEX $ 1A	=	0001 1010	DECIMAL	83	=	HEX $ 53	=	0101 0011
DECIMAL	27	=	HEX $ 1B	=	0001 1011	DECIMAL	84	=	HEX $ 54	=	0101 0100
DECIMAL	28	=	HEX $ 1C	=	0001 1100	DECIMAL	85	=	HEX $ 55	=	0101 0101
DECIMAL	29	=	HEX $ 1D	=	0001 1101	DECIMAL	86	=	HEX $ 56	=	0101 0110
DECIMAL	30	=	HEX $ 1E	=	0001 1110	DECIMAL	87	=	HEX $ 57	=	0101 0111
DECIMAL	31	=	HEX $ 1F	=	0001 1111	DECIMAL	88	=	HEX $ 58	=	0101 1000
DECIMAL	32	=	HEX $ 20	=	0010 0000	DECIMAL	89	=	HEX $ 59	=	0101 1001
DECIMAL	33	=	HEX $ 21	=	0010 0001	DECIMAL	90	=	HEX $ 5A	=	0101 1010
DECIMAL	34	=	HEX $ 22	=	0010 0010	DECIMAL	91	=	HEX $ 5B	=	0101 1011
DECIMAL	35	=	HEX $ 23	=	0010 0011	DECIMAL	92	=	HEX $ 5C	=	0101 1100
DECIMAL	36	=	HEX $ 24	=	0010 0100	DECIMAL	93	=	HEX $ 5D	=	0101 1101
DECIMAL	37	=	HEX $ 25	=	0010 0101	DECIMAL	94	=	HEX $ 5E	=	0101 1110
DECIMAL	38	=	HEX $ 26	=	0010 0110	DECIMAL	95	=	HEX $ 5F	=	0101 1111
DECIMAL	39	=	HEX $ 27	=	0010 0111	DECIMAL	96	=	HEX $ 60	=	0110 0000
DECIMAL	40	=	HEX $ 28	=	0010 1000	DECIMAL	97	=	HEX $ 61	=	0110 0001
DECIMAL	41	=	HEX $ 29	=	0010 1001	DECIMAL	98	=	HEX $ 62	=	0110 0010
DECIMAL	42	=	HEX $ 2A	=	0010 1010	DECIMAL	99	=	HEX $ 63	=	0110 0011
DECIMAL	43	=	HEX $ 2B	=	0010 1011	DECIMAL	100	=	HEX $ 64	=	0110 0100
DECIMAL	44	=	HEX $ 2C	=	0010 1100	DECIMAL	101	=	HEX $ 65	=	0110 0101
DECIMAL	45	=	HEX $ 2D	=	0010 1101	DECIMAL	102	=	HEX $ 66	=	0110 0110
DECIMAL	46	=	HEX $ 2E	=	0010 1110	DECIMAL	103	=	HEX $ 67	=	0110 0111
DECIMAL	47	=	HEX $ 2F	=	0010 1111	DECIMAL	104	=	HEX $ 68	=	0110 1000
DECIMAL	48	=	HEX $ 30	=	0011 0000	DECIMAL	105	=	HEX $ 69	=	0110 1001
DECIMAL	49	=	HEX $ 31	=	0011 0001	DECIMAL	106	=	HEX $ 6A	=	0110 1010
DECIMAL	50	=	HEX $ 32	=	0011 0010	DECIMAL	107	=	HEX $ 6B	=	0110 1011
DECIMAL	51	=	HEX $ 33	=	0011 0011	DECIMAL	108	=	HEX $ 6C	=	0110 1100
DECIMAL	52	=	HEX $ 34	=	0011 0100	DECIMAL	109	=	HEX $ 6D	=	0110 1101
DECIMAL	53	=	HEX $ 35	=	0011 0101	DECIMAL	110	=	HEX $ 6E	=	0110 1110
DECIMAL	54	=	HEX $ 36	=	0011 0110	DECIMAL	111	=	HEX $ 6F	=	0110 1111
DECIMAL	55	=	HEX $ 37	=	0011 0111	DECIMAL	112	=	HEX $ 70	=	0111 0000
DECIMAL	56	=	HEX $ 38	=	0011 1000	DECIMAL	113	=	HEX $ 71	=	0111 0001

DECIMAL	114	=	HEX $ 72	=	0111 0010	DECIMAL	171	=	HEX $ AB	=	1010 1011
DECIMAL	115	=	HEX $ 73	=	0111 0011	DECIMAL	172	=	HEX $ AC	=	1010 1100
DECIMAL	116	=	HEX $ 74	=	0111 0100	DECIMAL	173	=	HEX $ AD	=	1010 1101
DECIMAL	117	=	HEX $ 75	=	0111 0101	DECIMAL	174	=	HEX $ AE	=	1010 1110
DECIMAL	118	=	HEX $ 76	=	0111 0110	DECIMAL	175	=	HEX $ AF	=	1010 1111
DECIMAL	119	=	HEX $ 77	=	0111 0111	DECIMAL	176	=	HEX $ B0	=	1011 0000
DECIMAL	120	=	HEX $ 78	=	0111 1000	DECIMAL	177	=	HEX $ B1	=	1011 0001
DECIMAL	121	=	HEX $ 79	=	0111 1001	DECIMAL	178	=	HEX $ B2	=	1011 0010
DECIMAL	122	=	HEX $ 7A	=	0111 1010	DECIMAL	179	=	HEX $ B3	=	1011 0011
DECIMAL	123	=	HEX $ 7B	=	0111 1011	DECIMAL	180	=	HEX $ B4	=	1011 0100
DECIMAL	124	=	HEX $ 7C	=	0111 1100	DECIMAL	181	=	HEX $ B5	=	1011 0101
DECIMAL	125	=	HEX $ 7D	=	0111 1101	DECIMAL	182	=	HEX $ B6	=	1011 0110
DECIMAL	126	=	HEX $ 7E	=	0111 1110	DECIMAL	183	=	HEX $ B7	=	1011 0111
DECIMAL	127	=	HEX $ 7F	=	0111 1111	DECIMAL	184	=	HEX $ B8	=	1011 1000
DECIMAL	128	=	HEX $ 80	=	1000 0000	DECIMAL	185	=	HEX $ B9	=	1011 1001
DECIMAL	129	=	HEX $ 81	=	1000 0001	DECIMAL	186	=	HEX $ BA	=	1011 1010
DECIMAL	130	=	HEX $ 82	=	1000 0010	DECIMAL	187	=	HEX $ BB	=	1011 1011
DECIMAL	131	=	HEX $ 83	=	1000 0011	DECIMAL	188	=	HEX $ BC	=	1011 1100
DECIMAL	132	=	HEX $ 84	=	1000 0100	DECIMAL	189	=	HEX $ BD	=	1011 1101
DECIMAL	133	=	HEX $ 85	=	1000 0101	DECIMAL	190	=	HEX $ BE	=	1011 1110
DECIMAL	134	=	HEX $ 86	=	1000 0110	DECIMAL	191	=	HEX $ BF	=	1011 1111
DECIMAL	135	=	HEX $ 87	=	1000 0111	DECIMAL	192	=	HEX $ C0	=	1100 0000
DECIMAL	136	=	HEX $ 88	=	1000 1000	DECIMAL	193	=	HEX $ C1	=	1100 0001
DECIMAL	137	=	HEX $ 89	=	1000 1001	DECIMAL	194	=	HEX $ C2	=	1100 0010
DECIMAL	138	=	HEX $ 8A	=	1000 1010	DECIMAL	195	=	HEX $ C3	=	1100 0011
DECIMAL	139	=	HEX $ 8B	=	1000 1011	DECIMAL	196	=	HEX $ C4	=	1100 0100
DECIMAL	140	=	HEX $ 8C	=	1000 1100	DECIMAL	197	=	HEX $ C5	=	1100 0101
DECIMAL	141	=	HEX $ 8D	=	1000 1101	DECIMAL	198	=	HEX $ C6	=	1100 0110
DECIMAL	142	=	HEX $ 8E	=	1000 1110	DECIMAL	199	=	HEX $ C7	=	1100 0111
DECIMAL	143	=	HEX $ 8F	=	1000 1111	DECIMAL	200	=	HEX $ C8	=	1100 1000
DECIMAL	144	=	HEX $ 90	=	1001 0000	DECIMAL	201	=	HEX $ C9	=	1100 1001
DECIMAL	145	=	HEX $ 91	=	1001 0001	DECIMAL	202	=	HEX $ CA	=	1100 1010
DECIMAL	146	=	HEX $ 92	=	1001 0010	DECIMAL	203	=	HEX $ CB	=	1100 1011
DECIMAL	147	=	HEX $ 93	=	1001 0011	DECIMAL	204	=	HEX $ CC	=	1100 1100
DECIMAL	148	=	HEX $ 94	=	1001 0100	DECIMAL	205	=	HEX $ CD	=	1100 1101
DECIMAL	149	=	HEX $ 95	=	1001 0101	DECIMAL	206	=	HEX $ CE	=	1100 1110
DECIMAL	150	=	HEX $ 96	=	1001 0110	DECIMAL	207	=	HEX $ CF	=	1100 1111
DECIMAL	151	=	HEX $ 97	=	1001 0111	DECIMAL	208	=	HEX $ D0	=	1101 0000
DECIMAL	152	=	HEX $ 98	=	1001 1000	DECIMAL	209	=	HEX $ D1	=	1101 0001
DECIMAL	153	=	HEX $ 99	=	1001 1001	DECIMAL	210	=	HEX $ D2	=	1101 0010
DECIMAL	154	=	HEX $ 9A	=	1001 1010	DECIMAL	211	=	HEX $ D3	=	1101 0011
DECIMAL	155	=	HEX $ 9B	=	1001 1011	DECIMAL	212	=	HEX $ D4	=	1101 0100
DECIMAL	156	=	HEX $ 9C	=	1001 1100	DECIMAL	213	=	HEX $ D5	=	1101 0101
DECIMAL	157	=	HEX $ 9D	=	1001 1101	DECIMAL	214	=	HEX $ D6	=	1101 0110
DECIMAL	158	=	HEX $ 9E	=	1001 1110	DECIMAL	215	=	HEX $ D7	=	1101 0111
DECIMAL	159	=	HEX $ 9F	=	1001 1111	DECIMAL	216	=	HEX $ D8	=	1101 1000
DECIMAL	160	=	HEX $ A0	=	1010 0000	DECIMAL	217	=	HEX $ D9	=	1101 1001
DECIMAL	161	=	HEX $ A1	=	1010 0001	DECIMAL	218	=	HEX $ DA	=	1101 1010
DECIMAL	162	=	HEX $ A2	=	1010 0010	DECIMAL	219	=	HEX $ DB	=	1101 1011
DECIMAL	163	=	HEX $ A3	=	1010 0011	DECIMAL	220	=	HEX $ DC	=	1101 1100
DECIMAL	164	=	HEX $ A4	=	1010 0100	DECIMAL	221	=	HEX $ DD	=	1101 1101
DECIMAL	165	=	HEX $ A5	=	1010 0101	DECIMAL	222	=	HEX $ DE	=	1101 1110
DECIMAL	166	=	HEX $ A6	=	1010 0110	DECIMAL	223	=	HEX $ DF	=	1101 1111
DECIMAL	167	=	HEX $ A7	=	1010 0111	DECIMAL	224	=	HEX $ E0	=	1110 0000
DECIMAL	168	=	HEX $ A8	=	1010 1000	DECIMAL	225	=	HEX $ E1	=	1110 0001
DECIMAL	169	=	HEX $ A9	=	1010 1001	DECIMAL	226	=	HEX $ E2	=	1110 0010
DECIMAL	170	=	HEX $ AA	=	1010 1010	DECIMAL	227	=	HEX $ E3	=	1110 0011

DECIMAL	228	=	HEX $ E4	=	1110 0100	DECIMAL	242	=	HEX $ F2	=	1111 0010
DECIMAL	229	=	HEX $ E5	=	1110 0101	DECIMAL	243	=	HEX $ F3	=	1111 0011
DECIMAL	230	=	HEX $ E6	=	1110 0110	DECIMAL	244	=	HEX $ F4	=	1111 0100
DECIMAL	231	=	HEX $ E7	=	1110 0111	DECIMAL	245	=	HEX $ F5	=	1111 0101
DECIMAL	232	=	HEX $ E8	=	1110 1000	DECIMAL	246	=	HEX $ F6	=	1111 0110
DECIMAL	233	=	HEX $ E9	=	1110 1001	DECIMAL	247	=	HEX $ F7	=	1111 0111
DECIMAL	234	=	HEX $ EA	=	1110 1010	DECIMAL	248	=	HEX $ F8	=	1111 1000
DECIMAL	235	=	HEX $ EB	=	1110 1011	DECIMAL	249	=	HEX $ F9	=	1111 1001
DECIMAL	236	=	HEX $ EC	=	1110 1100	DECIMAL	250	=	HEX $ FA	=	1111 1010
DECIMAL	237	=	HEX $ ED	=	1110 1101	DECIMAL	251	=	HEX $ FB	=	1111 1011
DECIMAL	238	=	HEX $ EE	=	1110 1110	DECIMAL	252	=	HEX $ FC	=	1111 1100
DECIMAL	239	=	HEX $ EF	=	1110 1111	DECIMAL	253	=	HEX $ FD	=	1111 1101
DECIMAL	240	=	HEX $ F0	=	1111 0000	DECIMAL	254	=	HEX $ FE	=	1111 1110
DECIMAL	241	=	HEX $ F1	=	1111 0001	DECIMAL	255	=	HEX $ FF	=	1111 1111

Table 2-3. Technical Data Needed to Use the Decimal Mini-Monitor Program.



NOTE 1: See Opcode Table 2-4 for the decimal opcode numbers that you can enter with this program.

NOTE 2: If you do not want to change the data in a memory location, you must enter the same data back into that location. Pressing the RETURN key without entering a number will automatically enter a ZERO in that memory location.

```
5 REM PROGRAM 2.2  FOR TABLE 2.2
10 DIM A$(20):DIM B$(20)
15 D= 0:OPEN4,4:CMD4
20 :
40 W=INT(D/16):X=W*16:Y=D-X
400 B$(0)="0":B$(1)="1":B$(2)="2":B$(3)="3":B$(4)="4":B$(5)="5":B$(6)="6"
405 B$(7)="7":B$(8)="8":B$(9)="9"
407 B$(10)="A":B$(11)="B"
410 B$(12)="C":B$(13)="D":B$(14)="E":B$(15)="F"
500 A$(0)="0000":A$(1)="0001":A$(2)="0010":A$(3)="0011":A$(4)="0100"
510 A$(5)="0101":A$(6)="0110":A$(7)="0111":A$(8)="1000":A$( 9)="1001"
520 A$(10)="1010":A$(11)="1011":A$(12)="1100":A$(13)="1101":A$(14)="1110"
530 A$(15)="1111"
600 PRINTCHR$(147)
635 PRINT"DECIMAL "D;:PRINT" =  HEX $ ";:PRINTB$(W)B$(Y);:PRINT"  =  ";:
636 PRINTA$(W);:PRINT" ";:PRINTA$(Y)
650 D=D+1:GOTO20
```

Program 2-2. This program was used to generate Table 2-2.

6502 monitor programs. If you are using a mini-monitor program, you will have to use Table 2-2 to convert the hexadecimal data into decimal data for that monitor.

The memory map of the computer is a listing that displays all of the memory address locations in the computer and the function that is allocated to each memory location. The usable RAM memory in the VIC-20, the C-16, and the PLUS/4 are all allocated to the BASIC program. If you want to use a machine-language subroutine, you must tell the computer to set up an area of memory for the machine-language subroutine. You do not have to do this with the C-64 because there is an area of memory between $C000 and $CFFF that can be used for machine-language subroutines. In the VIC-20 and the PLUS/4 this special area of machine-language memory can be set-up by using the POKE routines that are presented in Table 2-5. These routines must be entered into the computer using the *Immediate Programming Mode* before you enter your BASIC or machine-language subroutine. Remember that these POKE routines will lower the usable BASIC memory. It should be brought up at

```
1 REM PROGRAM 2.3
5 PRINTCHR$(147)
10 PRINT"A SIMPLE VIC-20
20 PRINT"MACHINE LANGUAGE"
30 PRINT"MONITOR PROGRAM"
40 PRINT" "
50 PRINT"INPUT DECIMAL ADDRESS"
60 INPUT"---";A
70 PRINT" LOCATION -- DATA
80 Z=PEEK(A)
90 PRINTA;"--";Z;:INPUTB
100 POKEA,B
110 A=A+1
120 GOTO80
```

Program 2-3. A VIC-20 mini-monitor program.

```
1 REM PROGRAM 2.4
5 PRINTCHR$(147)
10 PRINT"        A SIMPLE C-64"
20 PRINT"       MACHINE LANGUAGE"
30 PRINT"       MONITOR PROGRAM"
40 PRINT" "
50 PRINT"INPUT DECIMAL ADDRESS";:
60 INPUT" - ";A
70 PRINT"            LOCATION -  DATA"
80 Z=PEEK(A)
90 PRINT"      ADDRESS";A;"  --  ";Z;
   :INPUTB
100 POKEA,B
110 A=A+1
120 GOTO80
```

Program 2-4. A C-64 mini-monitor program.

Table 2-4. Decimal and Hexadecimal Opcodes of the Machine-Language Instructions Used In This Chapter.

INSTRUCTION	MNEMONIC	OPCODE HEX	OPCODE DECIMAL	ADDRESSING
Load the Accumulator	LDA	$A9	#169	Immediate
Load the Accumulator	LDA	$AD	#173	Absolute
Store the Accumulator	STA	$8D	#141	Absolute
Test Bits 6 and 7	BIT	$2C	#44	Absolute
- - - Branch Instructions Using Bits 6 and 7. - - -				
Branch if Bit 7 is ONE	BMI	$30	#48	Relative
Branch if Bit 7 is ZERO	BPL	$10	#16	Relative
Branch if Bit 6 is ONE	BVS	$70	#112	Relative
Branch if Bit 6 is ZERO	BVC	$50	#80	Relative
- - - Jump and Return Instructions - - -				
Jump to New Location	JMP	$4C	#76	Absolute
Jump to Subroutine	JSR	$20	#32	Absolute
Return from Subroutine	RTS	$60	#96	Implied

NOTE: A complete description of the machine language instruction set for the 6502 is presented in chapter 12.

this time that if your computer has a lot of unused (free) BASIC memory, you can most likely get by without adjusting your memory map. Generally, there will be an unused area in the memory where the BASIC part of your program will not interfere with the machine-language routine. A good spot to try is about **IK** below the highest memory location that is allocated to the BASIC RAM.

Programs 2-5 and 2-6 are written for the PLUS/4 computer. The objective of these two pro-

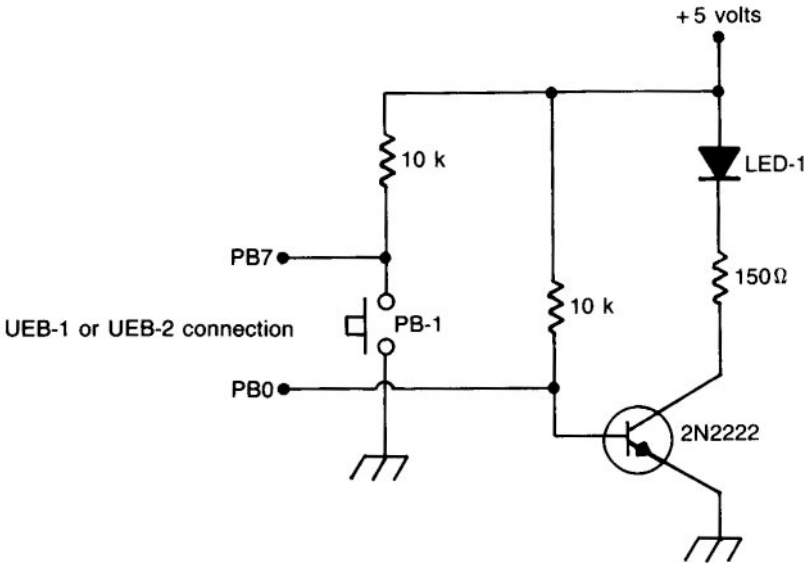


Fig. 2-2. This is the schematic of a simple pushbutton and transistor switch circuit that can be built on the UEB-I or UEB-2 circuit board. The main purpose of this circuit is to teach you the techniques of input and output interfacing using the machine-language programs in this chapter.

grams is to detect if the pushbutton PB-1 in the circuit of Fig. 2-2 is on or off. Program 2-5 is the main BASIC program while Program 2-6 is the machine-language subroutine that does the actual technical work. These first two programs will be described line by line so you can see how a BASIC program and a machine-language subroutine work together.

The first jump to the subroutine program occurs in BASIC program line 20. The SYS32512 command sends the program operation to memory location $7F00 (32512-$7F00). Remember that the $ just means that the number is a hexadecimal number. Location $7F00 contains the hex data $A9 which is the opcode for LDA Immediate (Load the Accumulator). This $A9 LDA opcode tells the computer's microprocessor to load the hex data in the

Table 2-5. POKE Routines that Can Be Used to Set-up a Machine-language Memory Area In the VIC-20 and the PLUS/4 Computers.

VIC-20 POKE data for setting up a machine language subroutine between $1000 and $1DFF.

POKE5,0:POKE52,29:POKE55,0:POKE56,29:CLR <press-return>

PLUS/4 POKE data for setting up a machine language subroutine between $7F00 and $7FFF.

POKE5,1,0:POKE52,27:POKE55,0:POKE56,127:CR <press-return>

NOTE: A "$" in front of a number means that the number is a Hexadecimal number.

You can recheck the free BASIC memory by using this routine.

? FRE(O) <press-return>

```
1 REM - PROGRAM 2.5 FOR USE WITH MACHINE LANGUAGE SUBROUTINE OF PROGRAM 2.6
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE DEMONSTRATION"
20 SYS 32512: REM JUMP TO I/O SET-UP ROUTINE AT $7F00
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS 32520: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE
50 A=PEEK(32544):REM CHECK PUSH BUTTON DATA IN $7F20
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"": PRINT"PUSH BUTTON IS OFF     ":GOTO40
90 PRINT"": PRINT"PUSH BUTTON IS ON     ":GOTO40
```

Program 2-5. This PLUS/4 BASIC program is used to check if PB-1 of Fig. 2-2 is open or closed using the machine-language subroutine of Program 2-6.

next memory location ($7F01) into the microprocessor's accumulator register for further processing. The computer's microprocessor then looks at the hex data in location ($7F02) which is $8D. The opcode $8D (ST A) tells the microprocessor to store the accumulator's hex data in the memory location that is called for by the data that is contained in the next two memory locations ($7F03 and $7F04). Location $7F03 is called the low-byte data and $7F04 is called the high-byte data for the opcode ST A. The hex data in $7F04 is $FD and $7F03 is $01. The computer then puts the HI byte and the LOW byte data together to form a memory address number of $FD10. This means that the hex data of $C0, which is in the accumulator's register, will be moved to the memory location $FD10, which is the PLUS/4's on board I/O port. The $C0, which is 11000000 in binary notation, makes bits 7 and 6 input lines and bits 0, 1, 2, 3, 4, and 5 output lines for our program. Memory location $7F05 contains the data $60, which is the opcode for *RTS* or *Return from subroutine*. This opcode sends the program control back to the BASIC program at line 30. Line 30 simply tells you that the I/O port is set up and adds a one second delay loop. Table 2-6 presents some of the nomenclature that is used with machine-language programming.

The next SYS command in line 40 (SYS32520) sends the program control back to the machine-language routine at address $7F08. The machine-language opcode instruction in location $7F08 is $2C, which is the BIT test instruction. The BIT test instruction performs several tests, but in this program it is checking to see if bit 7 of the I/O USER PORT is a ONE or ZERO. Memory locations $7F09 and $7F0A contain the hex data that specifies the memory location to be tested, which is $FD10. Note that this memory address is specified in the low byte-high byte order, which is the way all machine language addresses are specified.

The next opcode in location $7F0B is $30, which is a branching instruction. If bit 7 of the

```
MACHINE LANGUAGE SUBROUTINE
PROGRAM 2.6.

7F00  A9 C0     LDA #$C0
7F02  8D 10 FD  STA $FD10
7F05  60        RTS
7F06  00        BRK
7F07  00        BRK
7F08  2C 10 FD  BIT $FD10
7F0B  30 03     BMI $7F10
7F0D  4C 18 7F  JMP $7F18
7F10  A9 00     LDA #$00
7F12  8D 20 7F  STA $7F20
7F15  60        RTS
7F16  00        BRK
7F17  00        BRK
7F18  A9 FF     LDA #$FF
7F1A  8D 20 7F  STA $7F20
7F1D  60        RTS
7F1E  00        BRK
7F1F  00        BRK
```

Program 2-6. This PLUS/4 machine-language subroutine is to be used with BASIC Program 2-5.

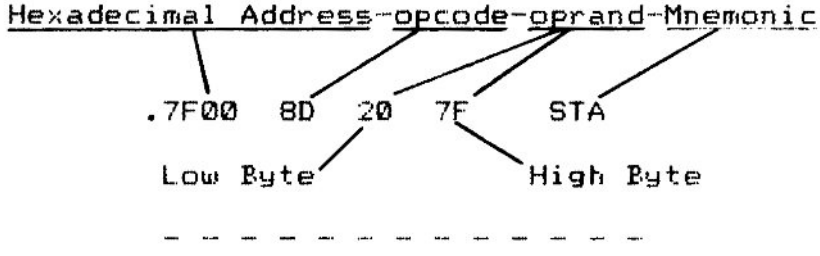
Table 2-6. This Machine-Language Nomenclature
Presentation Should Help You Understand the Machine-language Programming In This Chapter.

Opcode : The hexadecimal equivalent of the binary code that is assigned to a machine language instruction.

Oprand or Oprand Field : The required data or address used by a specific machine language instruction.

Hexadecimal Address : A four digit memory location somewhere in the computer's 65,535 bytes of memory.

Mnemonic : A three letter abbreviation for the name of an opcode instruction.



This machine language subroutine uses the BIT test and Branching instruction to check for a logic ZERO on bit 7.

```
7F00  2C  10  FD    BIT  $FD10
7F03  30  FB        BMI  $7F00
7F05  60            RTS
```

In the above subroutine, the BIT test checks the logic level of bit 7 at address $FD10. The BRANCH instruction ($30) branches back to $7F00 if the BIT test shows that bit 7 is a logic ONE. As long as bit 7 remains at a logic ONE level, the program will stay in the branching loop formed by the BIT and BMI instructions. When bit 7 goes to a logic ZERO, the program does not branch, but goes to the next instruction in location $7F05 which is a RETURN from SUBROUTINE instruction. One might compare the machine language BRANCH instruction to the IF-THEN statement in BASIC, because its like saying that "IF" bit 7 is a logic ONE, "THEN" branch back to location $7F00. This application of the BRANCH instruction is only one of the many ways it can be used.

```
1 REM - PROGRAM 2.7 FOR USE WITH MACHINE LANGUAGE SUBROUTINE OF PROGRAM 2.8
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE DEMONSTRATION"
20 SYS 49152: REM JUMP TO I/O SET-UP ROUTINE AT $C000
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS 49163: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE AT C00B
50 A=PEEK(49184):REM CHECK PUSH BUTTON DATA IN $C020
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"■": PRINT"PUSH BUTTON IS OFF     ":GOTO40
90 PRINT"■": PRINT"PUSH BUTTON IS ON     ":GOTO40
```

Program 2-7. This C-64 BASIC program is used to check if PB-1 of Fig. 2-2 is open or closed using the machine-language subroutine of Program 2-8.

USER PORT is a logic ONE, a branch to a new location occurs that is controlled by the hex data, which is in the following memory location. If bit 7 is a logic ZERO, no branching occurs and the program continues on with the opcode in the next location. If a branch does occur, the $03 in $7F0C sends the program operation to location $7F10. Locations $7F10 and $7F11 contain $A9-$00 and locations $7F12, $7F13, and $7F14 contain $8D-$20-$7F, which tells the microprocessor to store a $00 in location $7F20 if bit 7 of the USER PORT is a logic ONE during the BIT test. If bit 7 is a logic ZERO, no branch occurs and the program operation goes to location $7F0D which contains a JUMP instruction ($4C-$18-$7F) to location $7F18. Locations $7F18 to $7FIC contain $A9-$FF-$8D-$20-$7F, which tells the microprocessor to store a $FF in location $7F20. Locations $7F15 and $7FID contain $60 which is an RTS instruction that sends the program control back to BASIC line 50. It can be seen now that the main objective of this part of the machine-language subroutine is to store a $00 in location $7F20 if the pushbutton is OFF or a $FF if the pushbutton is ON.

Line 50 of the BASIC program runs a PEEK to memory location 32544 ($7F20) to see if the number in 32544 is a 0 ($00) or 255 ($FF). Variable "A" is set equal to the number data that is contained in memory location 35244. Lines 60 and 70 are IF-THEN STATEMENTS, that do the decision making in our BASIC program. If A = 0, then the program jumps to line 80, and if A = 255, the program jumps to line 90. Line 80 prints the video message that the pushbutton is off, and line 90 prints the video message that the pushbutton is on. Both lines 80 and 90 contain a GOTO 40 command that loops the program back around for another trip through the subroutine to check the pushbutton. Note that line 30 is only used once to set-up the I/O USER PORT.

If you study Programs 2-5 and 2-6 thoroughly, you will gain an understanding of how the BASIC and machine-language subroutines function in this chapter. Since all of the other machine-language programs in this chapter are about the same, we will not go into a detailed discussion of these pro-

```
MACHINE LANGUAGE SUBROUTINE
    2.8 FOR PROGRAM 2.7

,C000 A9 63    LDA #$63
,C002 8D 03 DD STA $DD03
,C005 A9 00    LDA #$00
,C007 8D 01 DD STA $DD01
,C00A 60       RTS
,C00B 2C 01 DD BIT $DD01
,C00E 30 03    BMI $C013
,C010 4C 19 C0 JMP $C019
,C013 A9 00    LDA #$00
,C015 8D 20 C0 STA $C020
,C018 60       RTS
,C019 A9 FF    LDA #$FF
,C01B 8D 20 C0 STA $C020
,C01E 60       RTS
```

Program 2-8. This C-64 machine-language subroutine is to be used with BASIC Program 2-7.

A VIC-20 PROGRAM

```
1 REM - PROGRAM 2.9 FOR USE WITH MACHINE LANGUAGE SUBROUTINE OF PROGRAM 2.10
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE   DEMONSTRATION"
20 SYS  7424: REM JUMP TO I/O SET-UP ROUTINE AT $1D00
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS  7435: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE AT $1D0B
50 A=PEEK( 7456):REM CHECK PUSH BUTTON DATA IN $1D20
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"■": PRINT"PUSH BUTTON IS OFF     ":GOTO40
90 PRINT"■": PRINT"PUSH BUTTON IS ON     ":GOTO40
```

Program 2-9. This VIC-20 BASIC program is used to check if PB-l of Fig. 2-2 is open or closed using the machine-language subroutine of Program 2-10.

grams. The LOAD, STORE, BRANCH, BIT, and RETURN machine-language instructions are all that are needed to use your computer as a high-speed programmable controller. When you combine them with a BASIC program, you can make a very intelligent high-speed control system. One can appreciate the speed of machine language when you considerthat most of the machine-language instructions are completed in three to five microseconds.

Programs 2-7,2-8,2-9, and 2-10 are the same as 2-5 and 2-6 but they are used with the C-64 and the VIC-20. Programs 2-11 through 2-16 are about the same as 2-5 through 2-10 but they have the added machine-language subroutine to turn LED-l on and off. Using Programs 2-11 and 2-12 for the PLUS/4 again shows that two SYS jumps have been added in lines 80 and 90. The SYS32546 in line 80 jumps the program operation to memory location $7F22, which is used to tum off the LED. The SYS32552 in line 90 jumps the program operation to memory $7F28, which is used to tum the LED on. Since the SYS command requires an RTS machine-code instruction, lines 85 and 95 are used to receive the program operation back from the machine-languagesubroutine and loop it back to the pushbutton check part of the program. The objectives of Programs 2-11 to 2-16 are to show you how an input logic signal can be used to control an output logic signal using a machine-language subroutine. Using the BIT test instruction as shown, it is possible to easily detect logic signal pulses as short as 12 microseconds long.

A VIC-20 PROGRAM

```
1D00 LDA #$63
1D02 STA $9112
1D05 LDA #$00
1D07 STA $9110
1D0A RTS
1D0B BIT $9110
1D0E BMI $1D13
1D10 JMP $1D19
1D13 LDA #$00
1D15 STA $1D20
1D18 RTS
1D19 LDA #$FF
1D1B STA $1D20
1D1E RTS
1D1F BRK
1D20 BRK
```

PROGRAM 2. Ie

Program 2-10. This VIC-20 machine-language subroutine is to be used with BASIC Program 2-9.

```
1 REM - PROGRAM 2.11 FOR USE WITH MACHINE LANGUAGE SUBROUTINE OF PROGRAM
  2.12
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE DEMONSTRATION"
20 SYS 32512: REM JUMP TO I/O SET-UP ROUTINE AT $7F00
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS 32520: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE
50 A=PEEK(32544):REM CHECK PUSH BUTTON DATA IN $7F20
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"": PRINT"PB-1 IS OFF - LED-1 IS OFF":SYS32546
85 GOTO40
90 PRINT"": PRINT"PB-1 IS ON - LED-1 IS ON    ":SYS32552
95 GOTO40
```

Program 2-11. This PLUS/4 BASIC program along with the machine-language subroutine of Program 2-12 is used to turn LED-I on and off using the circuit of Fig. 2-2 and PB-1.

A TIME-DELAY SUBROUTINE

Most control programs need time-delay routines at some point in the operation of the program. If you are programming in BASIC, you will not have any problems because a simple FOR-NEXT loop can be used to generate the needed time delay. Writing a time-delay routine for machine-language control is not quite as simple. To help you out of this problem, three time-delay subroutines are presented in Programs 2-17, 2-18, and 2-19 for the VIC-20, the C-64, and the PLUS/4. These time-delay routines can be used to generate time delays from about 20 microseconds to over 250 milliseconds.

The time-delay routines can be located anywhere in the computer's free RAM memory and called up by using a SYS BASIC command, a JMP, or a JSR machine-language instruction. Because all of the routines are the same with the exception of the different address locations for the three computers, we will use the routine in Program 2-18 for the C-64 computer to explain the operation of the time-delay function. Using program address lines $C000, $C002, $C005, and $C007, the routine first loads the hex number $FF into memory locations $C015 and $C016. Program line $C00A decrements memory location $C016 by "I" and program line $C00D checks to see if that decrement caused location $C016 to become $00. If $C016 was not zero,

```
MACHINE LANGUAGE SUBROUTINE 2.12
FOR PROGRAM 2.11

. 7F00  A9 C0     LDA #$C0
. 7F02  8D 10 FD  STA $FD10
. 7F05  60        RTS
. 7F06  00        BRK
. 7F07  00        BRK
. 7F08  2C 10 FD  BIT $FD10
. 7F0B  30 03     BMI $7F10
. 7F0D  4C 18 7F  JMP $7F18
. 7F10  A9 00     LDA #$00
. 7F12  8D 20 7F  STA $7F20
. 7F15  60        RTS
. 7F16  00        BRK
. 7F17  00        BRK
. 7F18  A9 FF     LDA #$FF
. 7F1A  8D 20 7F  STA $7F20
. 7F1D  60        RTS
. 7F1E  00        BRK
. 7F1F  00        BRK
. 7F20  00        BRK
. 7F21  00        BRK
. 7F22  A9 C0     LDA #$C0
. 7F24  8D 10 FD  STA $FD10
. 7F27  60        RTS
. 7F28  A9 C1     LDA #$C1
. 7F2A  8D 10 FD  STA $FD10
. 7F2D  60        RTS
. 7F2E  00        BRK
```

Program 2-12. This PLUS/4 machine-language subroutine is used with BASIC Program 2-11.

```
1 REM - FOR THE C-64
2 REM - PROGRAM 2.13 FOR USE WITH MACHINE LANGUAGE SUBROUTINE 2.14
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE DEMONSTRATION"
20 SYS 32512: REM JUMP TO I/O SET-UP ROUTINE AT $7F00
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS 32523: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE AT $7F0B
50 A=PEEK(32544):REM CHECK PUSH BUTTON DATA IN $7F20
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"■": PRINT"PB-1 IS OFF - LED-1 IS OFF":SYS32545
85 GOTO40
90 PRINT"■": PRINT"PB-1 IS ON - LED-1 IS ON    ":SYS32551
95 GOTO40
```

Program 2-13. This C-64 BASIC program along with the machine-language subroutine of Program 2-14 is used to turn LEO-I on and off using the circuit of Fig. 2-2 and PB-I. Note that the machine-language subroutine for this BASIC program is located at $7FOO to show you that a subroutine can be located in the BASIC program RAM if the BASIC program is not too large.

```
MACHINE LANGUAGE SUBROUTINE 2.14
FOR PROGRAM 2.13


,7F00 A9 63    LDA #$63
,7F02 8D 03 DD STA $DD03
,7F05 A9 00    LDA #$00
,7F07 8D 01 DD STA $DD01
,7F0A 60       RTS
,7F0B 2C 01 DD BIT $DD01
,7F0E 30 03    BMI $7F13
,7F10 4C 13 7F JMP $7F19
,7F13 A9 00    LDA #$00
,7F15 8D 20 7F STA $7F20
,7F18 60       RTS
,7F19 A9 FF    LDA #$FF
,7F1B 8D 20 7F STA $7F20
,7F1E 60       RTS
,7F1F 00       BRK
,7F20 00       BRK
,7F21 A9 00    LDA #$00
,7F23 8D 01 DD STA $DD01
,7F26 60       RTS
,7F27 A9 01    LDA #$01
,7F29 8D 01 DD STA $DD01
,7F2C 60       RTS
,7F2D 00       BRK
```

Program 2-14. This C-64 machine-language subroutine is used with BASIC Program 2-13.

the routine loops to line $C00A and decrements again until $C016 is zero. When $C016 is zero, the routine does not loop back, but goes to program line $C00F and decrements memory location $C015. Line $C012 checks to see if location $C015 was decremented to zero, and if it was not zero, the routine loops to line $C005 and reloads $FF into $C016 for another decrement loop. As one can observe, there are two loops in this routine. The routine will continue the decrement function until both loops are zero at the same time. When both locations $C015 and $C016 are $00, the routine goes on to line $C014, which is a RETURN FROM SUBROUTINE instruction that sends the program control operation back to the main computer program.

The time delay of the routine is controlled by the data that is loaded into locations $C001 and $C006. The longest delay is obtained when $FF is loaded into both of these locations and the shortest time delay is obtained when $02 is used.

A FINAL NOTE ON CHECKING SWITCHES AND PUSHBUTTONS

The example programs in this chapter only used bit 7 to check for an open or closed pushbutton. You can use the BIT TEST instruction to check the logic levels of both bits 6 and 7 of any

```
1 REM - FOR THE VIC-20
2 REM - PROGRAM 2.15 FOR USE WITH MACHINE LANGUAGE SUBROUTINE OF PROGRAM 2.16
5 PRINTCHR$(147)
10 PRINT" A MACHINE LANGUAGE DEMONSTRATION"
20 SYS  7424: REM JUMP TO I/O SET-UP ROUTINE AT $1D00
30 PRINT" ":PRINT" I/O PORT IS SET-UP":FOR I=1 TO 1000:NEXT:PRINTCHR$(147)
40 SYS  7435: REM JUMP TO PUSH BUTTON CHECK SUBROUTINE AT $1D0B
50 A=PEEK( 7456):REM CHECK PUSH BUTTON DATA IN $1D20
60 IF A=0 THEN GOTO 80
70 IF A=255 THEN GOTO 90
80 PRINT"■": PRINT"LED-1 IS OFF     ":SYS7458
85 GOTO40
90 PRINT"■": PRINT"LED-1 IS ON     ":SYS7464
95 GOTO40
```

Program 2-15. This VIC-20 BASIC program along with the machine-language subroutine of Program 2-15 is used to turn LED-I on and off using the circuit of Fig. 2-2 and PB-I.

memory location in the computer. Table 2-7 presents four routines that can be used for checking the logic levels of bits 6 and 7. These bit test routines show only one of the many different applications that use the function of the BIT TEST instruction.

The routine shown in program lines $C000 and $C002 can be used to test for a logic ZERO on bit 7. The routine will loop within itself as long as bit 7 remains a logic ONE. Any logic ZERO pulse that is longer than 10 microseconds will be detected by the BIT TEST and the program control will go on to the next memory location after the routine, which in our routine is location $C005. $C005 is a BRK

```
.
MACHINE LANGUAGE SUBROUTINE PROGRAM
2.16 FOR PROGRAM 2.15
., 1D00 LDA #$63
., 1D02 STA $9112
., 1D05 LDA #$00
., 1D07 STA $9110
., 1D0A RTS
., 1D0B BIT $9110
., 1D0E BMI $1D13
., 1D10 JMP $1D19
., 1D13 LDA #$00
., 1D15 STA $1D20
., 1D18 RTS
., 1D19 LDA #$FF
., 1D1B STA $1D20
., 1D1E RTS
., 1D1F BRK
., 1D20 BRK
., 1D21 BRK
., 1D22 LDA #$00
., 1D24 STA $9110
., 1D27 RTS
., 1D28 LDA #$01
., 1D2A STA $9110
., 1D2D RTS
., 1D2E BRK
.
```

Program 2-16. This VIC-20 machine-language subroutine is used with BASIC Program 2-15.

```
,1D00 A9 FF     LDA #$FF
,1D02 8D 15 1D  STA $1D15
,1D05 A9 FF     LDA #$FF
,1D07 8D 16 1D  STA $1D16
,1D0A CE 16 1D  DEC $1D16
,1D0D D0 FB     BNE $1D0A
,1D0F CE 15 1D  DEC $1D15
,1D12 D0 F1     BNE $1D05
,1D14 60        RTS
,1D15 00        BRK
```

A TIME DELAY MACHINE LANGUAGE SUBROUTINE FOR THE VIC-20.

Program 2-17. This is a machine-language time-delay subroutine for the VIC-20.

```
,C000 A9 FF     LDA #$FF
,C002 8D 15 C0 STA $C015
,C005 A9 FF     LDA #$FF
,C007 8D 16 C0 STA $C016
,C00A CE 16 C0 DEC $C016
,C00D D0 FB     BNE $C00A
,C00F CE 15 C0 DEC $C015
,C012 D0 F1     BNE $C005
,C014 60        RTS
,C015 00        BRK


A TIME DELAY MACHINE LANGUAGE
SUBROUTINE FOR THE C-64.
```

Program 2-18. This is a machine-language time-delay subroutine for the C-64.

```
,7F00 A9 FF     LDA #$FF
,7F02 8D 15 7F STA $7F15
,7F05 A9 FF     LDA #$FF
,7F07 8D 16 7F STA $7F16
,7F0A CE 16 7F DEC $7F16
,7F0D D0 FB     BNE $7F0A
,7F0F CE 15 7F DEC $7F15
,7F12 D0 F1     BNE $7F05
,7F14 60        RTS
,7F15 00        BRK


A TlME DELAY MACHINE LANGUAGE
SUBROUTINE FOR THE PLUS/4.
```

Program 2-19. This is a machine-language time-delay subroutine for the PLUS/4.

Table 2-7. Four Machine-Language Routines that Will Check the Logic Levels of Bits 6 and 7 of a Memory Location. See the Discussion In the Text.

```
,C000 2C 01 DD BIT $DD01
,C003 30 FB     BMI $C000
,C005 00        BRK
,C006 2C 01 DD BIT $DD01
,C009 10 FB     BPL $C006
,C00B 00        BRK
,C00C 2C 01 DD BIT $DD01
,C00F 50 FB     BVC $C00C
,C011 00        BRK
,C012 2C 01 DD BIT $DD01
,C015 70 FB     BVS $C012
,C017 00        BRK
```

instruction which will stop the program function. If you have a monitor program with a GO command, you can load this routine and study its operation. You can also change $C005 to aRTS instruction ($60- Return From Subroutine) and use this routine with the BASIC SYS command. The routine in lines $C006 and $C009 can be used to detect a logic ONE on bit 7. The routine in lines $C00C and $C00F is used to detect a logic ONE on bit 6 while the routine at lines $C012 and $C015 can be used to detect a logic ZERO on bit 6.

In this chapter, you have been introduced to machine-language routines that can be used to control experiments and machines. The scope of this chapter has been kept narrow so one can learn how to use a few of the really important machine-language instructions.

Chapter 3

Special Projects

ALL OF THE SPECIAL PROJECTS THAT FOLlow in this book use some form of a timing program. The timing function can be as simple as a FOR-NEXT loop or as complicated as a timing subroutine that can detect time intervals as short as a few microseconds.

PROJECT 3-1- TIMING PROGRAMS

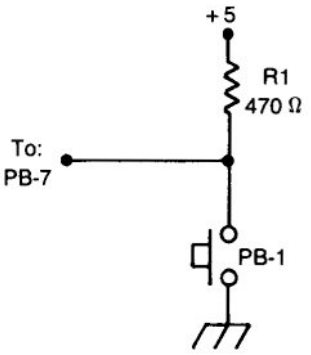
The timing programs presented in this section can be used to measure time intervals of over 1000 minutes with a verifiable accuracy of .0001 seconds. The programs really have a time measurement resolution of .000010 seconds, but the average hobbyist or experimenter will find it difficult to verify the accuracy of this level of measurement resolution.

Timer Programs Using Only BASIC

The first three timing programs that will be presented are Program 3-1 for the VIC-20, Program 3-2 for the C-64, and Program 3-3 for the PLUS/4. These programs use the built-in real time clock that is part of each computer. The built-in clock keeps track of time in increments of 1160of a second. The clock starts out at zero when the computer is turned on and has the ability to count up to about 14,400 hours before it resets back to zero. If you do not have to measure time in increments smaller than 1/60 of a second, the built-in real time clock is the way to go.

All three of the timing programs are designed to operate with the USER PORTs of the three computers. The I/O functions of the USER PORTs give you the ability to time external events using input sensors. The input sensor that is used with these programs is a simple pushbutton. The pushbutton circuit is presented in Fig. 3-1 and can be built on the UEB-1 or UEB-2 (see Chapter 1). The function of the pushbutton circuit is to generate a logic ONE or ZERO on input line PB7. The pushbutton (PB-1) is connected so that port line PB7 is held at a logic ONE level when PB-1 is open or dropped to a logic ZERO level when PB-1 is pressed (closed).

The programs are written so the timing period starts when the <A> key is pressed on the com-



This circuit can be built on the UEB-1 or UEB-2

Fig. 3-1. This simple pushbutton circuit is used to place a logic ONE or ZERO on input line PB7 for timer Programs 3-1 to 3-9.

puter's keyboard and stops when the pushbutton of Fig. 3-1 is pressed. The time interval data is then displayed on the video screen. Programs 3-1 and 3-2 for the VIC-20 and the C-64 are the same with the exception of the different PEEK address in line 60 and the video data format. Line 30 and 40 of each program checks to see if the <A> key has been pressed. When it has been pressed, line 50 sets the variable "B" equal to the current time value. Lines 60, 70, and 80 are used to check if PB-1 has been pressed. When PB-1 is pressed, variable "D" is set equal to the current timer value. Line 100 is used to compute the time interval between the time when the <A> key was pressed and PB-1 was pressed. The time interval data that is shown on the video screen is only accurate to plus or minus 1160 of a second. Program 3-3 for the PLUS/4 computer is about the same as Programs 3-1 and 3-2, but line 5 contains a POKE command that is needed to make the PLUS/4's USER PORT an input port.

Machine-Language Timing Subroutines

The timer Programs of 3-1, 3-2, and 3-3 use only BASIC programming and the built-in real time clock in the computer. These programs have a limited time resolution because BASIC is slow and the real time clock increments only once every 1160 of a second. If one needs to secure a time interval with a resolution better than .016 seconds (1/60), you must use a machine-language subroutine that works with microseconds and not a BASIC program that works with milliseconds.

BASIC Programs 3-4, 3-6, and 3-8 along with their machine-language subroutines are designed to provide a verifiable time interval measurement resolution of better than .0001 seconds for the measured time period. Each of these BASIC programs use the machine-language subroutine to do the actual time keeping function using the execution speed of machine language programming. The machine-language subroutines that are presented in Programs 3-5, 3-7, and 3-9 are shown in the machine-language monitor disassembly format for each of the computers. One special note about using long machine-language subroutines is that you will

```
5 PRINTCHR$(147)
10 PRINT"A VIC-20 TIMER"
20 PRINT" ":PRINT"TO START TIMER":PRINT"PUSH <A>"
30 GET A$:IF A$="A" THEN GOTO50
40 GOTO30
50 B=TI: PRINT" ":PRINT"TIMER RUNNING"
60 C=PEEK(37136)
70 IF C=127 THEN GOTO90
80 GOTO60
90 D=TI:PRINT" ":PRINT"FINISHED"
100 E=(D-B)/60:E=INT(E*100)/100
110 PRINT" ":PRINT" TIME PERIOD = ";:PRINTE;:PRINT" SECONDS"
```

Program 3-1. This BASIC program is used to measure time intervals with a VIC-20 computer.

```
5 PRINTCHR$(147)
10 PRINT"A C-64 TIMING PROGRAM"
20 PRINT" ":PRINT"TO START TIMER - PUSH <A>"
30 GET A$:IF A$="A" THEN GOTO50
40 GOTO30
50 B=TI: PRINT" ":PRINT"TIMER RUNNING"
60 C=PEEK(56577)
70 IF C=127 THEN GOTO90
80 GOTO60
90 D=TI:PRINT" ":PRINT"FINISHED"
100 E=(D-B)/60:E=INT(E*100)/100
110 PRINT" ":PRINT" TIME PERIOD = ";:PRINTE;:PRINT" SECONDS"
```

Program 3-2. This BASIC program is used to measure time intervals with a C-64 computer.

```
5 PRINTCHR$(147)
10 PRINT"A PLUS/4 TIMING PROGRAM"
20 PRINT" ":PRINT"TO START TIMER - PUSH <A>"
30 GET A$:IF A$="A" THEN GOTO50
40 GOTO30
50 B=TI: PRINT" ":PRINT"TIMER RUNNING"
60 C=PEEK(64874)
70 IF C=127 THEN GOTO90
80 GOTO60
90 D=TI:PRINT" ":PRINT"FINISHED"
100 E=(D-B)/60:E=INT(E*100)/100
110 PRINT" ":PRINT" TIME PERIOD = ";:PRINTE;:PRINT" SECONDS"
```

Program 3-3. This BASIC program is used to measure time intervals with a PLUS/4 computer.

```
5 X=16777216:Y=65535:Z=255:PRINTCHR$(147)
10 PRINT"A VIC-20 TIMER":PRINT" "
20 PRINT"TO START PUSH <G>":PRINT" "
25 PRINT"TO STOP PUSH                    <CRSR DOWN>":PRINT" "
30 PRINT"READY - PUSH  <G>":PRINT" "
35 GETA$:IF A$="G"THEN GOTO 40
36 GOTO35
40 SYS7168
50 A=PEEK( 7219):B=PEEK( 7218):C=PEEK( 7217):D=PEEK( 7216)
60 FOR I= 0 TO 500: NEXT
70 A=255-A:B=255-B:C=255-C:D=255-D
80 W =(A)+(B*Z)+(C*Y)+(D*X)
90 T=W*9.00199394E-6
100 T1=INT(T*10000)/10000
110 PRINT" ":PRINT"THE TIME PERIOD IS - ";:
120 PRINTT1;:PRINT" SECONDS"
```

Program 3-4. This BASIC program uses the machine-language subroutine in Program 3-5 to measure time intervals with a VIC-20 computer.

```
.
.; 1C00 NOP
.; 1C01 NOP
.; 1C02 NOP
.; 1C03 NOP
.; 1C04 NOP
.; 1C05 LDA #$FF
.; 1C07 STA $1C30
.; 1C0A STA $1C31
.; 1C0D STA $1C32
.; 1C10 STA $1C33
.; 1C13 DEC $1C33
.; 1C16 BNE $1C13
.; 1C18 DEC $1C32
.; 1C1B BNE $1C28
.; 1C1D DEC $1C31
.; 1C20 BNE $1C28
.; 1C22 DEC $1C30
.; 1C25 BNE $1C28
.; 1C27 RTS
.; 1C28 BIT $9121
.; 1C2B BMI $1C13
.; 1C2D RTS
.; 1C2E BRK
.; 1C2F BRK
.
```

Program 3-5. This machine-language subroutine is used with BASIC Program 3-4.

have to save your subroutine programs on tape or disk after you have entered them into the computer with the monitor program. When you are loading a machine-language program using the LOAD "XXX" ,8,1 method for a disk system or the LOAD "XXX",1,l for the tape cassette, you will have to "RESET" the computer before you can load in the BASIC program. To reset the computers, you can use the PLUS/4's built-in reset button or the reset button on the UEB-l for the VIC-20 or the C-64.

The three BASIC programs and their machine-language subroutines use the same general program format that was used in Programs 3-1,3-2, and 3-3. The program timing interval starts when you press the <G> key and stops when you press the <RUN/STOP> key for the C-64, the <CRSR/DOWN> key for the VIC-20, or pushbutton PB-1 of Fig. 3-1 for PLUS/4. The C-64 machine-language subroutine of Program 3-5 uses the timers in the 6526 CIA to measure the time period. The VIC-20 and the PLUS/4 subroutines of Programs 3-7 and 3-9 use an internal timing loop that keeps track of time by decrementing three memory locations. After the time interval is complete, each of the main BASIC programs computes the measured

```
5 X=65535: Y=256 : PRINTCHR$(147)
10 PRINT " A C-64 TIMER PROGRAM" : PRINT" "
20 PRINT" TO START TIMER PUSH THE 'G' KEY":PRINT" "
25 PRINT" TO STOP - PUSH  STOP/RUN KEY": PRINT" "
27 PRINT"READY TO START- PUSH 'G'":PRINT" "
30 GET A$:IF A$="G" THEN GOTO40
35 GOTO30
40 SYS49152
45 FOR I=0TO500:NEXT
50 A=PEEK(56580):B=PEEK(56581):C=PEEK(56582):D=PEEK(56583)
70 FOR I=0TO500:NEXT
80 POKE 56334,01
90 W=X-(B*256+A)
100 Q=X-(D*256+C)
110 R=Q*X
120 D=R+W
130 M=(D/1000000)*.978641907
140 PRINT" TIMED PERIOD =";:PRINTM;:PRINT" SECONDS"
150 END
```

Program 3-6. This BASIC program uses the machine-language subroutine in Program 3-7 to measure time intervals with a C-64 computer.

```
,C000 A9 00    LDA #$00
,C002 8D 0E DC STA $DC0E
,C005 A9 FF    LDA #$FF
,C007 8D 04 DD STA $DD04
,C00A 8D 05 DD STA $DD05
,C00D 8D 06 DD STA $DD06
,C010 8D 07 DD STA $DD07
,C013 A9 10    LDA #$10
,C015 8D 0E DD STA $DD0E
,C018 8D 0F DD STA $DD0F
,C01B A9 01    LDA #$01
,C01D 8D 0E DD STA $DD0E
,C020 A9 41    LDA #$41
,C022 8D 0F DD STA $DD0F
,C025 2C 01 DC BIT $DC01
,C028 30 FB    BMI $C025
,C02A A9 00    LDA #$00
,C02C 8D 0E DD STA $DD0E
,C02F 8D 0F DD STA $DD0F
,C032 A9 00    LDA #$00
,C034 8D 0E DC STA $DC0E
,C037 60       RTS
,C038 00       BRK
```

Program 3-7. This machine-language subroutine is used with BASIC Program 3-6.

time interval by PEEKing the memory locations where the time data is stored and using that data in the program calculations for the time interval video display.

The time measurement that is made by Programs 3-4, 3-6, and 3-8 depend on the internal clock frequency in your computer. If each computer had 1 MHz clock frequency, calculating the correct time interval would be very easy because every clock cycle would be .000001 seconds. Because none of the computers use an even 1 MHz clock frequency, you must correct for this factor. The correction factor in Program 3-4 is in line 90 and is 9.00199394E-6. The correction factor in Program 3-6 is in line 130 and is .978641907. The correction factor in Program 3.8 is in line 90 and is 1.59416303E-5. These three factors are correct for the computers which were used to write this book, but yours will not be exactly the same. The best way to find your correction factor is to locate a radio station that generates a tone burst at the beginning of every hour. You can start your timing interval on one tone burst and stop it on the next burst. It is pretty easy to adjust your correction factor to secure a measured time interval of one hour plus or minus .01 seconds by using the pushbuttons. More accurate measurements will require some form of a tone decoder circuit that is connected to port line PB7.

PROJECT 3.2-CONVERTING ANALOG SIGNALS INTO DIGITAL DATA

The computer system is a digital electronic

```
5 X=16777216:Y=65535:Z=255:POKE64875,255:PRINTCHR$(147)
10 PRINT"A PLUS/4 TIMER PROGRAM":PRINT" "
20 PRINT"TO START THE TIMER - PUSH THE (G) KEY":PRINT" "
25 PRINT"TO STOP THE TIMER - PUSH PB-1":PRINT" "
30 PRINT"READY TO START - PUSH  (G)":PRINT" "
35 GETA$:IF A$="G"THEN GOTO 40
36 GOTO35
40 SYS28672
50 A=PEEK(28927):B=PEEK(28926):C=PEEK(28925):D=PEEK(28924)
60 FOR I= 0 TO 500: NEXT
70 A=255-A:B=255-B:C=255-C:D=255-D
80 W =(A)+(B*Z)+(C*Y)+(D*X)
90 T=W*1.59416303E-5
100 T1=INT(T*10000)/10000
110 PRINT" ":PRINT"THE TIME PERIOD IS - ";:
120 PRINTT1;:PRINT" SECONDS"
```

Program 3-8. This BASIC program uses the machine-language subroutine in Program 3-9 to measure time intervals with a PLUS/4 computer.

```
. 7000  EA        NOP
. 7001  EA        NOP
. 7002  EA        NOP
. 7003  EA        NOP
. 7004  EA        NOP
. 7005  A9 FF     LDA #$FF
. 7007  8D FC 70  STA $70FC
. 700A  8D FD 70  STA $70FD
. 700D  8D FE 70  STA $70FE
. 7010  8D FF 70  STA $70FF
. 7013  CE FF 70  DEC $70FF
. 7016  D0 10     BNE $7028
. 7018  CE FE 70  DEC $70FE
. 701B  D0 0B     BNE $7028
. 701D  CE FD 70  DEC $70FD
. 7020  D0 06     BNE $7028
. 7022  CE FC 70  DEC $70FC
. 7025  D0 01     BNE $7028
. 7027  60        RTS
. 7028  2C 10 FD  BIT $FD10
. 702B  30 E6     BMI $7013
. 702D  60        RTS
. 702E  00        BRK
```

Program 3-9. This machine-language subroutine is used with BASIC Program 3-8.

system that uses logic ONEs and ZEROs to do its work. This means that everything that happens inside a digital computer happens either at a logic ONE level (around a positive 5 volts) or at a logic ZERO level (0 volts). All of the experiments that we have presented to this point have interfaced either a logic ONE or ZERO to the USER port of your computer. But to really use your computer to do experiments in the areas of science and technology, you must be able to interface your computer to an analog electronic environment that is based on uniformly changing signals such as a simple sinewave. It is easy to realize that a sinewave is not too compatible with a computer's digital logic electronic system. But, with all of the advances in the electronic world, there are IC circuits called *analog-to-digital converters* (ADC) that will convert an analog signal into a digital signal that is proportionally equivalent in magnitude value. These ADC circuits are very easy to interface to the Commodore computers using either the EXPANSION or USER PORT. The ADC circuit that will be described now will interface into the computer through the USER PORT.

The ADC IC that was chosen for this project is the ADC0809. This IC is a complete electronic ADC system on one IC chip. The only additional circuit that is needed for this application is an external clock oscillator. The completed ADC circuit as shown in Figs. 3-2 and 3-3 can be built for about $25 (1985 prices). The ADC0809 can be purchased at most electronic hobby stores or at several electronics mail order houses that advertise in the leading electronics magazines.

Two circuits will now be presented for the VIC-20 and the C-64 computers. At the time of writing this book, there was not enough information released on the PLUS/4's USER PORT to build an ADC circuit for this computer. When the data is published, it should be a very simple conversion project that requires only the proper connections to the USER PORT pins. The schematic of Fig. 3-4 shows the ADC wiring connections for the VIC-20, and Fig. 3-5 shows the wiring connections for the C-64 version. The two ADC circuits are the same with the exception of the edge connector pin connections for the OUTPUT ENABLE and START CONVERSION signals.

The ADC0809 IC, being a stand alone ADC chip, needs only the control logic and clock signals to function. Since there is no clock signal available at the USER PORT, a simple clock oscillator was built from a TTL 7402 IC. The only requirement of this oscillator is that the oscillation frequency should be between 1.0 and 1.2 MHz. Looking at Figs. 3-4 or 3-5, the oscillation frequency is controlled by capacitor C2. A .001 μF capacitor will get you close to the oscillation frequency, but C2 will most likely need to be a little lower. The ADC0809 requires about 60 clock cycles to complete an *AID* conversion and so, the faster that you run the clock oscillator, the faster the AID converter will work. I have seen ADC0809 ICs that would work with clock frequencies as high as 2 MHz.

In this ADC application, only one input *AID* channel is used and so the other seven inputs are grounded to keep down the oscillation tendency of the IC chip. The circuit is really simple and the only

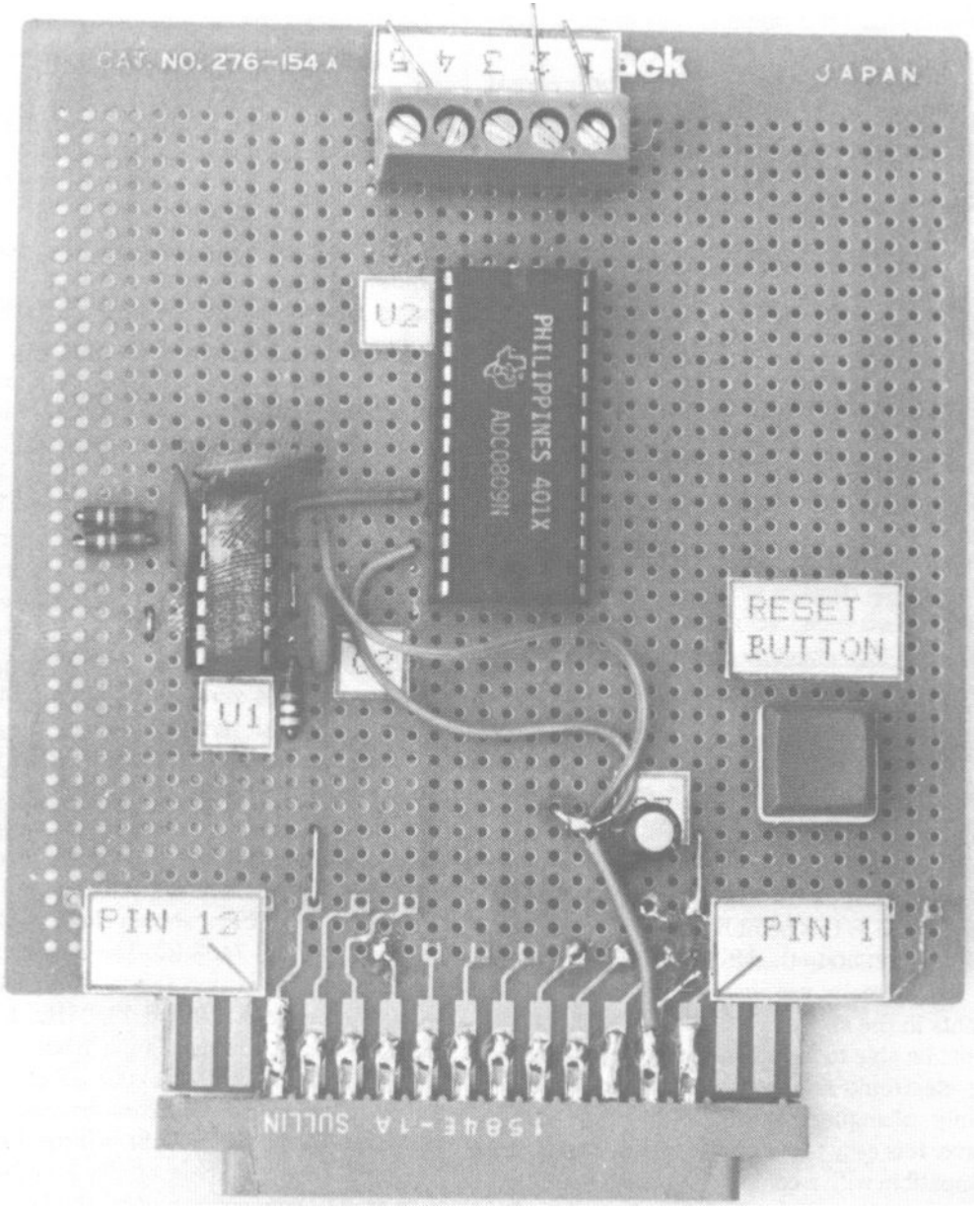


Fig. 3-2. A pictorial view of the top-side of the NO converter board.

problems that you should encounter will be wiring errors, so go slow and double check everything. The circuit is built on a Radio Shack multi-purpose circuit board. Figure 3-6 shows how the circuit board is modified and connected to the edge-card connector. If you can not find a 24-pin edge-connector socket, you can make one from a 44-pin socket by simply cutting it down to size and reconnecting the end with super glue. This is what was done to the connector socket that is shown in Figs.

3-2,3-3, and 3-6. Tables 3-1 and 3-2 present a step-by-step procedure for building this ADC board for the VIC-20 and the C-64.

When your board is completed, check it over very carefully for solder shorts. Of all the problems that one will have with this type of circuit-board construction 99% will be solder bridges between circuit pads or broken connecting wires. When you have built your ADC board following the instructions in Table 3-1 or 3-2, you can safely test out your ADC board using your computer's USER PORT. Figure 3-7 presents a test circuit that can

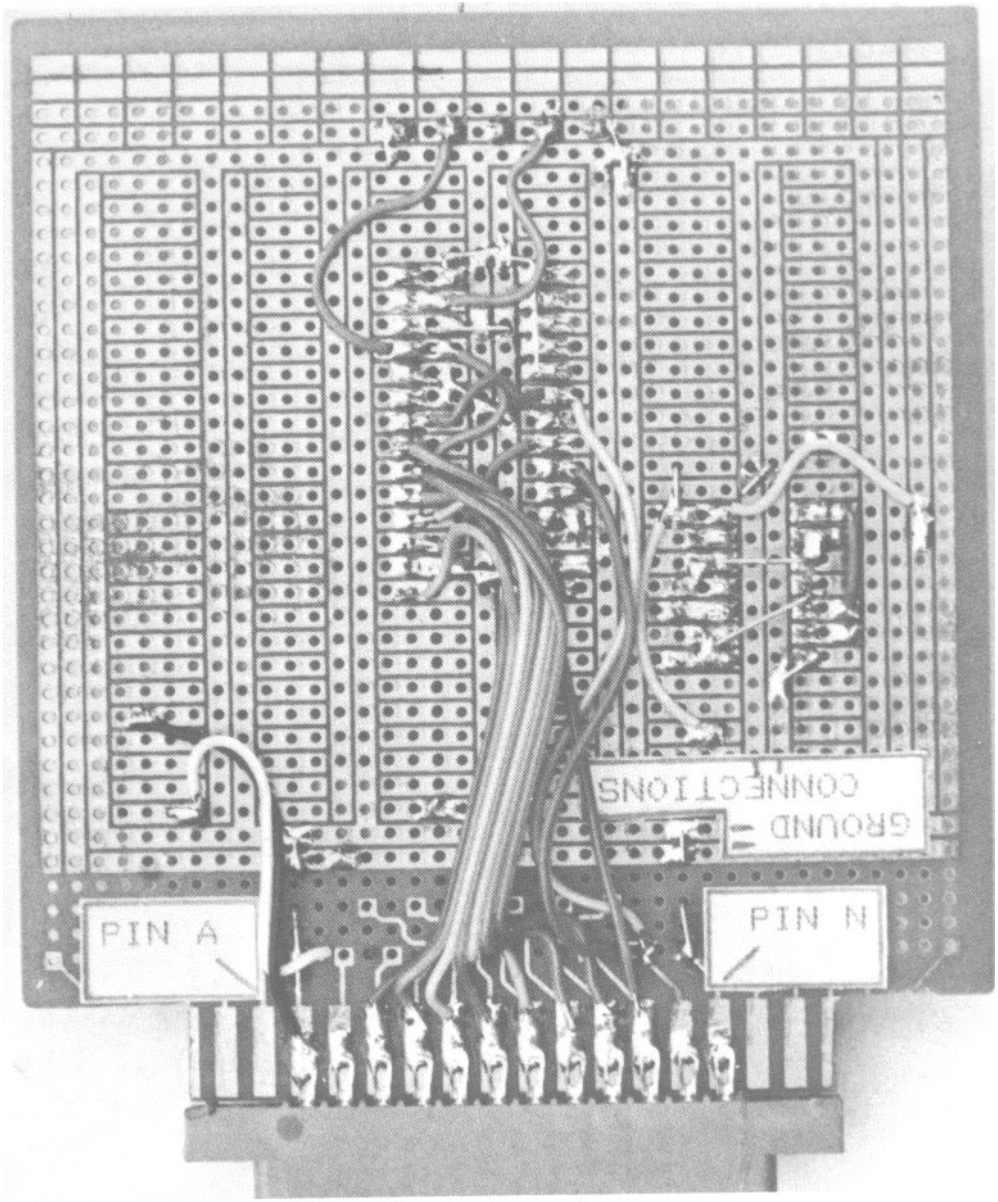


Fig. 3-3. Pictorial view of the bottom-side of the AiD converter board.

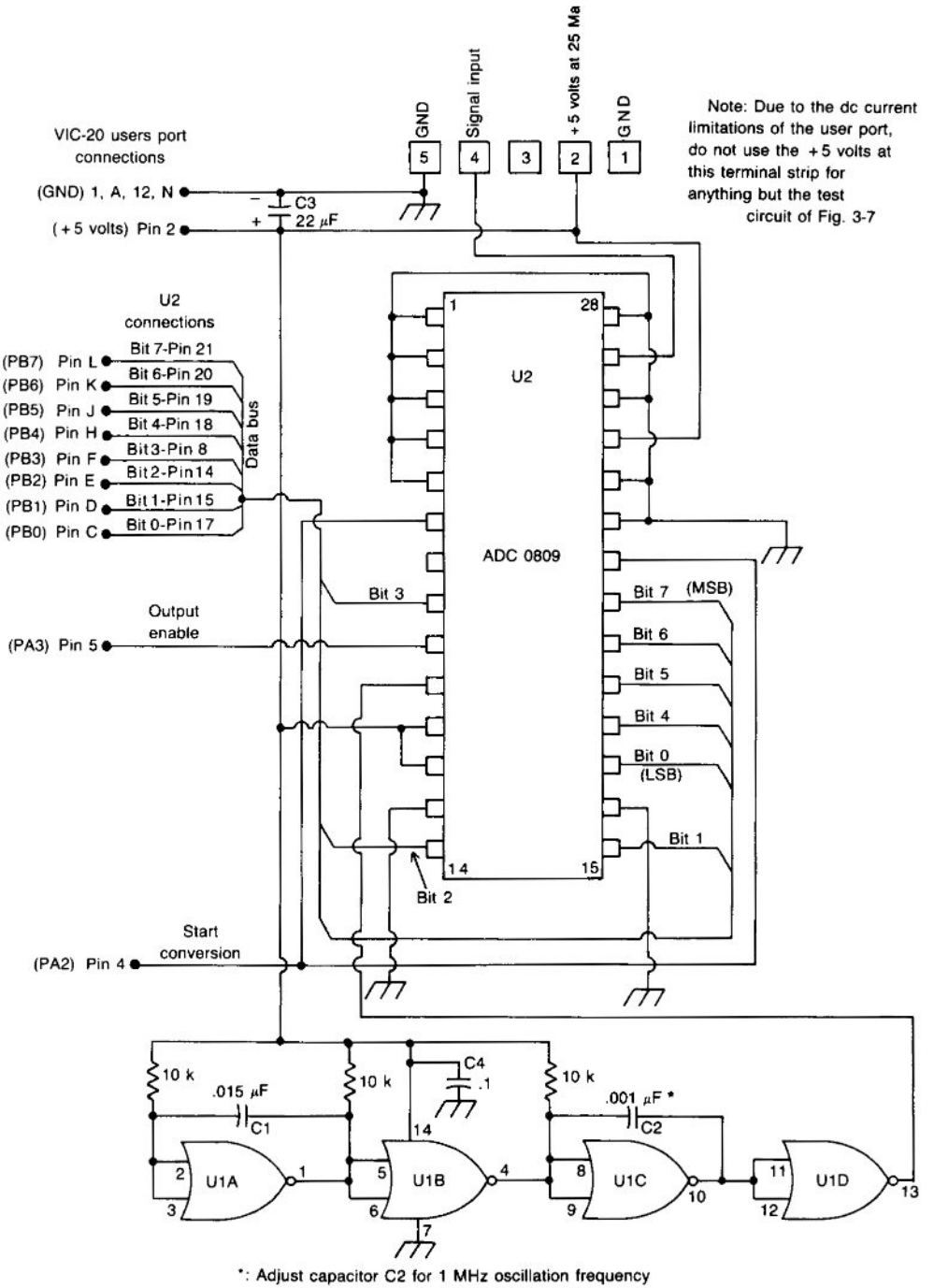


Fig. 3-4. The schematic for the VIC-20 *ND* converter board.

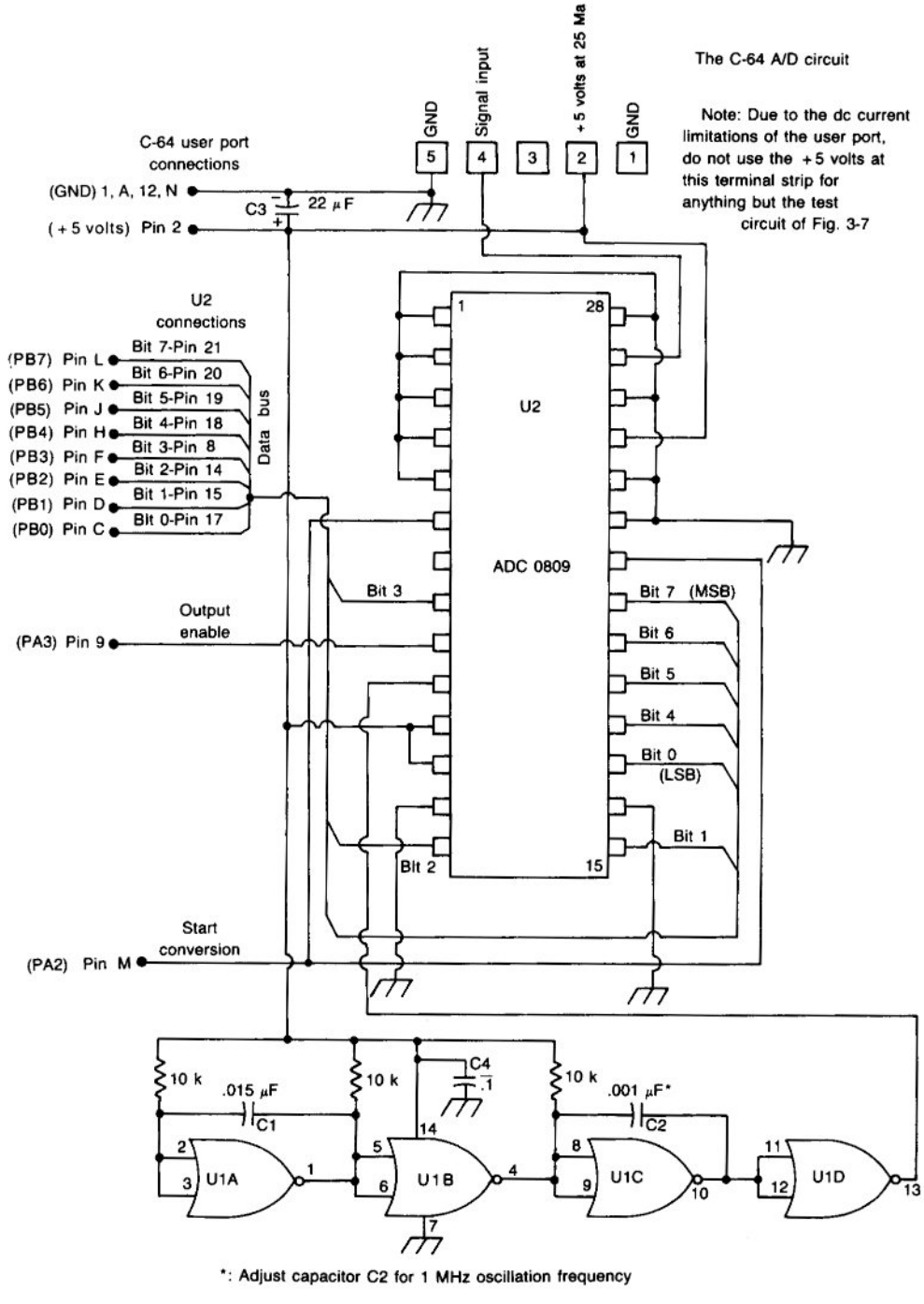


Fig. 3-5. The schematic for the C-64 AID converter board.

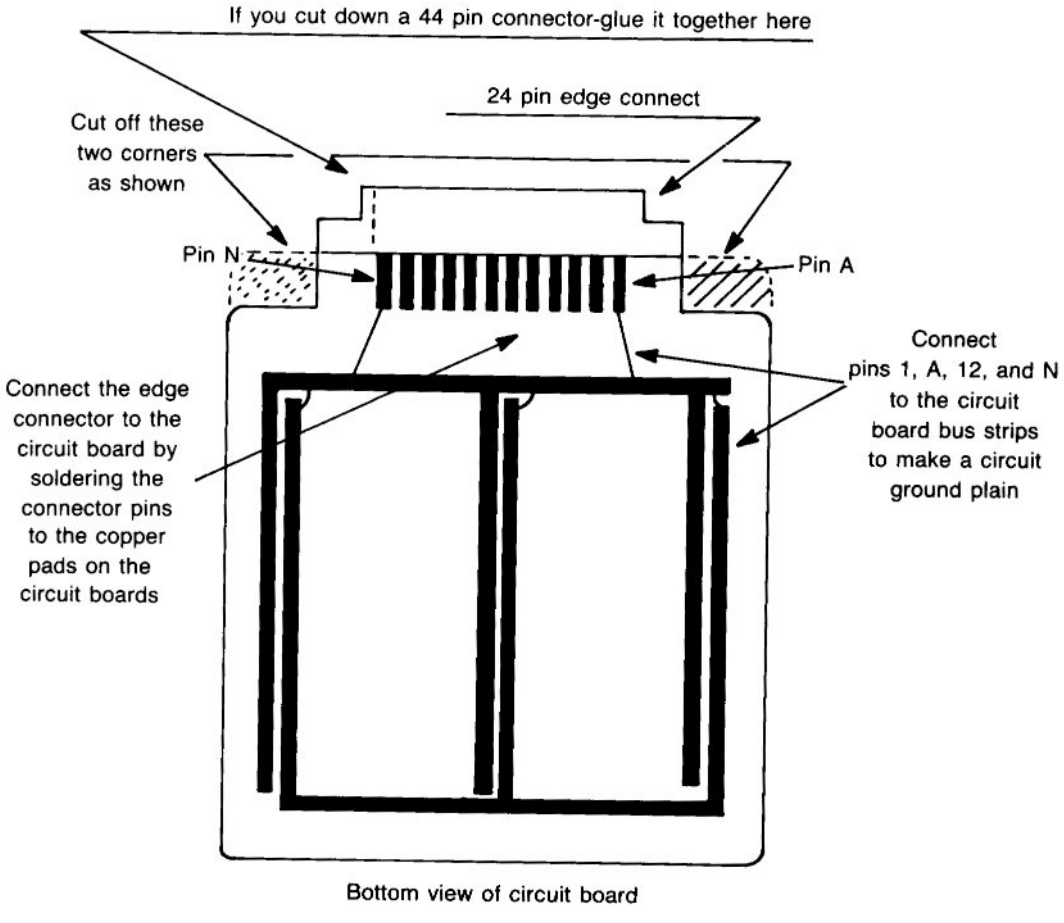


Fig. 3-6. How the edge connector is fastened to the modified *A/D* circuit board.

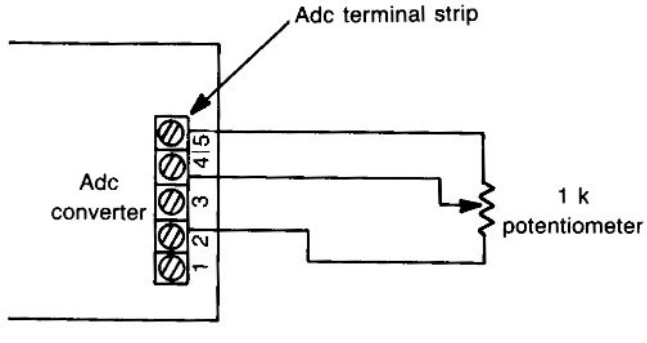


Fig. 3-7. How a potentiometer can be connected to the *A/D* converter to test the circuit board and demonstrate the voltmeter program.

Table 3-1. **Step-by-Step Instructions on Building the AID Converter** Circuit.

Step #1 - Modify the circuit board and connect it to the edge connector socket as shown in figure 3"6.

STEP #2 - Connect the RESET button, the terminal strip, and the two Ie sockets to the circuit board.

STEP #3 - Install capacitor C3 a 22 uF electrolitic capaci tor'.

STEP #4 - Connect ground pins 1 and A - and pins 12 and N together.

STEP #5 - Connect these ground pins to the long copper strips that run between the IC sockets for the circuit board's ground plain.

STEP #6 - Install the two IC sockets and connect pin 7 of IC UI socket to the two copper strips that run between the pins.

STEP #7 ....Connect the following U2 socke·t pine. to the ground strips that run between the pins. PINS 1, 2, 3, 4, 5, 14, 16, 23, 24, 26, 27, and 28.

STEP #8 - Connect the 5 volts supply voltage from the edgE ConnI£ector' pin #2 to Ie U2 pins 11..12, cHId 25"~ and then to IC Ul Fin 14t.

STEP #9 - Connect the RESET button up by soldering two wi.r··es to the RESET Lutton and the'n s.OldeIring Ithe othe'Ir' of one of them to pin A and the other to pin 3.

STEP #9 ..Us..E a VOLT···O'HMI meter and meaS·ur·e ihe 1e'itance· between pins 1 and 2 of the edge connector socket. This measurement should be over 50,000 ohms after capacitor C3 has charged up. If the ·r·esis·ta.nce reading is. lowe'I'··check and make sure that capacitor C3 is not backwards *01"* that no solder bridges exist.

STEP #10 .- If your board passes the resis.tanCe check, thEn plug the board into the USER PORT on your computer and tuy n on thE' com,put e'r.' T'A FE' a tew lEt t er's on thE': screen and then press the RESET button to see that the computer resets itself.

STEP #11 - Now construct the oscillator circuit of IC U1. When you are finished, make another rEsistance check between Fins 1 and 2 to make sure that you did not short the plus 5 volts bus line. Next plug your circuit board back into the USER POFT and turn on the computer. If you have an oscilloscope, you can check the clock oscillator out by observing the waveform at pin 13. If you do not have a scope, try and use a small AM radio to pick-up the oscillator signal around the 1000 point on your dial. If your circuit works, you should be able to hear it some where between 800 and 1200 on the dial. The oscillatior frequency does not have to be adjusted at this time unless you want to do it.

STEP #12 - Connect the rest of the wires to Ie U2 and check the completed board for solder bridges. Check to make sure that the DATA pins are connected as tollows.

USER PORT SOCKET	PIN	IC U2 FIN
	C	17
	D	15
	E	l·4
	F	8
	H	18
	J	19
	K	20
	L	21

STEP #13 - AGAIN, lecheck all connections for 131 oken wires or solder bridges.

NOTE: About the only way that you can damage your computer is to short out the pOwer· s.upply for a per iod of time that is longer than a fellw seconds. Other wiring errors should not cause any damage.

be used with Program 3-10 for the VIC-20 or Program 3-11 for the C-64. When you run either of these two programs, you will realize how valuable an AID converter can be to a computer system.

Programs 3-12 and 3-13 are simple ADC applications that turn your computer into a simple 0 to 5 volts voltmeter. Since the resolution of the ADC is eight bits, the measuring voltage range is divided into 256 parts. This means that the voltage reading that is displayed on the video screen can have an error of plus or minus .0195 volts (5 volts/ 256 steps) from the actual voltage.

Table 3-2. Construction Steps to Follow when Building the Universal Op-amp Circuit on an Experimenter's Board.

1. Secure a Radio Shack or equivalent experimenter's building board"

2" Select which long copper strip will be the positive voltage strip and which strip will be the ground or common striP"

3. Install R5 the 200 ohm resistor (You can use two 100 ohms resistors if needed). Install capacitor Cl and zener diode 01 on the board. Connect the positive point of Dl and Cl to the positive copper strip.

4. Connect the RED wire of a 9 volts transistor battery connector to the unused end of resistor R5 and the BLACK wire to the common copper strip.

5. Connect a 9 volts transistor battery to the connector and measure the voltage at test point "V" which is the positive copper striP" It should be 5.6 volts plus or minus a small amount" Now disconnect the battery.

6. Solder an eight pin IC socket onto the board. Connect pin 4 to the common copper strip and pin eight to the positive copper strip.

7" Reconnect the transistor battery and measure the voltage at pin 8 of the IC socket" It should be 5.6 volts. Check the voltage at pin 4. It should be zero. Disconnect the battery"

8. Install trimpots Rl, R2, R3, and R4 and adjust them to their midpoints.

9" Install all remaining components and interconnecting wires but do not install the IC chip.

10. Reconnect the transistor battery and measure the voltage at test point "V", pin 8, and pin 4. They should be the same as before"

11" Disconnect the battery and install the Ie Chip and test out the circuit as described in the text.

```
5 REM PGM 3.10
10 REM A VIC-20 ADC TEST PGM
15 POKE 37139,255
20 POKE 37137,151
30 POKE 37137,155
35 PRINTCHR$(147)
40 PRINTPEEK(37136)
50 GOTO20
```

Program 3-10. This BASIC program is used to test the VIC-20 A/D converter circuit.

```
5 REM PGM 3.11
10 REM A C-64 ADC TEST PGM
20 POKE 56576,155
30 POKE 56576,151
40 PRINTCHR$(147)
50 PRINTPEEK(56577)
60 GOTO 20
```

Program 3-11. This BASIC program is used to test the C-64 A/D converter circuit.

PROJECT 3-3-A UNIVERSAL OP-AMP CIRCUIT FOR TEMPERATURE MEASUREMENTs AND OTHER APPLICATIONS USING THE AID CONVERTER IN PROJECT 3-2.

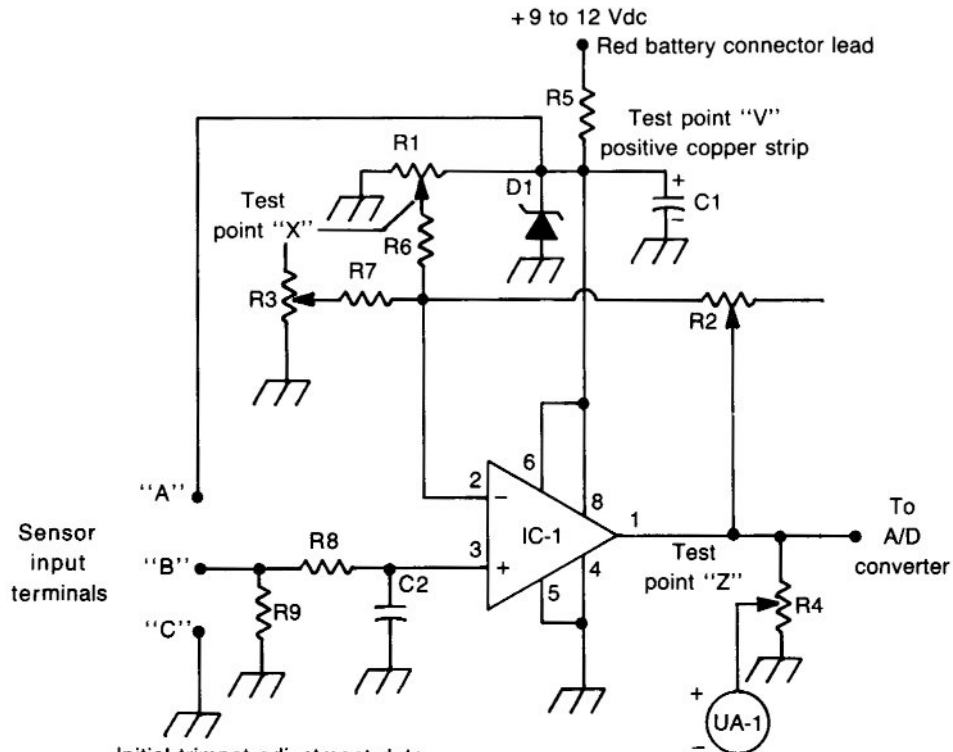
The AID converter in Project 3-2 is designed to operate with signal inputs that range from zero to five volts. The *AID* 0 to 5 volts input range is fine if all of the signals that you wanted to digitize were in the range of zero to five volts. You will find many applications, though, where the signal to be digitized into its equivalent binary form for further computer processing does not have the full dynamic voltage range required for the *AID* converter. For these applications, a universal op-amp circuit is presented in Fig. 3-8. This noninverting op-amp circuit can be adjusted to handle a wide range of signal applications that require a voltage amplification to secure an analog signal with an amplitude range of zero to five volts. The circuit is designed to be a low-noise low-frequency amplifier with a gain that can be adjusted to operate from about "I" to over 80 with good linearity. This means that an analog

```
5 REM PGM 3.12
10 REM A VIC-20 VOLTMETER
15 POKE37139,255
20 POKE 37137,151
30 POKE 37137,155
40 A=PEEK(37136)
50 B=A*.0196078431
60 C= INT(B*100)/100
70 PRINTCHR$(147):PRINTC;:PRINT" VOLTS DC
80 GOTO20
```

Program 3-12. This BASIC program can be used to turn the VIC-20 into a zero- to five-volt voltmeter.

```
5 REM PGM 3.13
10 REM A C-64 VOLTMETER
20 POKE 56576,155
30 POKE 56576,151
40 A=PEEK(56577)
50 B=A*.0196078431
60 C= INT(B*100)/100
70 PRINTCHR$(147):PRINTC;:PRINT" VOLTS DC
80 GOTO20
```

Program 3-13. This BASIC program can be used to turn the C-64 into a zero- to five-volt voltmeter.



Initial trimpot adjustment data

R1: Adjust test point "X" to equal the input "B" terminal voltage.

R2: Adjust to the minimum resistance which gives the required op-amp gain.

R3: Adjust to the maximum resistance which gives the required op-amp gain.

R4: Adjust for correct meter reading.

Note: With no input signal, the output meter should read zero.

Universal Op-Amp Parts List

		Radio Shack P/N			
Capacitor	C1 - 1000 μF	- 272-1032	Trim pot	R1 - 10 k	- 271-218
	C2 - .05 μF	- 272-157		R2 - 500 k	- 271-221
Resistor	R5 - 150 Ohm	- 271-013		R3 - 10 k	- 271-218
	R6 - 33 k	- 271-040		R4 - 50 k	- 271-219
	R7 - 1 k	- 271-023	Meter	UA-1 - 50 μA	- 271-1751
	R8 - 6.8 k	- 271-032	Op-Amp	IC-1 - TLC-272	- 276-1749
	R9 - 150 k	- 271-047	Circuit Board	----------	- 276-1749
Zener Diode	D1 - 6.8 Volt	- 276-561	9 Volt Battery Connector		- 270-325
			IC Socket	----------	276-1995

Fig. 3-8. This is the schematic for the universal op-amp. This amplifier is very handy for amplifying dc or low-frequency signals that are encountered when using electronic sensors to make physical measurements such as temperature, pressure, and force.

signal that is less than .10 volts peak-to-peak can be amplified to a full five-volts peak-to-peak analog signal.

Looking at Fig. 3-8 shows that the op-amp circuit is built around the Radio Shack TLC27M (276-1749) op-amp IC. The op-amp, shown in Figs. 3-9 and 3-10, is designed to function with a 9-volt transistor battery which makes the amplifier self contained and portable. Resistor R4, capacitor Cl, and diode Dl are used to provide a constant 5.6-volts supply voltage to the op-amp and the input sensor terminals A, B, and C. The input to the op-amp is connected to terminal "B." The input operating point of the amplifier is adjusted by trimpot Rl, and the op-amp gain is adjusted by the setting of trimpots R2 and R3. The microammeter

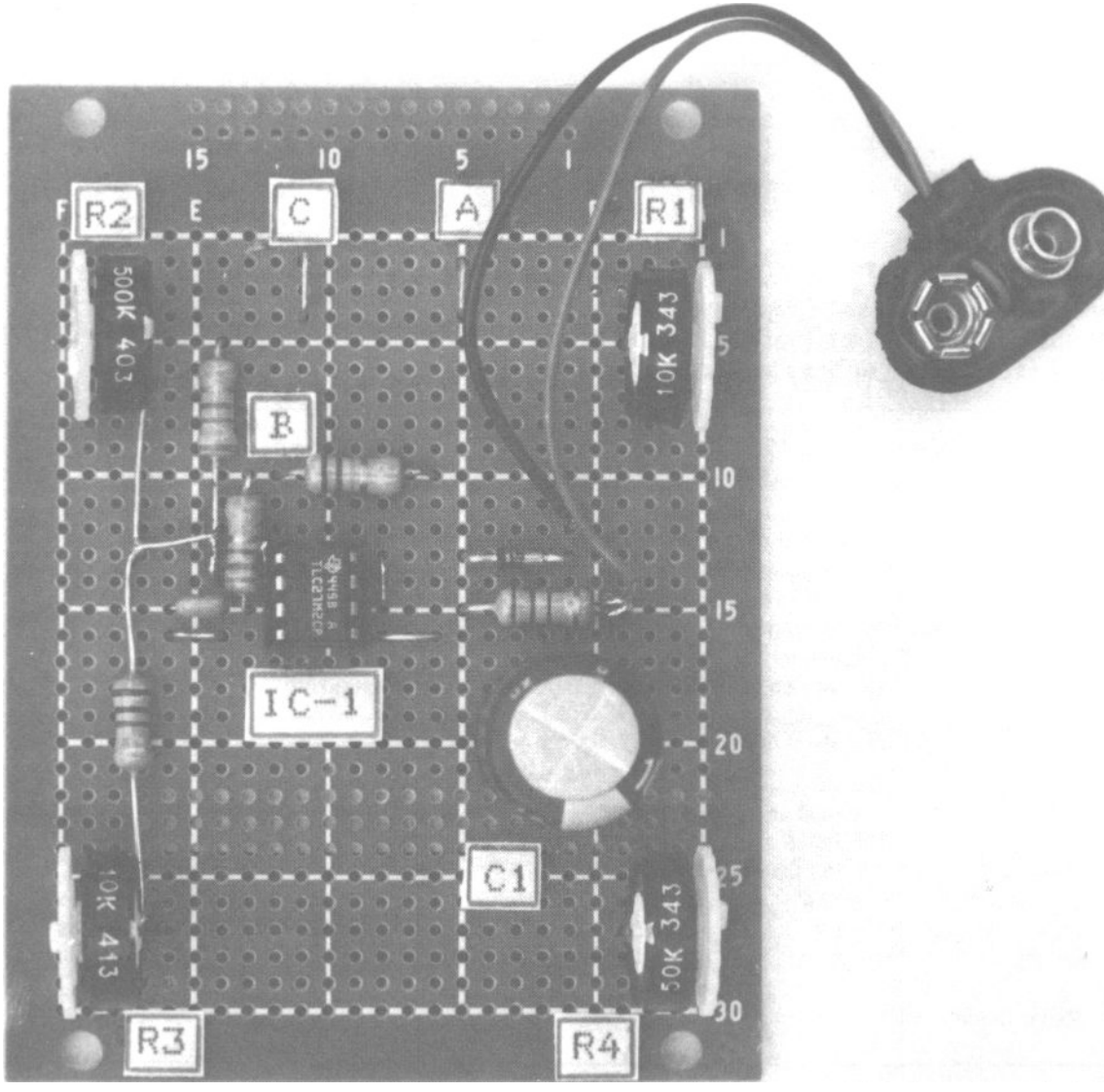


Fig. 3-9. This is the top-view of the universal op-amp circuit board.

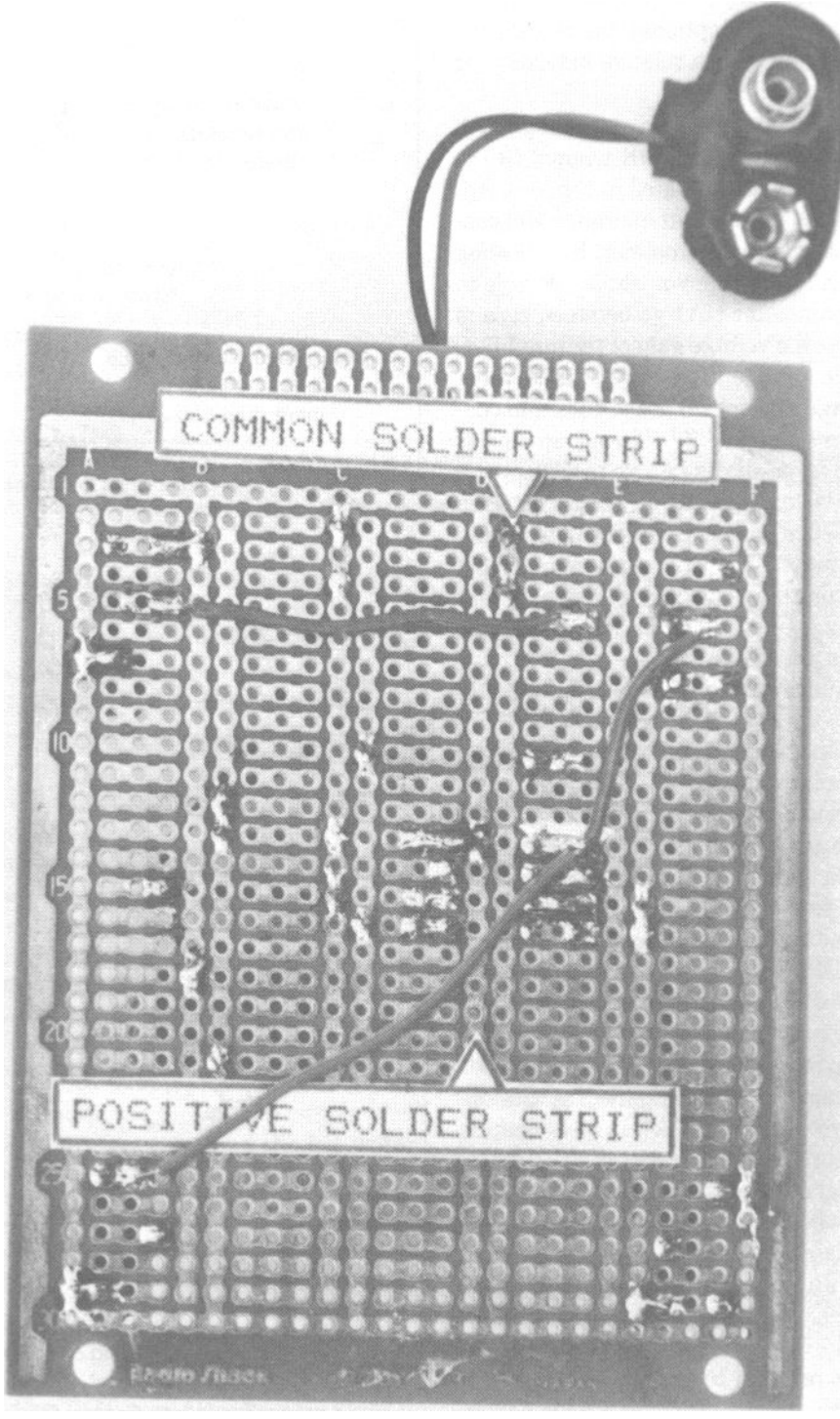


Fig. 3-10. This is the bottom-view of the universal op-amp circuit board.

UA1 and trimpot R4 are optional, but when they are used, they do give you a relative indication of the output signal level.

Testing the op-amp circuit is very simple. All you need to do is to connect a 10K trimpot (Rt) to terminals A, B, and C as indicated in Fig. 3-8. Adjust Rl, R2, R3 and R4 to about midrange and connect the 9-volt battery to the amplifier. By adjusting trimpot R_t around midscale, you should be able to make the microammeter UA1 go between zero to full scale. To test the voltage gain of the amplifier, adjust Rl so the voltage at point "X" is .2 volts. Adjust the voltage at point "B" to .4 volts and connect a voltmeter to point "Z". If your op-amp is working properly, you should be able to adjust trimpot "R2" so the voltage at point "Z" varies between .2 and 5 volts. When you have the circuit working, use the test trimpot "Rt" to learn how to adjust the op-amp circuit to handle different signal input levels, operating points, and gains. After you have played with this circuit for awhile, you will have a much better understanding of how an op-amp works.

Now, what can you do with the universal op-amp circuit? Well, as an example, Fig. 3-11 A, B, and C shows three methods that can be used to turn our universal op-amp circuit into a thermometer by using a diode, a transistor, or a thermistor as a temperature pick-up sensor.

The thermistor is a resistor that changes resistance with temperature. Figure 3-9A shows how to connect a thermistor so the thermistor's changing resistance value will develop a varying voltage that can be applied to the op-amp input at terminal "B." This voltage change is then amplified by the op-amp and displayed on meter UA1 as an indication of the temperature of the thermistor.

In recent years, the most common temperature sensor has been the silicon semiconductor junction. The forward biased diode has a voltage drop of about .6 volts with a current of about 10 milliamps and generally behaves like a 10-ohm resistor in series with a - .45 volt battery. The - .45 volt so-called battery is referred to as the band-gap voltage which changes with temperature. The band-gap voltage of the semiconductor junction increases as

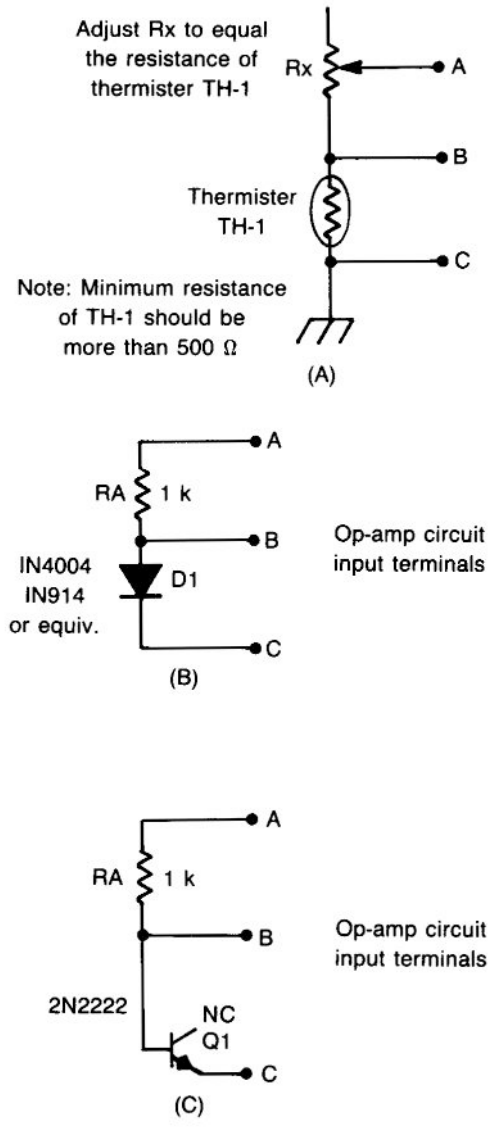


Fig. 3-11. Here are three methods that can be used to measure temperature using an electronic component as a sensing device with the universal op-amp.

the temperature goes down and decreases when the temperature goes up. The band-gap voltage change which is about 2 mV per degree C is what makes the diode a good temperature sensor. The most im-

```
1 REM - PROGRAM 3.14 FOR THE C-64
5 PRINTCHR$(5) :PRINTCHR$(147)
10 B=0
15 FOR I=1 TO 10
20 POKE56576,155
25 POKE56576,151
30 A=PEEK(56577)
40 B=A+B
50 NEXTI
60 C=B/10:C=INT(C)
70 D=(C*.0196078431)
80 E=(D-.45)/.041:E=INT(E)
90 PRINTCHR$(19);:PRINT"      ";:PRINTCHR$(19);:PRINTE;:PRINT"   DEGREES      "
100 GOTO10
READY.
```

Program 3-14. This C-64 program can be used to produce a video display of the temperature that is measured by a diode sensor connected to the universal op-amp and the *AID* converter of Project 3-2. Lines 70 and 80 may need to be adjusted a little to match the characteristics of the diode that you use. See the text.

portant parameter to control when using the diode as a temperature sensor is the dc current that flows through the diode. The dc current must turn the diode fully on and be constant.1

The easiest way to use the universal op-amp circuit in a temperature-sensing application is to adjust all of the op-amp trim pots to their midpoints. Next connect the diode circuit of Fig. 3-11B to the input terminals A, B, and C. Connect a 9-volt battery to the op-amp and then adjust trimpot Rl for a midscale reading on meter UA-1. You should be able to dip the diode sensor into ice water and then hot water and watch meter UA-1 go back and forth between 0 and 100. Next, adjust trimpot R4 until there is no reading on meter UA-1 and then adjust the trimpot back for about 1 1/2 turns. Readjust trimpots R2 and R3 until you can secure a 0- to 5-volts from the output of the op-amp circuit when you go between the ice water and the hot water. When you have secured the 0- to 5-volts range, readjust trimpot R4 so that a 5-volt output from the op-amp will give you a full scale meter deflection. You can now calibrate meter UA-l to read degrees if you know the temperature of the ice water and hot water. The circuit of Fig. 3-11C uses a 2N2222 transistor in place of the diode. The operation of the circuit in Fig. 3-11C is just the same as in Fig. 3-11B, but the transistor is a little slower reacting to temperature changes.

If you have built the *AID* converter circuit of Project 3-2, don't use meter UA-l to read the temperature because you can use the computer to do the job better and faster. Adjust the op-amp circuit to generate output voltages of 5- and 0-volts when you dip the semiconductor sensor in the ice at -10 degrees F. and hot water at 110 degrees F. Program 3-14 is written for the C-64 and Program 3-15 is written for the VIC-20 to indicate temperatures between -10 to 110 degrees F. if the universal op-amp circuit is adjusted as per the above instructions. **If** you can not secure a temperature of - 10 degrees, then use 3.2 volts for 32 degrees F.2

PROJECT 3-4-AN ANALOG WAVEFORM RECORDER

Recording an analog waveform by using an *AID* converter and a computer gives one the ability to display the recorded waveform on the computer's

(1) Kuecken, *How To Measure Anything With Electronic Instruments,* TAB Books, Inc.: 1981, p. 213.

(2) Salt water and ice will go below 32 0F if the ice is frozen to below -20°F.

```
1 REM - PROGRAM 3.15 FOR THE VIC-20
5 PRINTCHR$(5) :PRINTCHR$(147)
10 B=0: POKE37139,255
15 FOR I=1 TO 10
20 POKE37137,151
25 POKE37137,155
30 A=PEEK(37136)
40 E=A+B
50 NEXTI
60 C=B/10:C=INT(C)
70 D=(C*.0196078431)
80 E=(D-.45)/.041:E=INT(E)
90 PRINTCHR$(19);:PRINT"      ";:PRINTCHR$(19);:PRINTE;:PRINT"   DEGREES      "
100 GOTO10
READY.
```

Program 3-15. This VIC-20 program can be used to produce a video display of the temperature measured by a diode sensor connected to the universal op-amp and the *AID* converter of Project 3-2. Lines 79 and 80 may need adjusted a little to match the characteristics of the diode that you use. See the text.

video monitor. It's sort of like having a storage oscilloscope that lets you store the waveform on the screen so you can observe that waveform at a later time. The analog waveform recorder's actual job is to sample the analog waveform at the AID converter's input, convert those samples into a representative digital format, and store the digital representations of the sampled waveform in the computer's memory. A few years ago, a good digital waveform recorder cost $50,000 or more based on how fast it would record a signal, but prices in recent years have dropped, just like other computer prices. Project 3-4 presents four analog waveform recording programs for the C-64, which along with the AID converter of Project 3-2 and the universal op-amp of Project 3-3 can be used to make very accurate low-frequency waveform recordings.

A Block Diagram of the analog waveform recorder (AWR) for this project is shown in Fig. 3-12. This AWR is an elementary version, but it can record a low-frequency waveform, display that waveform on the computer's monitor in low-resolution or high-resolution graphics, and print out a hardcopy of the waveform on the computer's printer. The recorded digital data can also be used to do low-frequency signal analyzation. Four AWR programs will be described for Project 3-4. Two programs will use a machine-language subroutine to record fast waveforms between one and 500 Hz and the other two are all BASIC programs for recording slowly progressing waveforms of one hertz or lower. Also, two of the programs are designed to use low-resolution graphics for the waveform display while the other two use high-resolution graphics with the help of SIMON'S BASIC. The two low-resolution programs will be described as five independent program routines that can be put together as required for the specific waveform recording application. The five routines are the "A" BASIC control routine, the "B" BASIC control routine, the waveform display routine, the hard copy screenprint routine that can be used with any Commodore printer, and the machine language subroutine that is used with "B" BASIC control program. The two SIMON'S BASIC AWR programs use the same control routines as the low-resolution programs, but the waveform display is generated using Simon's high-resolution graphics commands.

The "A" BASIC control routine is located between lines 1 and 300 of waveform recording Program 3-16. The routine is used with the waveform display and screen print routines to make the all BASIC waveform recording program. The all BASIC waveform recording program is used to digitize (record in memory) slowly progressing waveforms. A temperature curve or the movement of a long oscillating pendulum are examples of

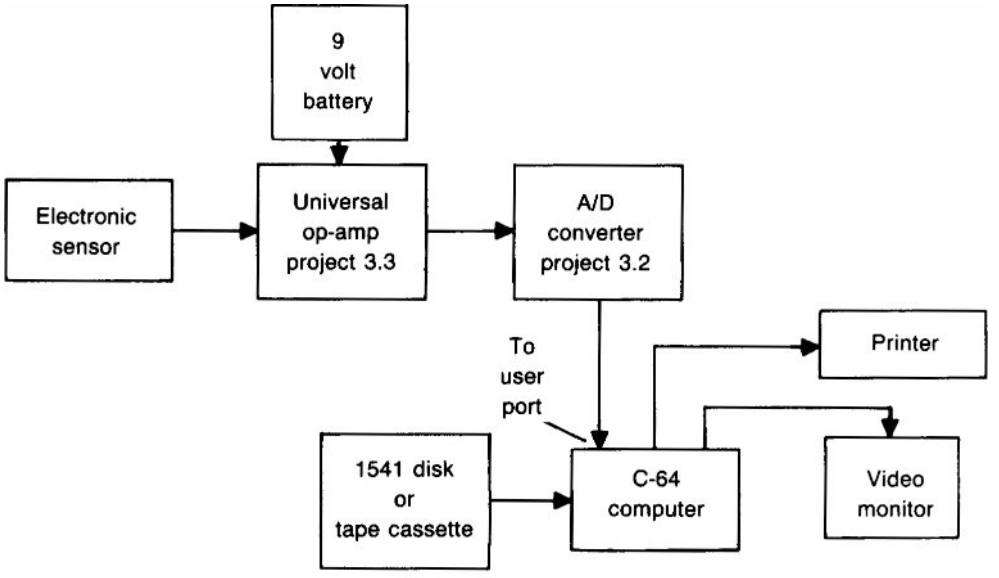


Fig. 3-12. Basic block diagram of the simple waveform recorder.

```
1 REM - PROGRAM 3-16 - ALL BASIC CONTROL PROGRAM
5 POKE53281,1
10 DIM FZ(75)
15 PRINTCHR$(147):SQ=2
20 PRINT"      A WAVEFORM RECORDING PROGRAM"
25 PRINT" "
30 PRINT"    THIS IS AN A/D CONVERTER WAVEFORM   RECORDING  PROGRAM THAT WILL";
35 PRINT"  SECURE 64 WAVEFORM DATA POINTS OVER A GIVEN PERIODOF TIME WHICH";
40 PRINT" IS UNDER  PROGRAM CONTROL.THE RECORDED  DATA MAY THEN BE USED TO ";
45 PRINT" GENERATE A GRAPHICAL DISPLAY OF THE     WAVEFORM."
50 PRINT" ":PRINT"INPUT THE TIME DATA - ";: INPUT J1
55 PRINT" ":PRINT"PRESS <R> TO START RECORDING"
60 GET A$: IF A$="R" THEN GOTO 100
65 GOTO 60
100 PRINT" ":PRINT"RECORDING":PRINT" "
120 FOR I=0 TO 65
125 FOR H=0 TO J1: NEXT H
130 POKE 56576,155:POKE56576,151:FM=PEEK(56577)
135 FM=INT((FM*.01953125)*1000)/1000
140 AM=AM+1
145 FZ(I)=FM
150 NEXT I
200 PRINTCHR$(147)
220 PRINT" ": PRINT"THE 64 PLOT POINTS ARE -":PRINT" "
230 FOR I=0 TO 63
240 PRINTFZ(I),;
250 NEXTI
275 IF SQ=2 THEN GOTO290
```

Program 3-16. This is the "A" BASIC waveform recording program.

```
280 GOSUB 8000
285 PRINT"PRESS  W  TO SEE WAVEFORM."
290 PRINT" ":PRINT"PRESS ANY KEY TO EXIT GRAPH WHEN        FINISHED"
295 GET A$:IF A$="W" THEN GOTO 300
296 GOTO295
300 GOTO2000
2000 AE=0: REM GRAPH
2003 REM -- GRAPH ALGORITHM BY RUGG AND FELDMEN --(SEE REFERENCE)
2004 REM -- GRAPH ALGORITHM CONVERTER FOR WAVE FORM PRESENTATION BY R. LUETZOW.
2005 REM GRAPH ALGORITHM IS IN LINES 2070 TO 2095  AND 2280 TO 2500
2025 PRINTCHR$(147)
2040 GOSUB 2065:GOSUB2245
2043 PRINT CHR$(13)
2045 PRINTCHR$(19):PRINTTAB(26):PRINT"TIME=";:PRINT J1
2046 IF SQ=2 THEN GOTO2050
2047 GOSUB 8000
2050 GET Q$: IF Q$="" THEN 2050
2051 PRINTCHR$(147) :  PRINT"WANT A HARD COPY Y=1 N=2";:INPUTSQ
2052 IF SQ=1 THEN GOTO50
2053 PRINTCHR$(147):PRINT"DO YOU WANT ANOTHER TEST ( Y / N )  ";:INPUT C$
2054 IF C$="Y"THEN GOTO 11
2060 END
2065 PRINT CHR$(147);
2070 POKE 1870,91:XX=40
2075 FOR M=1 TO 32:POKE 1870+M,114:NEXT
2080 FOR M=1TO20:POKE1870-(XX*M),115:NEXT
2085 FOR M=0 TO 32 STEP 02
2090 POKE 1910+M,93:POKE 1869-(XX*M),64
2095 NEXT
2170 POKE1950,48:POKE1954,56:POKE1957,49:POKE 1958,54:POKE1961,50:POKE1962,52
2175 POKE 1965,51:POKE1966,50:POKE1969,52:POKE1970,48:POKE1973,52
2180 POKE 1974,56:POKE1977,53:POKE1978,54:POKE1981,54:POKE1982,52
2185 POKE 1998,20:POKE1999,05:POKE2000,19:POKE2001,20
2190 POKE2003,16:POKE2004,15:POKE2005,09:POKE2006,14:POKE2007,20:POKE2008,19
2195 POKE1075,9:POKE1076,14:POKE1077,16:POKE1078,21:POKE1079,20
2200 POKE1081,23:POKE1082,1:POKE1083,22:POKE1084,5:POKE1085,6:POKE1086,15
2205 POKE1087,18:POKE1088,13
2222 PRINTTAB(42):PRINT"5"
2224 FOR I=1 TO 19:PRINT:NEXTI:PRINTTAB(2):PRINT"0"
2225 POKE1105,19:POKE1145,9:POKE1185,7:POKE1225,14:POKE1265,1:POKE1305,12
2230 POKE1385,9:POKE1425,14:POKE1465,16:POKE1505,21:POKE1545,20
2235 POKE1625,22:POKE1665,15:POKE1705,12:POKE1745,20:POKE1785,19
2240 RETURN
2245 :
2280 FOR N=1 TO 32: GOSUB2375
2285 GG=126: IF E=0 THEN GG=123
2310 POKE 1870+N-(XX*WZ),GG
2315 GOSUB2375
2330 GG=124:IF E=0 THEN GG=108
2365 POKE  1870+N-(XX*WZ),GG
2370 NEXT:RETURN
2375 GOSUB 2500:D= Y/.25  :WZ=INT(D)
2390 E=1:IF (D-WZ))=0.5 THEN WZ=WZ+1:E=0
2395 RETURN
2500 Y =FZ(AE)
2502 AE=AE+1
2600 RETURN
6000 REM SCREEN PRINT FROM COMPUTE! APRIL 1984.
```

```
6010 USED BY PERMISSION FROM COMPUTE! PUBLICATIONS, INC
8000 PRINT CHR$(19);:SS=(PEEK(210))*256:OPEN3,3:OPEN4,4
8110 FOR R= 0 TO 24 : B$=""
8120 FOR C= 0 TO 39 : A$=""
8130 IFPEEK(SS+((R*40)+C))>127THEN:GET#3,A$:B$=B$+CHR$(18)+A$+CHR$(146):GOTO8160
8140 GET#3,A$:IF A$=CHR$(13)THEN:A$=" "
8150 B$=B$+A$
8160 NEXTC:PRINT#4,B$:NEXTR:CLOSE4:CLOSE3
8170 RETURN
READY.
```

slowly progressing analog waveforms. The "A" control routine is designed to record 32 equally spaced waveform samples with a programmable sampling rate that can be set as fast as 15 samples per second to less than 1 per hour according to whatever the need requires. The time interval between these samples is controlled by a FOR-NEXT time-delay loop that uses an INPUT statement so you can enter the time interval data to secure the required sampling intervals. After the time interval has been entered, the program then tells you to push the <R> key to start the waveform recording. After the waveform is recorded, the 32 recorded sample points are shown in a video display format for your observation. If the data is what you wanted to see, you can press <W> to go on to the waveform display. If you want to start over, press the RUN/STOP key. Table 3-3 gives a functional description of program "A" BASIC control routine.

Waveform recording Program 3-17 uses the "B" BASIC control routine that is located between lines 1 and 600. This routine uses a machine-language subroutine to control the *AID* converter's waveform sampling. The sampling rate of this routine can run as fast as eight samples per millisecond. The actual sampling rate will depend upon the clock frequency of the *AID* converter. This control routine is also set-up to secure 32 waveform samples per measurement period, but the main operational difference between the "A" BASIC and the "B" BASIC routines is the way the sampling rate time interval data is entered into the routine. The "A" routine just requested a decimal number of one or over, but the "B" routine must use a number between 09 and 255. The 09 number is used to set-up the fastest sampling rate, so the actual number may be plus or minus a little bit depending upon the clock frequency of the *AID* converter. The "B" routine's operation is the same as the "A" routine after the waveform samples have

Table 3-3. **Where the Main Program Routines are Located In the** "A" **BASIC Program 3-16.**

Lines	1-50: Program set-up and description.
Lines	55-65: Record start routine.
Lines	100-150: The waveform sampling routine.
Lines	200-300: The recorded data display routine.
Lines	2000-2600: Waveform display routine.
Lines	8000-8170: Screen print routine.

```
1 REM - PROGRAM 3.17 - 'B' BASIC CONTROL ROUTINE
2 REM - USE MACHINE LANGUAGE SUBROUTINE PROGRAM 3.18 WITH THIS PROGRAM
5 POKE53281,1 : DIM FZ(75)
10 PRINTCHR$(147):SQ=2
15 PRINT"      A WAVEFORM RECORDING PROGRAM":PRINT" "
20 PRINT"    THIS IS AN A/D CONVERTER WAVEFORM   RECORDING  PROGRAM THAT WILL";
30 PRINT"  SECURE 64 WAVEFORM DATA POINTS OVER A GIVEN PERIODOF TIME WHICH";
35 PRINT" IS UNDER  PROGRAM CONTROL.THE RECORDED  DATA MAY THEN BE USED TO ";
40 PRINT" GENERATE A GRAPHICAL DISPLAY OF THE     WAVEFORM."
45 GOTO500
50 :
100 FOR I=0 TO 65
120 FM=PEEK(49272+AM)
130 FM=INT((FM*.01953125)*1000)/1000
140 FZ(I)=FM
150 NEXT I
250 PRINTCHR$(147)
255 PRINT" ": PRINT"THE 64 PLOT POINTS ARE -":PRINT" "
260 FOR I=0 TO 63
265 PRINTFZ(I),;
270 NEXTI
275 IF SQ=2 THEN GOTO290
280 GOSUB 8000
285 PRINT"PRESS  W  TO SEE WAVEFORM."
290 PRINT" ":PRINT"PRESS ANY KEY TO EXIT GRAPH WHEN        FINISHED"
295 GET A$:IF A$="W" THEN GOTO 300
296 GOTO295
300 GOTO2000
500 PRINT" ": PRINT"ENTER A NUMBER BETWEEN 9 AND 255";:
505 PRINT" ":PRINT"FOR THE TIME DELAY DATA BETWEEN THE 64";
510 PRINT"  RECORDING POINTS."
515 INPUT GD$
520 GT=VAL(GD$)
525 POKE 49178,GT
530 PRINTCHR$(147)
535 PRINT"THE PROGRAM IS NOW READY TO START THE   WAVEFORM RECORDING.":PRINT" "
540 PRINT"PRESS 'R' TO START RECORDING"
545 GET A$: IF A$="R"THEN GOTO555
550 GOTO545
555 POKE 56334,00
560 PRINT"RECORDING"
565 :
570 POKE 56334,01
575 GOTO50
700 PRINTCHR$(147)
2000 AE=0: REM GRAPH
2003 REM -- GRAPH ALGORITHM BY RUGG AND FELDMEN --(SEE REFERENCE)
2004 REM -- GRAPH ALGORITHM CONVERTER FOR WAVE FORM PRESENTATION BY R. LUETZOW.
2005 REM GRAPH ALGORITHM IS IN LINES 2070 TO 2095  AND 2280 TO 2500
2025 PRINTCHR$(147)
2040 GOSUB 2065:GOSUB2245
2043 PRINT CHR$(19)
2045 PRINTCHR$(19):PRINTTAB(26):PRINT"TIME=";:PRINTGD$
2046 IF SQ=2 THEN GOTO2050
2047 GOSUB 8000
2050 GET Q$: IF Q$="" THEN 2050
```

Program 3-17. This is the "B" BASIC program that uses the machine-language subroutine in Program 3-18.

```
2051 PRINTCHR$(147) :  PRINT"WANT A HARD COPY Y=1 N=2";:INPUTSQ
2052 IF SQ=1 THEN GOTO50
2053 PRINTCHR$(147):PRINT"DO YOU WANT ANOTHER TEST ( Y / N )  ";:INPUT C$
2054 IF C$="Y"THEN GOTO 11
2060 END
2065 PRINT CHR$(147);
2070 POKE 1870,91:XX=40
2075 FOR M=1 TO 32:POKE 1870+M,114:NEXT
2080 FOR M=1TO20:POKE1870-(XX*M),115:NEXT
2085 FOR M=0 TO 32 STEP 02
2090 POKE 1910+M,93:POKE 1869-(XX*M),64
2095 NEXT
2170 POKE1950,48:POKE1954,56:POKE1957,49:POKE 1958,54:POKE1961,50:POKE1962,52
2175 POKE 1965,51:POKE1966,50:POKE1969,52:POKE1970,48:POKE1973,52
2180 POKE 1974,56:POKE1977,53:POKE1978,54:POKE1981,54:POKE1982,52
2185 POKE 1998,20:POKE1999,05:POKE2000,19:POKE2001,20
2190 POKE2003,16:POKE2004,15:POKE2005,09:POKE2006,14:POKE2007,20:POKE2008,19
2195 POKE1075,9:POKE1076,14:POKE1077,16:POKE1078,21:POKE1079,20
2200 POKE1081,23:POKE1082,1:POKE1083,22:POKE1084,5:POKE1085,6:POKE1086,15
2205 POKE1087,18:POKE1088,13
2222 PRINTTAB(42):PRINT"5"
2224 FOR I=1 TO 19:PRINT:NEXTI:PRINTTAB(2):PRINT"0"
2225 POKE1105,19:POKE1145,9:POKE1185,7:POKE1225,14:POKE1265,1:POKE1305,12
2230 POKE1385,9:POKE1425,14:POKE1465,16:POKE1505,21:POKE1545,20
2235 POKE1625,22:POKE1665,15:POKE1705,12:POKE1745,20:POKE1785,19
2240 RETURN
2245 :
2280 FOR N=1 TO 32: GOSUB2375
2285 GG=126: IF E=0 THEN GG=123
2310 POKE 1870+N-(XX*WZ),GG
2315 GOSUB2375
2330 GG=124:IF E=0 THEN GG=108
2365 POKE  1870+N-(XX*WZ),GG
2370 NEXT:RETURN
2375 GOSUB 2500:D= Y/.25  :WZ=INT(D)
2390 E=1:IF (D-WZ)>=0.5 THEN WZ=WZ+1:E=0
2395 RETURN
2500 Y =FZ(AE)
2502 AE=AE+1
2600 RETURN
8000 REM SCREEN PRINT FROM COMPUTE! APRIL 1984.
8010 USED BY PERMISSION FROM COMPUTE! PUBLICATIONS, INC
8000 PRINT CHR$(19);:SS=(PEEK(210))*256:OPEN3,3:OPEN4,4
8110 FOR R= 0 TO 24 : B$=""
8120 FOR C= 0 TO 39 : A$=""
8130 IFPEEK(SS+((R*40)+C))>127THEN:GET#3,A$:B$=B$+CHR$(18)+A$+CHR$(146):GOTO8160
8140 GET#3,A$:IF A$=CHR$(13)THEN:A$=" "
8150 B$=B$+A$
8160 NEXTC:PRINT#4,B$:NEXTR:CLOSE4:CLOSE3
8170 RETURN
READY.
```

been recorded. Table 3-4 presents the functional description of the "B" control routine.

The machine-language subroutine of Program 3-18is called by the "B" BASIC routine to control the high speed sampling of the *AID* converter. The main points about this subroutine is that it is located in RAM memory starting at location $C000, contains a machine-language time loop to control the

Table 3-4. Where the Main Program Routines are Located In the "B" BASIC Program 3-17.

Lines 1-45: Program set-up and description.

Lines 51210-··575: Record start and machine language subroutine control.

Lines 1121121-1512The routine to retrieve the waveform data from RAM.

Lines 25121-312112The recorded data display routine.

Lines 21210121-2612Waveform display routine.

Lines 81211210-81712Screen print routine.

waveform sampling rate, and stores the sampled waveform data in RAM memory which is later PEEKed by the "B" BASIC control routine to retrieve the sampled waveform data. Table 3-5 presents the functional description for the machine-language subroutine.

The waveform display routine that is located between lines 2000 to 2600 uses the low-resolution

Table 3-5. This Table Is for Program 3-18 and Describes the Subroutine Functions that are Performed by the Machine-Language Instructions In the Specified Hexedeclmal Memory Locations. In This Table, the Hexedecimal Memory Locations Are Used Like the Line Numbers In Tables 3-3 and 3-4.

Address Locations:

$CI2II21LD2I$CI2I12I4:Loop set-up for 65 waveform samples.

$CI2I1218 to $CI2I14: *AID* converter start conversion logic generation"

$C12I19 to $CI2I21: Time delay looP"

$C12I28 to $CI2I2A: Set-up to read AID converter"

$C12I31 to $CI2I34: Read *AID* and store sample data.

$C12I38 to $CI2I3B: sample count routine. After 65 samples, $CI2I3Doperation returns to the BASIC program.

```
PROGRAM 3.18
  THE MACHINE LANGUAGE SUBROUTINE
  FOR PROGRAM 3.17
,C000 A2 00     LDX #$00
,C002 A9 41     LDA #$41
,C004 8D FF C0  STA $C0FF
,C007 EA        NOP
,C008 A9 9B     LDA #$9B
,C00A 8D 00 DD  STA $DD00
,C00D A9 97     LDA #$97
,C00F 8D 00 DD  STA $DD00
,C012 A9 9B     LDA #$9B
,C014 8D 00 DD  STA $DD00
,C017 EA        NOP
,C018 EA        NOP
,C019 A9 09     LDA #$09
,C01B 8D 45 C0  STA $C045
,C01E CE 45 C0  DEC $C045
,C021 D0 FB     BNE $C01E
,C023 EA        NOP
,C024 EA        NOP
,C025 EA        NOP
,C026 EA        NOP
,C027 EA        NOP
,C028 A9 97     LDA #$97
,C02A 8D 00 DD  STA $DD00
,C02D EA        NOP
,C02E EA        NOP
,C02F EA        NOP
,C030 EA        NOP
,C031 AD 01 DD  LDA $DD01
,C034 9D 78 C0  STA $C078,X
,C037 E8        INX
,C038 CE FF C0  DEC $C0FF
,C03B D0 CB     BNE $C008
,C03D 60        RTS
,C03E 00        BRK
```

Program 3-18. This is the machine-language subroutine for Program 3-17.

graphics in the C-64 to display the recorded waveform. The display resolution is limited, but the waveform presentation does provide a good relative indication of what the waveform really looks like without the addition of a high-resolution graphics software program. The plotting routine uses a calculation format that is similar to the high-resolution program on page 126 in the C-64 Programmer's Guide. A functional description of this type of waveform display routine is presented in Chapter 8.

The screenprint routine that is located between lines 8000 to 8170 is used with the permission of *Compute!* magazine. After the waveform has been recorded and displayed, you can exit the waveform display by pressing any key. The program then asks you if you want a hardcopy printout. If you say YES, the program goes back and reruns the data display and the waveform display and the screenprint routine copies each video display on the computer's printer. This screenprint is a low-resolution routine that will work on all Commodore printers. Figure 3-13 shows a recorded waveform printout using this routine.

The two high-resolution AWR programs are presented in Programs 3-19 and 3-21. These two AWR programs were written using the SIMON'S BASIC cartridge from Commodore, but there are several other high-resolution software programs that could be easily used if you already have one. Program 3-19 is a high-speed AWR program that uses the machine-language subroutine of Program 3-20. The recording capabilities of this program is

```
1 REM - PROGRAM 3.19 - A HI RESOLUTION WRP FOR THE C-64 USING SIMON'S BASIC
2 REM - THIS PROGRAM DISPLAYS 250 WAVEFORM SAMPLE POINTS
5 PRINTCHR$(147):DIM AA(256)
10 PRINT"       A HIGH RESOLUTION WAVEFORM                    RECORDING  PROGRAM"
15 PRINT" "
20 PRINT"INPUT DATA FOR TIME INTERVAL BETWEEN    RECORDED SAMPLES -";
30 INPUT J1
70 PRINT" ":PRINT" PUSH <R> TO START RECORDING"
75 GET A$:IF A$="R" THEN GOTO85
80 GOTO75
```

Program 3-19. This program is an example of the "B" BASIC hi-resolution waveform-recording program. The waveform display is much neater than the low-resolution display, but this program is written using SIMON's BASIC, which requires that you purchase Simon's Basic program cartridge for the C-64 to use this program. This program uses the machine-language subroutine of Program 3-20.

```
85 PRINT" ":PRINT" RECORDING":PRINT" "
90 :
100 FOR AM=0 TO 254
110 FOR H = 0 TO J1: NEXT H
120 POKE56576,155:POKE56576,151:AA(AM)=PEEK(56577):NEXTAM
130 HIRES1,6:GOTO 2000
140 FOR II=0TO250
145 A=II+40 :B=(AA(II) *.58):C=INT(164-B)
150 PLOT A,C,1
155 NEXTII
160 GOTO160
500 DRAW  0,10,10,190 ,1
1000 GOTO110
2000 :
2020 LINE 40,15, 40,165,1
2025 LINE 40,165,291,165,1
2030 FOR I=15 TO 165 STEP 6
2035 LINE 35,I,40,I,1
2040 NEXTI
2050 FOR I=15 TO 165 STEP30
2055 LINE 30,I,35,I,1:NEXTI
2060 FOR I= 40 TO 296 STEP 10
2065 LINE I,165,I,170,1:NEXT I
2070 FOR I=40 TO 290 STEP50
2080 LINE I,165,I,175,1:NEXTI
2090 CHAR 20,11,53,1,1
2100 CHAR 20,41,52,1,1
2110 CHAR 20,71,51,1,1
2115 CHAR 20,101,50,1,1
2120 CHAR 20,131,49,1,1
2125 CHAR 20,161,48,1,1
2130 CHAR 37,178,48,1,1
2135 CHAR 81,178,53,1,1
2140 CHAR 90,178,48,1,1
2145 CHAR 127,178,49,1,1
2150 CHAR 136,178,48,1,1
2155 CHAR 145,178,48,1,1
2160 CHAR 177,178,49,1,1
2165 CHAR 186,178,53,1,1
2170 CHAR 195,178,48,1,1
2175 CHAR 227,178,50,1,1
2180 CHAR 236,178,48,1,1
2185 CHAR 245,178,48,1,1
2190 CHAR 277,178,50,1,1
2195 CHAR 286,178,53,1,1
2200 CHAR 295,178,48,1,1
2210 TEXT 60,190,"■ RECORDED DATA POINTS",1,1,8
2215 CHAR 5,40,9,1,1
2220 CHAR 5,50,14,1,1
2225 CHAR 5,60,16,1,1
2230 CHAR 5,70,21,1,1
2235 CHAR 5,80,20,1,1
2240 CHAR 5,100,4,1,1
2245 CHAR 5,110,1,1,1
2250 CHAR 5,120,20,1,1
2255 CHAR 5,130,1,1,1
2400 GOTO 140
READY.
```

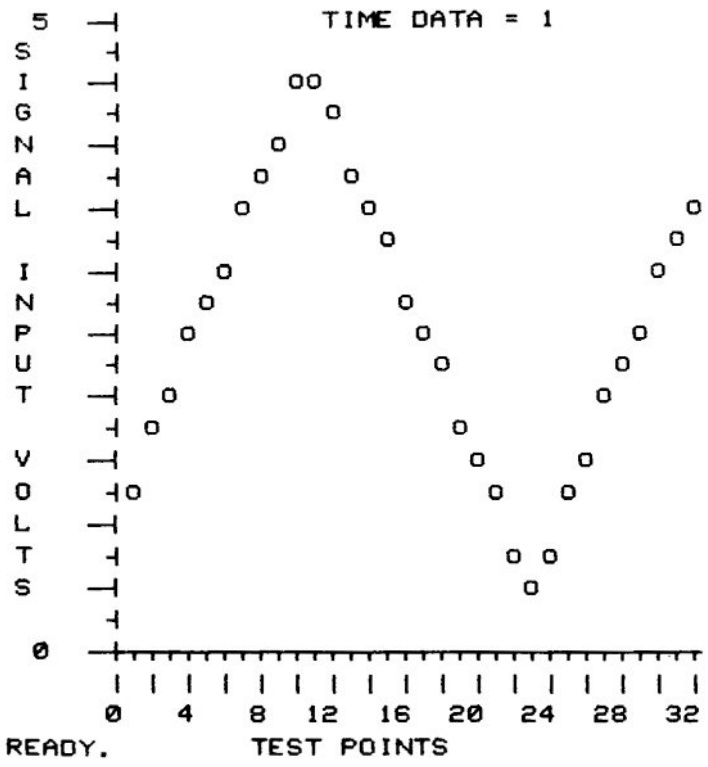


Fig. 3-13. Video and hardcopy printout display of the low-resolution waveform-recording programs.

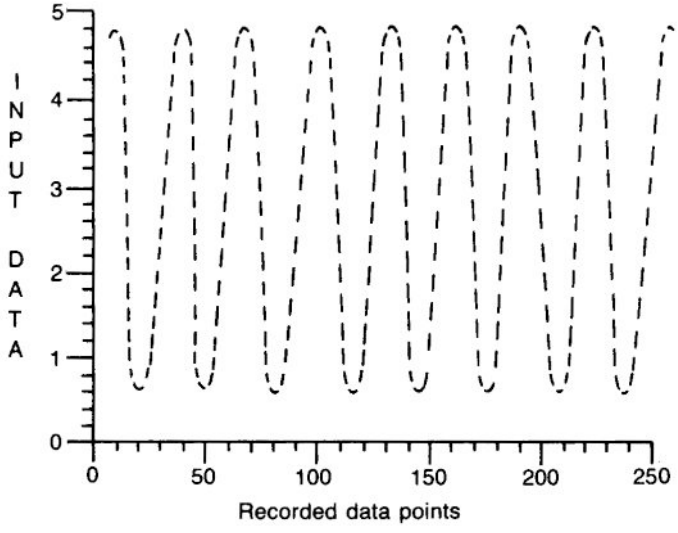


Fig. 3-14. Video and hardcopy printout display of the high-resolution waveform-recording programs.

```
SUBROUTINE PROGRAM 3.20

FOR PROGRAM 3.19


,CC00 A2 00     LDX #$00
,CC02 A9 FF     LDA #$FF
,CC04 8D FF CC  STA $CCFF
,CC07 EA        NOP
,CC08 A9 9B     LDA #$9B
,CC0A 8D 00 DD  STA $DD00
,CC0D A9 97     LDA #$97
,CC0F 8D 00 DD  STA $DD00
,CC12 A9 3B     LDA #$9B
,CC14 8D 00 DD  STA $DD00
,CC17 EA        NOP
,CC18 EA        NOP
,CC19 A9 0A     LDA #$0A
,CC1B 8D 45 CC  STA $CC45
,CC1E CE 45 CC  DEC $CC45
,CC21 D0 FB     BNE $CC1E
,CC23 EA        NOP
,CC24 EA        NOP
,CC25 EA        NOP
,CC26 EA        NOP
,CC27 EA        NOP
,CC28 A9 97     LDA #$97
,CC2A 8D 00 DD  STA $DD00
,CC2D EA        NOP
,CC2E EA        NOP
,CC2F EA        NOP
,CC30 EA        NOP
,CC31 AD 01 DD  LDA $DD01
,CC34 9D 00 CD  STA $CD00,X
,CC37 E8        INX
,CC38 CE FF CC  DEC $CCFF
,CC3B D0 CB     BNE $CC08
,CC3D 60        RTS
,CC3E 00        BRK
```

Program 3-20. This is the machine-language subroutine program for the "B" BASIC Program 3-19. Note that this subroutine is located at $CCOO instead of $COOO so it will not interfere with the screen display of Simon's Basic.

```
1 REM - PROGRAM 3.21 - A HI RESOLUTION WRP FOR THE C-64 USING SIMON'S BASIC
2 REM - THIS PROGRAM DISPLAYS 250 WAVEFORM SAMPLE POINTS
3 REM - THIS IS AN ALL BASIC PROGRAM
5 PRINTCHR$(147):DIM AA(256)
10 PRINT"        A HIGH RESOLUTION WAVEFORM                    RECORDING  PROGRAM"
15 PRINT" "
20 PRINT"INPUT DATA FOR TIME INTERVAL BETWEEN    RECORDED SAMPLES -";
30 INPUT J1
70 PRINT" ":PRINT" PUSH <R> TO START RECORDING"
75 GET A$:IF A$="R" THEN GOTO85
80 GOTO75
85 PRINT" ":PRINT" RECORDING":PRINT" "
90 POKE 56334,00
100 FOR AM=0 TO 254
110 FOR H = 0 TO J1: NEXT H
120 POKE56576,155:POKE56576,151:AA(AM)=PEEK(56577):NEXTAM
125 POKE 56334,01
130 HIRES1,6:GOTO 2000
140 FOR II=0TO250
145 A=II+40 :B=(AA(II) *.58):C=INT(164-B)
150 PLOT A,C,1
155 NEXTII
160 GET A$: IF A$="C" THEN GOTO170
165 GOTO160
170 COPY
180 GOTO180
2000 :
```

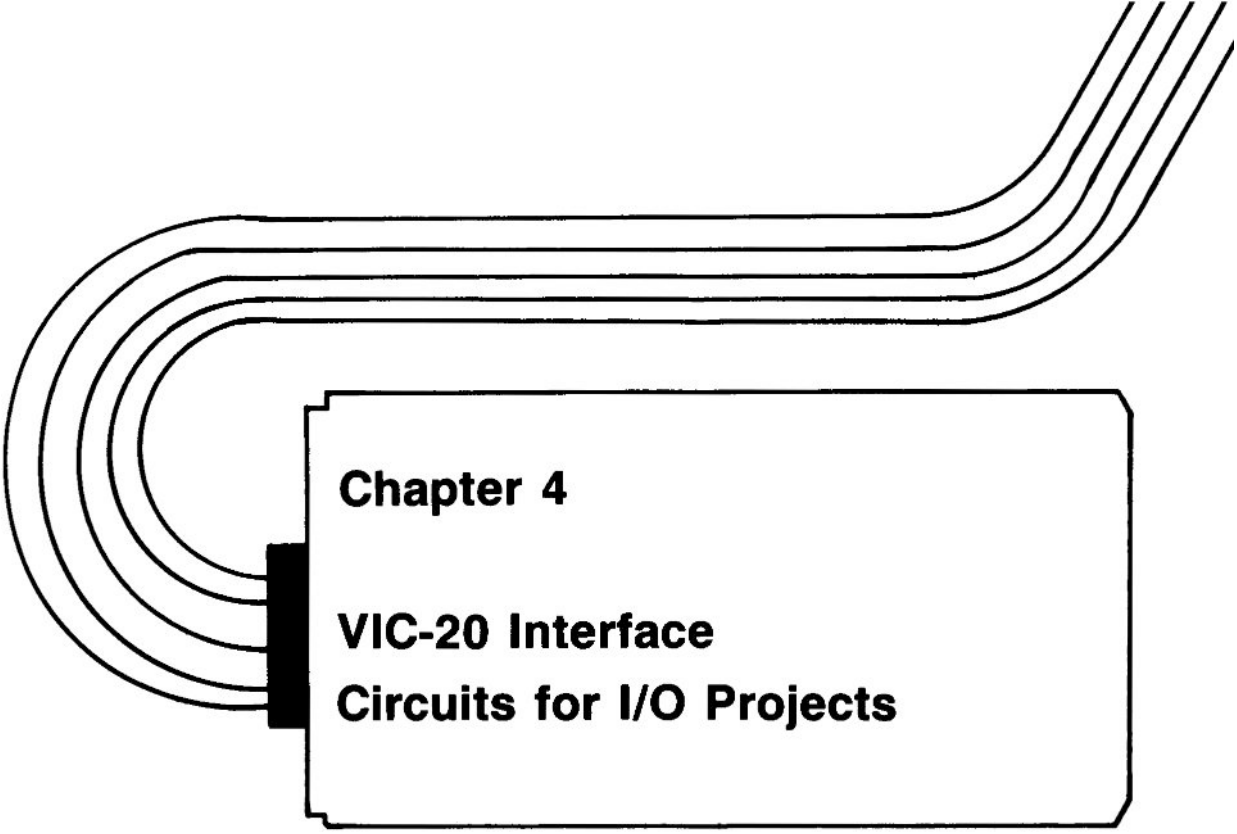
Program 3-21. This is the "A" BASIC high-resolution waveform recording program. This program will do a really good job of displaying slowly progressing analog waveforms such as a temperature-versus-time data graph.

```
2020 LINE 40,15, 40,165,1
2025 LINE 40,165,291,165,1
2030 FOR I=15 TO 165 STEP 6
2035 LINE 35,I,40,I,1
2040 NEXTI
2050 FOR I=15 TO 165 STEP30
2055 LINE 30,I,35,I,1:NEXTI
2060 FOR I= 40 TO 296 STEP 10
2065 LINE I,165,I,170,1:NEXT I
2070 FOR I=40 TO 290 STEP50
2080 LINE I,165,I,175,1:NEXTI
2090 CHAR 20,11,53,1,1
2100 CHAR 20,41,52,1,1
2110 CHAR 20,71,51,1,1
2115 CHAR 20,101,50,1,1
2120 CHAR 20,131,49,1,1
2125 CHAR 20,161,48,1,1
2130 CHAR 37,178,48,1,1
2135 CHAR 81,178,53,1,1
2140 CHAR 90,178,48,1,1
2145 CHAR 127,178,49,1,1
2150 CHAR 136,178,48,1,1
2155 CHAR 145,178,48,1,1
2160 CHAR 177,178,49,1,1
2165 CHAR 186,178,53,1,1
2170 CHAR 195,178,48,1,1
2175 CHAR 227,178,50,1,1
2180 CHAR 236,178,48,1,1
2185 CHAR 245,178,48,1,1
2190 CHAR 277,178,50,1,1
2195 CHAR 286,178,53,1,1
2200 CHAR 295,178,48,1,1
2210 TEXT 60,190,"  RECORDED DATA POINTS",1,1,8
2215 CHAR 5,40,9,1,1
2220 CHAR 5,50,14,1,1
2225 CHAR 5,60,16,1,1
2230 CHAR 5,70,21,1,1
2235 CHAR 5,80,20,1,1
2240 CHAR 5,100,4,1,1
2245 CHAR 5,110,1,1,1
2250 CHAR 5,120,20,1,1
2255 CHAR 5,130,1,1,1
2400 GOTO 140
READY.
```

the same as Program 3-17, but this program records and displays 250 waveform samples. Program 3-21 is an all BASIC AWR program which has similar recording characteristics to Program 3-16, but will also record and display 250 sample points. The screen printout routine for these two programs will only work with the VIC-1525 printer. To printout a waveform display, you must press <C> when the program has finished displaying the waveform. It may be necessary for you to reset the printer by turning the ac switch off and back on to get the high-resolution screen printout to work. A recorded waveform printout is shown in Fig. 3-14.

The waveform recording programs to Project

3-4 are not meant to be a tool for highly technical research, but you can learn a lot about waveform recording and digitizing with these programs when they are used to monitor school experiments that are performed in electronics, physics, and other science courses.



Chapter 4

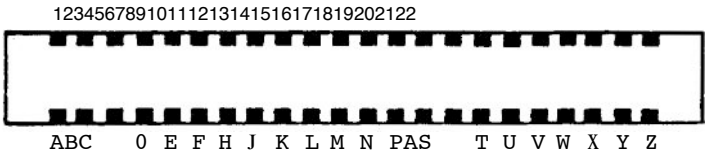
VIC-20 Interface Circuits for I/O Projects

IT IS MY OPINION THAT THE VIC-20 COMPUTER is the best computer to use when learning how to interface I/O chips and circuits to a computer. At the printing of this book, the VIC-20 is no longer in production, but there are plenty of these computers available through the used computer channels at very reasonable prices. The unexpanded VIC-20 computer has lots of unused memory map space that can be used for I/O chip operation. The VIC-20's memory map is a list of all of the possible memory address locations that are available in that computer and the function that is allocated to each of the memory locations. After you have studied this chapter, you will have the technical knowledge that is required to use these unused memory locations for I/O interface projects.

The VIC-20 as purchased from the computer store has only one eight-bit I/O port and a game oriented *AID* converter. This one byte I/O port and game oriented *AID* converter will only support small I/O projects of limited scope. It soon becomes apparent that the VIC-20 needs more I/O lines plus a good *AID* converter if you are going to control a project which is on a level other than beginning. The goal of this chapter is to show you how to add an additional 6522 VIA I/O chip, an *AID* converter circuit, and **lK** of extra machine-language memory to your computer. The addition of these three hardware circuits to your VIC-20 will give you the ability to investigate many new applications in the areas of science and engineering.

Each of the three circuits can be built on a Radio Shack 44-pin edge-card circuit board as a stand-alone plug-in module for the expansion port. A multislot expansion port plug-in card can be used if more than one circuit is needed. The I/O circuits are designed so the **lK** of extra memory is assigned to memory locations $A000 to $A3FF, the 6522 is assigned to locations $9800 to $980F, and the *AID* converter is assigned to locations $9C00 to $9C0F. These memory location assignments will still let you use other plug-in modules like VICMON or the BASIC memory expansion modules.

As mentioned above, the construction method that was selected for these circuits uses a 44-pin edge-card experimenter's circuit board which will



(Looking at the connector tram the back of the computer,

Pin	Function	Pin	Function
1	GND	A	GND
2	CD0	B	CA0
3	CD1	C	CA1
4	CD2	D	CA2
5	CD3	E	CA3
6	CD4	F	CA4
7	CD5	H	CA5
8	CD6	J	CA6
9	CD7	K	CA7
10	$\overline{\text{BLK1}}$ ($2000-$3FFF)	L	CA8
11	$\overline{\text{BLK2}}$ ($4000-$5FFF)	M	CA9
12	$\overline{\text{BLK3}}$ ($6000-$7FFF)	N	CA10
13	$\overline{\text{BLK5}}$ ($A000-$BFFF)	P	CA11
14	$\overline{\text{RAM1}}$ ($0400-$07FF)	R	CA12
15	$\overline{\text{RAM2}}$ ($0800-$0BFF)	S	CA13
16	$\overline{\text{RAM3}}$ ($0C00-$0FFF)	T	$\overline{\text{I/O2}}$ ($9800-$9BFF)
17	VR/W	U	$\overline{\text{I/O3}}$ ($9C00-$9FFF)
18	CR/W	V	$\overline{\text{SO2}}$
19	$\overline{\text{IRQ}}$	W	$\overline{\text{NMI}}$
20	-NC-	X	$\overline{\text{RESET}}$
21	5 Volts	Y	-NC-
22	GND	Z	GND

Fig. 4-1. The pinout data for the VIC-20 expansion port. The hexadecimal numbers to the right of the chip select pins (BLK1, RAM1, or 1103 etc.) indicates the memory address location areas that they control.

plug into the expansion port of the VIC-20. Using multicolored flat cables to run the data bus and address bus lines and a pointed tip soldering iron will help keep mistakes to a minimum. Be sure that you understand the nonstandard mirror image pin-out that is used on the VIC-20 expansion port as shown in Fig. 4-1. Only build one circuit board AT A TIME, and then make that circuit work before you start on the next circuit!

THE 1K MACHINE-LANGUAGE MEMORY CIRCUIT

The basic VIC-20 has only 5K of RAM memory of which 3,583 bytes are available for the BASIC program. Allocating some of this memory for machine-language routines can be tough at times when the computer's BASIC program has only 3.5 K of RAM memory available for programming. The VIC-20 does have unused memory locations in the area between $0400 and $0FFF, but this memory area can only be used for the BASIC PROGRAM because of the way the computer turns on at start-up. The easiest way to add a small amount of memory for machine-language operation is to place lK of RAM memory starting at location $A000. The area between $A000 and $BFFF is used for auto-power-up program ROMs which you

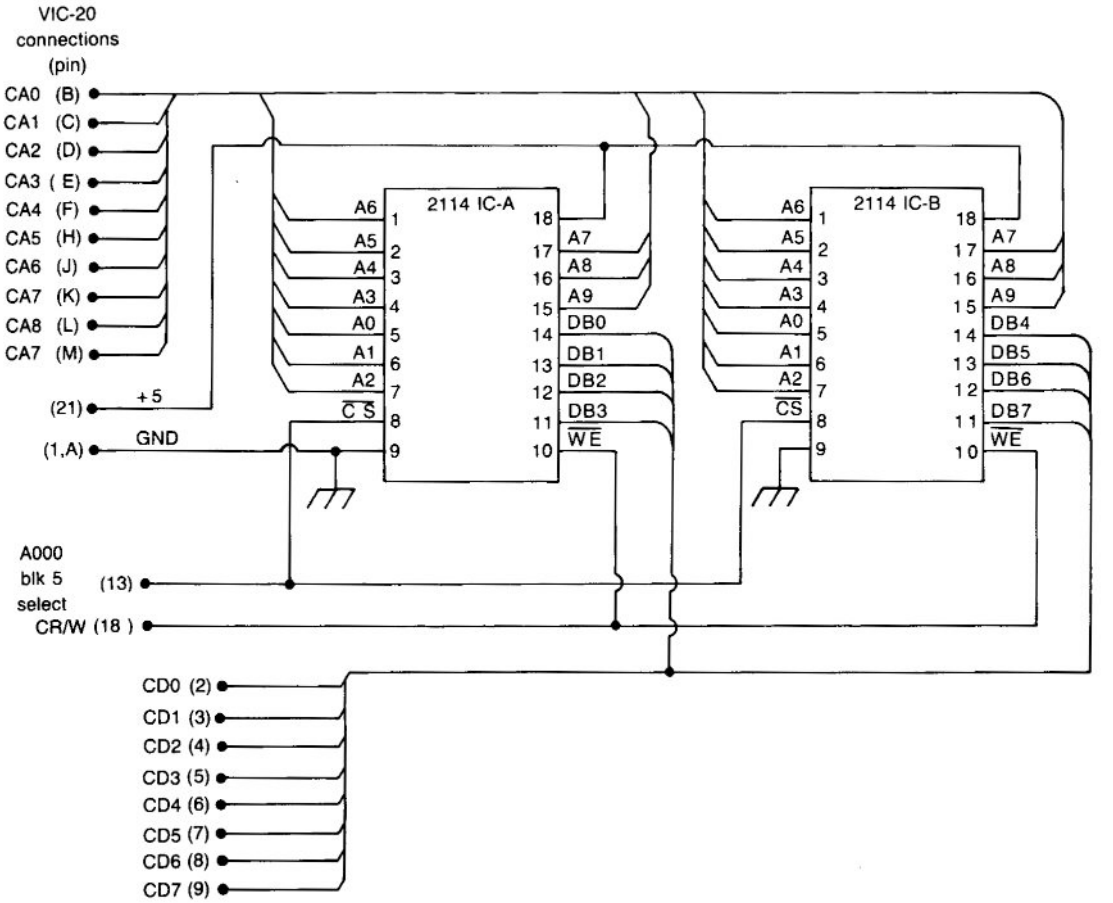


Fig. 4-2. The 1K of additional machine-language memory. Note that this circuit is connected to the block 5 chip-select line so the memory is located at $A000 to $A3FF in the VIC-20 memory map.

will not be using with an 110 project. The addition of 1K of RAM memory in this area will let the programmer have all of the standard memory for BASIC and a complete 1K of RAM for machine-language routines. The 1K of RAM memory will handle any programming problem that the student or hobbyist will encounter with a VIC-20.

The easiest RAM IC that can be used with the VIC-20 is the 2114. This chip can be purchased at most electronics hobby stores. The 2114 is a 1K-by-4 style memory chip and two IC chips are required to form a 1K block of memory. It is very simple, as shown in Fig. 4-2, to connect two 2114 memory chips so one chip contains the high-order data and the other contains the low-order data for a byte of memory. This means that the address bus lines (CA0 to CA9) are connected to both chips in parallel while the data bus lines CD0 to CD3 are connected to chip IC-A, and CD4 to CD7 are connected to chip IC-B. The device select and the read-write signals from the VIC-20 are also needed to complete the memory addition.

If there are no other uses planned for the area of the memory map between $A000 and $BFFF, the address decoding that is required for the 1K of memory and the rest of the area between $A3FF to $BFFF will be unusable. But, this chip selecting method is a quick, easy, and cost-effective way of adding 1K of RAM memory for machine-language routines if the unused memory area can be wasted. This added memory does not bother the operation of the VICMON machine-language monitor program or the start-up routine of the computer.

When you have the memory circuit completed, check it over for solder shorts between the IC socket pads. Make sure that Vee is going to pin 18 and pin 9 goes to ground. If the plus 5 volts and ground are connected correctly, you can not hurt the VIC-20 when you plug in the circuit board. TURN OFF the computer before you plug-in the circuit board. When you turn on the VIC-20, the "ON" LED should light and the video display should show up as normal. If this does not happen, check the board for wiring errors. If the computer does not turn-on OK, try POKE 49152,255 and then when you try a PEEK (49152) the computer should return a 156. These two POKE-PEEK commands will tell you if the machine-language memory addition is working correctly.

ADDING AN EXTRA 6522 VIA 110 CHIP

The VIC-20 has only one eight bit *1/0* port available at the USER PORT. Any *1/0* project will quickly use up these eight 110 lines. The addition of a 6522 VIA chip will give you 16 extra *1/0* lines plus the other extras that are included in the 6522. The *VIC-20 Programmer's Guide* has a good operational description of the 6522 which you should read before you build this part of the project.

Adding an extra 6522 to the VIC-20 is even easier than adding the 1K of memory as one can see by examining Fig. 4-3. All of the required operational signals for the 6522 are available at the VIC-20 expansion connector and all that is needed is the connecting wires between the expansion port and the 6522. This project only uses one 6522, so no address decoding is required. The 1102 chip-select line is connected to one of the 6522 chip-select pins, which locates the VIA at address locations $9800 to $980F in the VIC-20's memory map. When the VIC-20 is turned on, it will set up all of the internal registers in the external 6522 just like it does with the 6522 ICs inside the computer . You can program and use the additional 6522 exactly the same as you would use the 6522 that is associated with the USER PORT.

ADDING AN ANALOG- TO-DIGITAL CONVERTER CIRCUIT

In Chapter 3, an *A/D* converter was described that operated from the USER PORT. The AID converter in this project operates with the expansion port. The *AID* converter chip that was selected for this project is the ADC0817. This chip was selected because it can be easily purchased and has 16 programmably-selectable input channels. The ADC0817 is very easy to interface to the VIC-20 because of the internal input latching, multiplexing, and TTL-compatible tri-state output circuits that are used on the ADC chip.

A functional *AID* circuit is shown in Fig. 4-4 for interfacing the ADC0817 IC to a VIC-20. The operation is set up so a 0- to 5-volts analog input

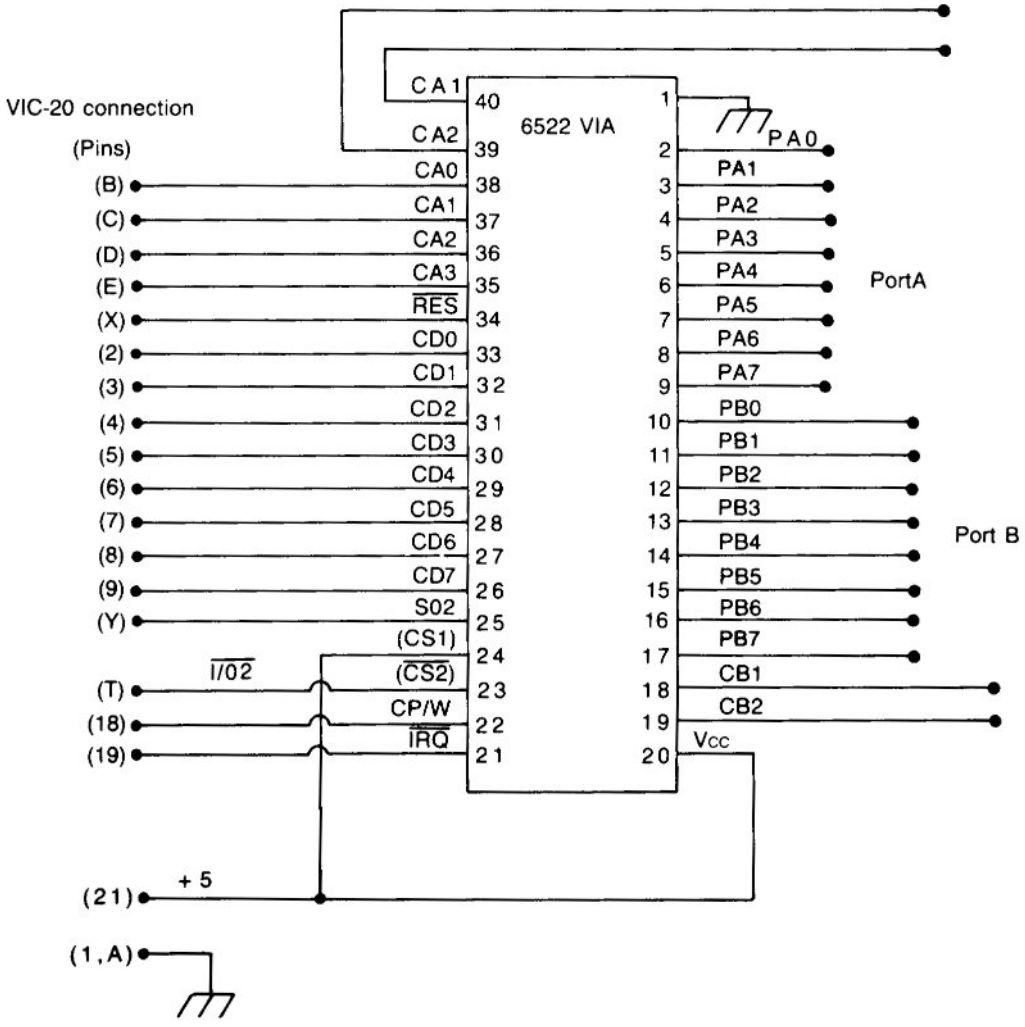


Fig. 4-3. The connection details of the 6522 VIA chip. The chip-select pin number 23 is connected to the 1102 chip-select line to locate the 6522 at addresses $9800 to $980F.

signal can be converted into an eight-bit digital representation for use with the computer. As with the other 110 circuits in this chapter, this circuit is designed so that no address decoding is required. The 1103 chip-select line is used with the four LSB address lines (CA0 to CA3) to select the AID IC and the analog input channel. The channels are addressed just like any other memory location. Using the 1103 chip select line and address lines CA0 to CA3 places the *AID* converter in the VIC-20 memory map at addresses $9C00 to $9C0F. When the AID converter is selected by the 1103 select line, the conversion process is started by writing to the input channel memory location that is to be con-

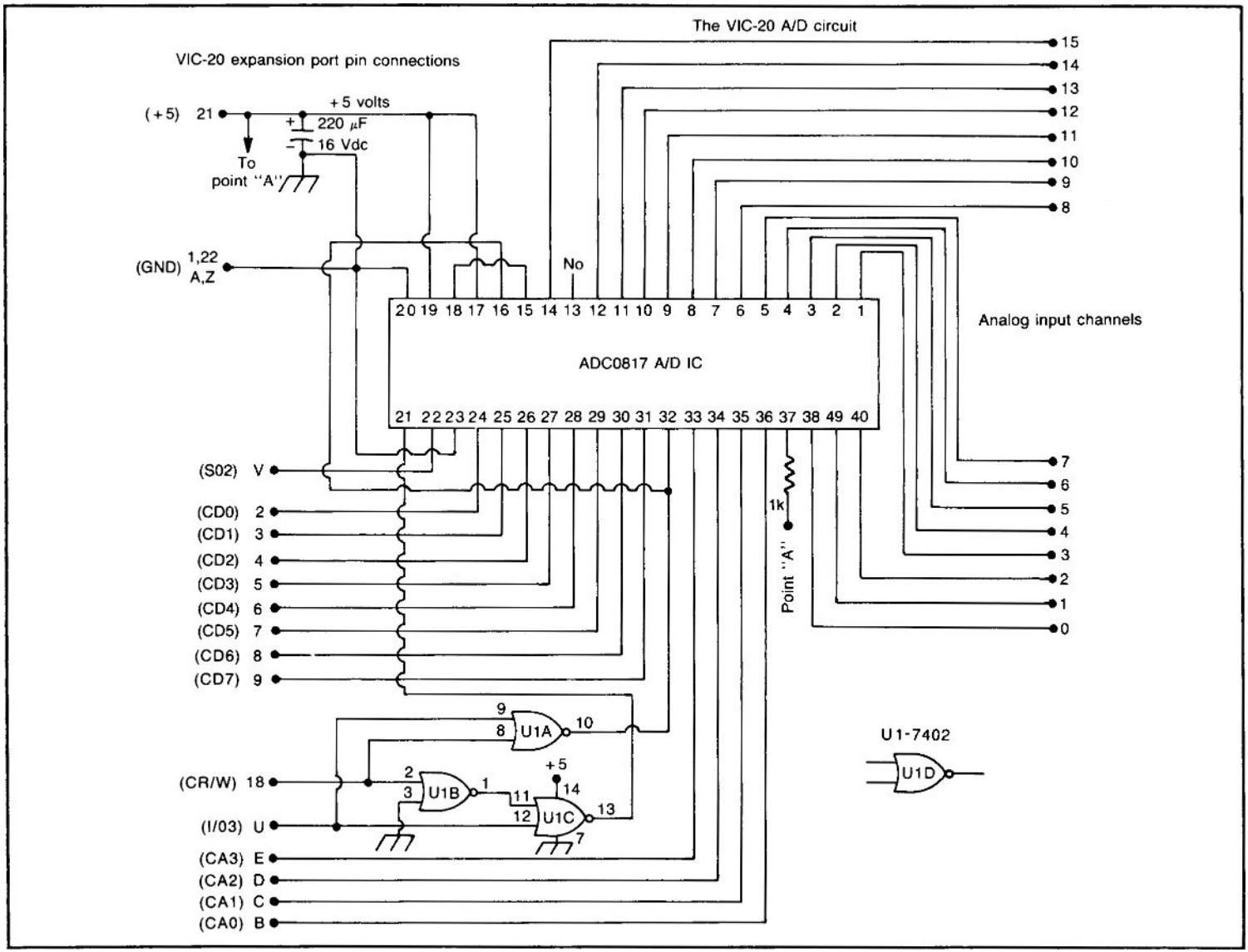


Fig. 4-4. The analog-to-digital converter circuit for the VIC-20 which has sixteen input channels that are located at addresses $9C00 to 9C0F.

```
1 REM  PROGRAM 4.1
2 REM A/D 16 CHANNEL TEST
5 QQ=39936
10 PRINTCHR$(147):GOTO 40
15 FOR DD=0 TO 15
20 POKE QQ+DD,00
25 PRINT"CHANNEL "DD "=";
30 PRINTPEEK(QQ+DD)
35 NEXT
36 PRINT" "
40 PRINT"PRESS (C) TO CHECK CHANNELS"
50 GET A$:IF A$="C" THEN GOTO 15
60 GOTO50
```

Program 4-1. This program is written to test all sixteen AID channels of the AID converter of Fig. 4-4.

verted. This operation places the proper address data on the address bus lines, drops the CR/W line low, and drops the 1103 select line low. This is accomplished on the VIC-20 by POKING the selected memory location *(AID* channel) with the data $00. The analog-to-digital conversion process takes about 60 microseconds to complete. After the conversion is completed, the data is loaded into the computer by performing a PEEK of the *AID* channel (memory) location. This PEEK (or read) operation drops the *1/03* select-line low but leaves the *CRIW* line high, which places the converted data on the tri-state data pins and onto the computer's data bus lines. The computer then loads this data into a memory location as a representation of the real world analog signal that was present at the selected input channel at the time of conversion.

The actual operation of the *AID* converter is straight forward and any problems can usually be traced to wiring errors. Program 4-1 is presented to help you test out the *AID* circuit.

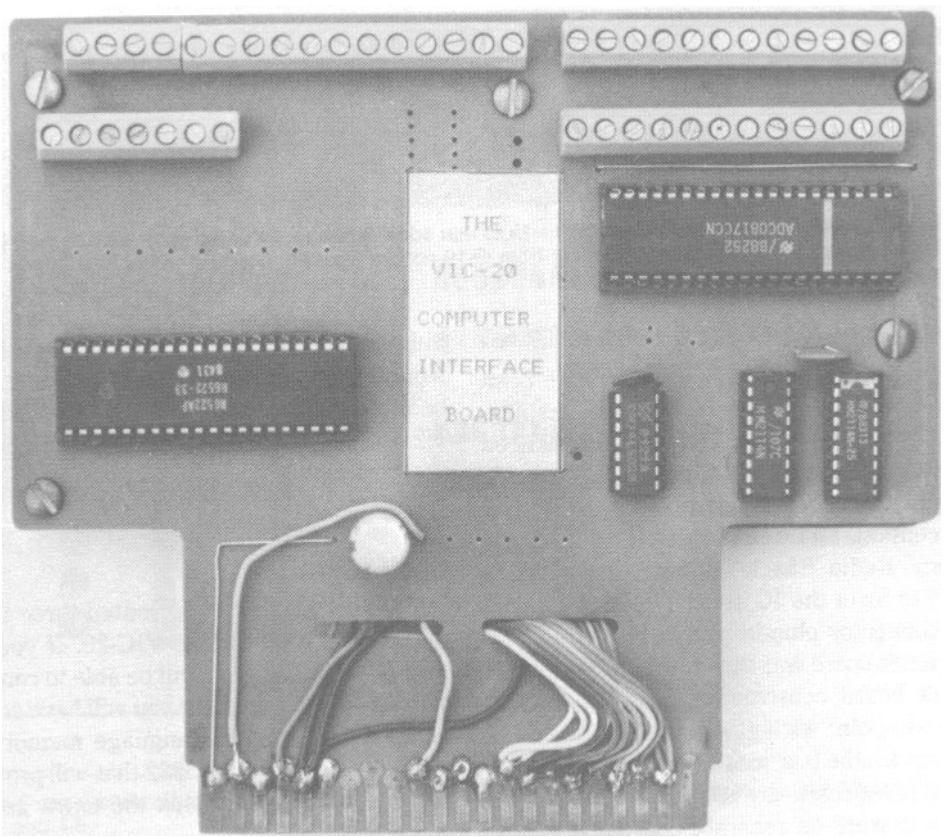


Fig. 4-5. The top-side of the etched circuit board that contains all three *1/0* cirCUits.

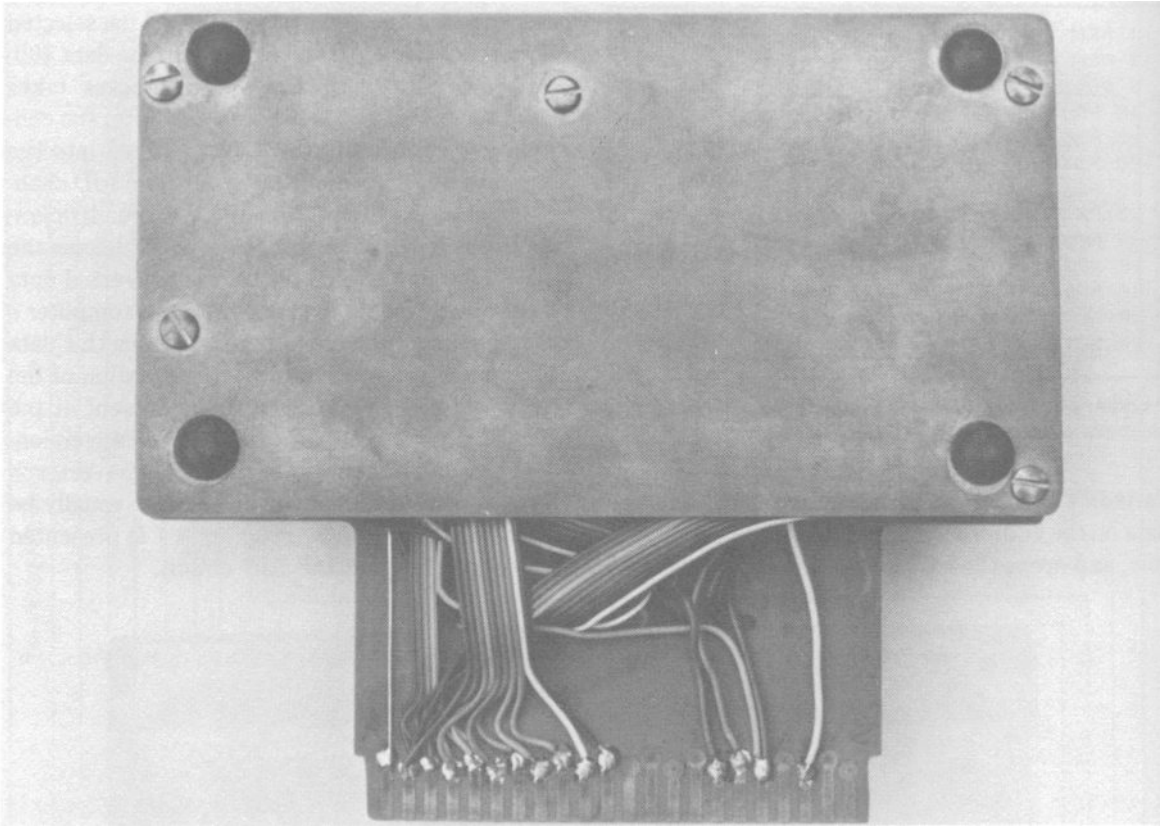


Fig. 4-6. The bottom-side of the etched circuit board. Note that edge-card connector pin pads were etched on both sides of the board. A protective bottom plate was made from G-10 copper clad PC board to protect the circuit wiring.

AN 1/0 SYSTEM ON A SINGLE PLUG-IN CARD

The three circuits of this chapter have been described separately in order to keep the complexity of the circuits down. It is easy for an advanced hobbyist to build all three of these circuits on one circuit board. Figures 4-5 and 4-6 shows a plug-in card that contains all three circuits. This card was made using Radio Shack rub-on transfer pads (276-1577) to form the IC socket solder pads and the edge connector plug-in pins as shown in Fig. 4-7. The circuit board was then finished by etching. The circuit board construction is completed by using point-to-point wiring. Again, the use of flat colored cable for the bus lines will help prevent wiring errors. The circuits are tested out the same as if the circuits were on separate boards. It is best to build one circuit at a time and make that circuit work before building the next circuit. It would also be a good idea to place a small fuse (no larger than 112amp) in the Vee supply line that comes from the computer.

CONCLUSION

This chapter has presented three 110 circuits that can be built for the VIC-20. If you build the three I/O circuits, you will be able to control a very complex project because you will have at your command 1K of machine-language memory, 16 *AID* channels, and an extra 6522 that will provide an additional 16 110 lines plus the extra goodies that come with the 6522.

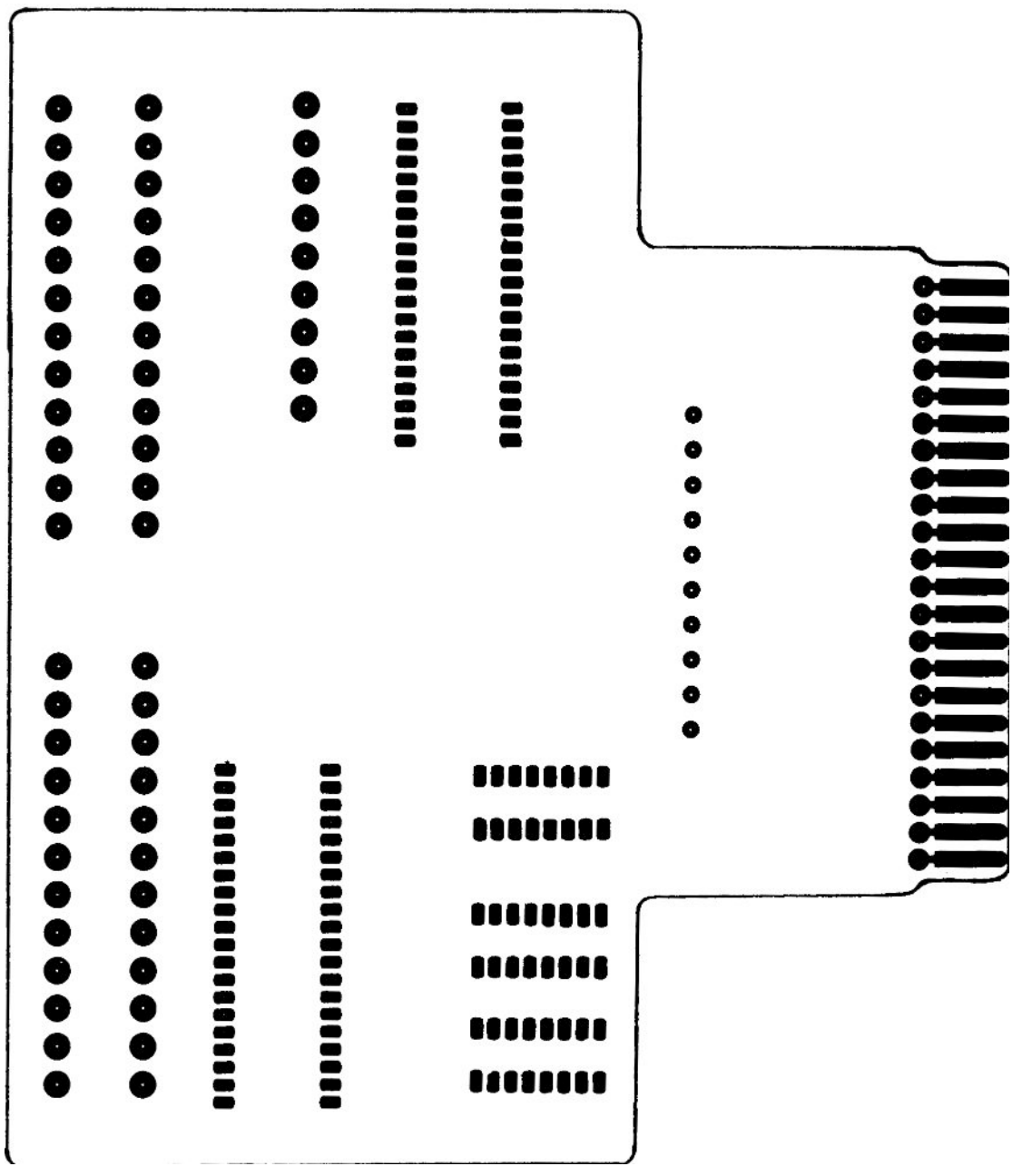


Fig. 4-7. A full-size etched circuit board layout that was used to build all three *1/0* circuits on one plug-in card as shown in Figs. 4-5 and 4-6.

Chapter 5

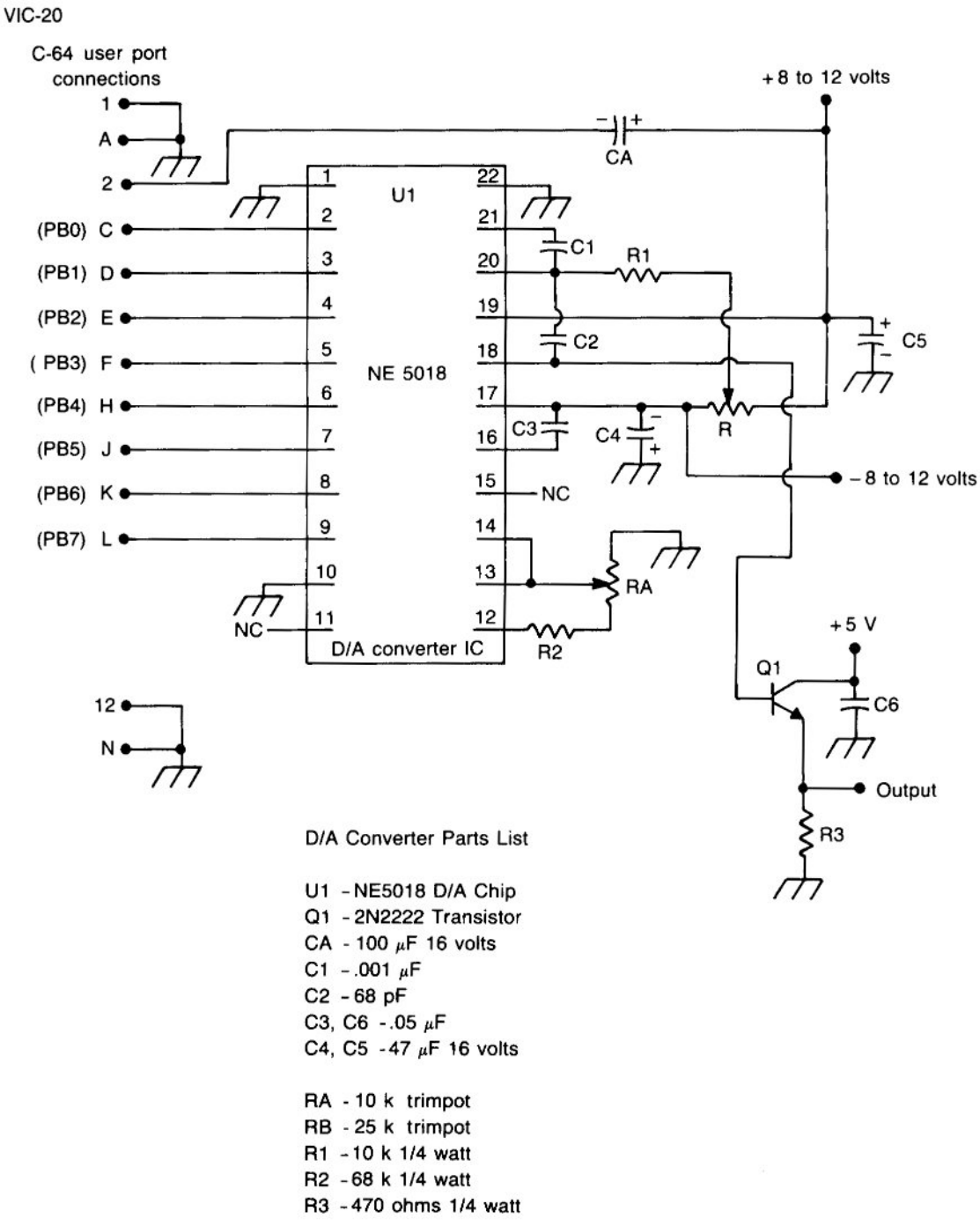
A Digital-to-Analog Converter Circuit for the VIC-20 and C-64

THIS eHAPTER DEseRIBES A DIGITAL-TO-ANAlog converter circuit that is designed to operate from the USER PORT on either the VIC-20 or the C-64. Using this *DIA* converter along with an *AID* converter will give you an eight-bit analog control system that can perform some really sophisticated control functions.

Because Chapters 4 and 6 present *AID* converter circuits that operate from the Expansion Ports on the computers, the *DIA* converter circuit of Fig. 5-1 was designed to operate from the USER PORT, giving you the ability to use both *AID* and a *DIC* converter at the same time. The circuit is built around a NE5018 single-chip microprocessor compatible *DIA* converter. Pin 10 of the NE5018 is grounded, which sets up a straight-through conversion function. This means that any parallel digital data applied to pins 2 through 9 generates a corresponding analog signal at pin 18. Pin 18 is connected to Q2, an emitter-follower circuit. Trimpot RA is used to control the amplitude of the analog output signal and trimpot RB is used to control the dc level of the output signal. To adjust this circuit, connect an oscilloscope to point "A", which is the circuit's analog signal output. Load in the machine-language program (Program 5-1) and run the program. Adjust trimpot RA for a zero to five volts output signal. Then adjust trimpot RB so the top of the signal is not clipped and then readjust trimpot RB so the bottom of the signal is not clipped. When both trimpots are adjusted correctly, you should see a good 0- to 5-volt sawtooth waveform.

The construction of this board uses an experimenter's board like most of the other projects in this book. The circuit construction is shown in Figs. 5-2, and 5-3. Capacitor CA is used to ac couple the power supply in the computer to the external de supply for the converter circuit to lower potential noise problems.

This circuit is simple and you should not have any trouble building and getting it working. Remember that you must make the USER PORT an output port to write data to the *DIA* converter. The *DIA* chip's latch function is not used in this circuit, so the data that is placed into the USER PORT



D/A Converter Parts List

U1 - NE5018 D/A Chip
Q1 - 2N2222 Transistor
CA - 100 μF 16 volts
C1 - .001 μF
C2 - 68 pF
C3, C6 - .05 μF
C4, C5 - 47 μF 16 volts

RA - 10 k trimpot
RB - 25 k trimpot
R1 - 10 k 1/4 watt
R2 - 68 k 1/4 watt
R3 - 470 ohms 1/4 watt

Fig. 5-1. The circuit diagram for the digital-to-analog converter circuit.

```
,C100 A9 FF     LDA #$FF
,C102 8D 03 DD STA $DD03
,C105 A9 00     LDA #$00
,C107 EA        NOP
,C108 8D 30 C1 STA $C130
,C10B AD 30 C1 LDA $C130
,C10E 8D 01 DD STA $DD01
,C111 EE 30 C1 INC $C130
,C114 4C 0B C1 JMP $C10B
,C117 00        BRK
```

Program 5-1. This machine-language program is used to generate a sawtooth waveform so you can adjust the circuit trimpots of the DIA converter circuit using an oscilloscope.

is converted into an analog voltage as soon as the data appears on the port pins. After you have it working, you will be able to program an analog output voltage anywhere between zero and five volts in .0195-volt steps using BASIC Program 5-2. The waveforms that can be generated with this *DIA* converter circuit are limited only by your ability to write the program to generate the waveforms.

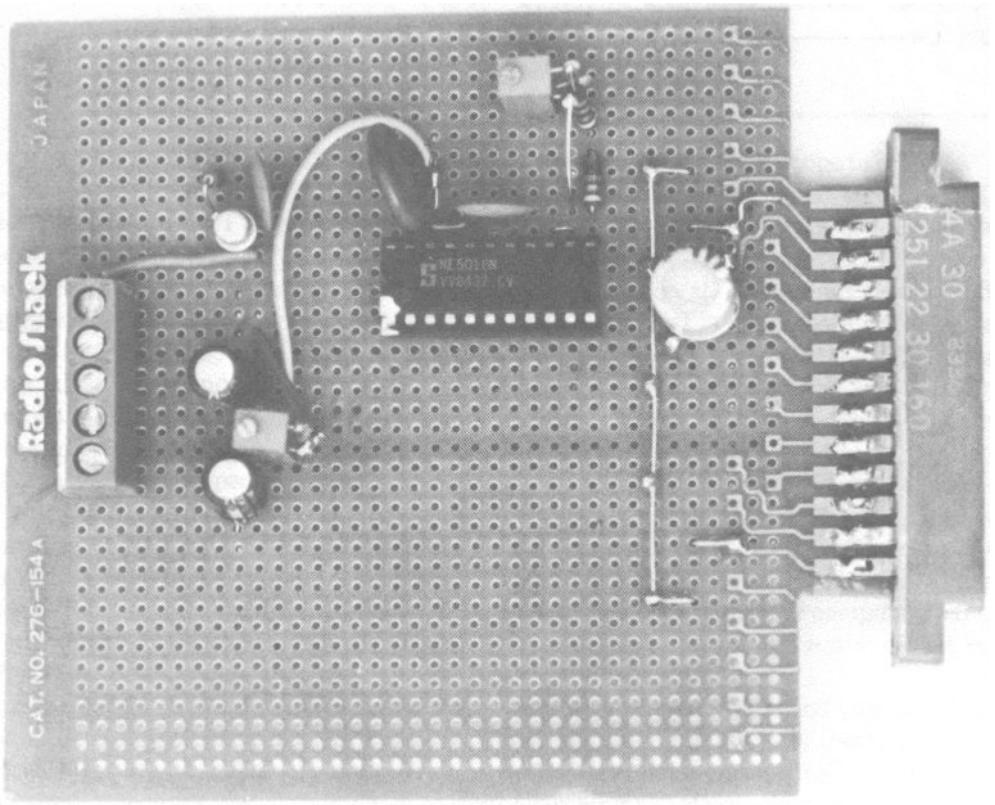


Fig. 5-2. Pictorial view of the top-side of the DIA converter circuit board.

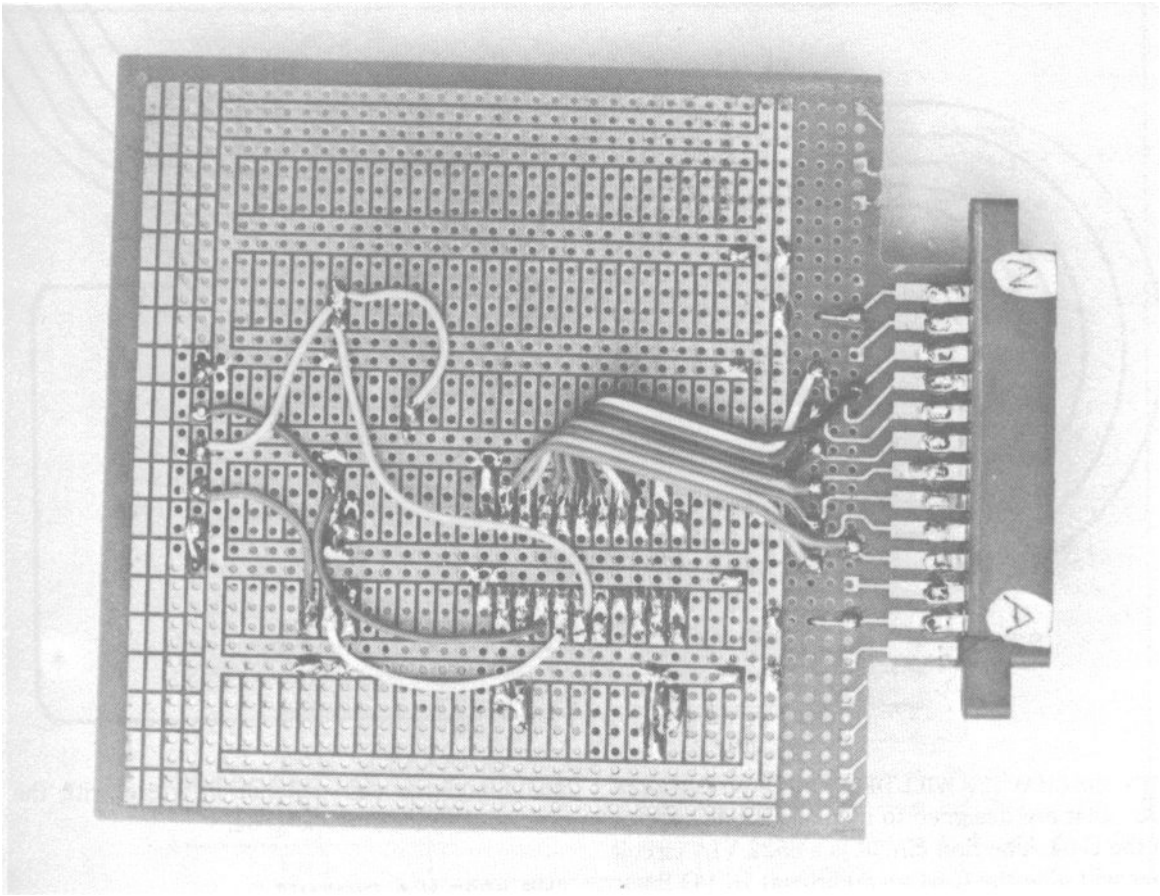
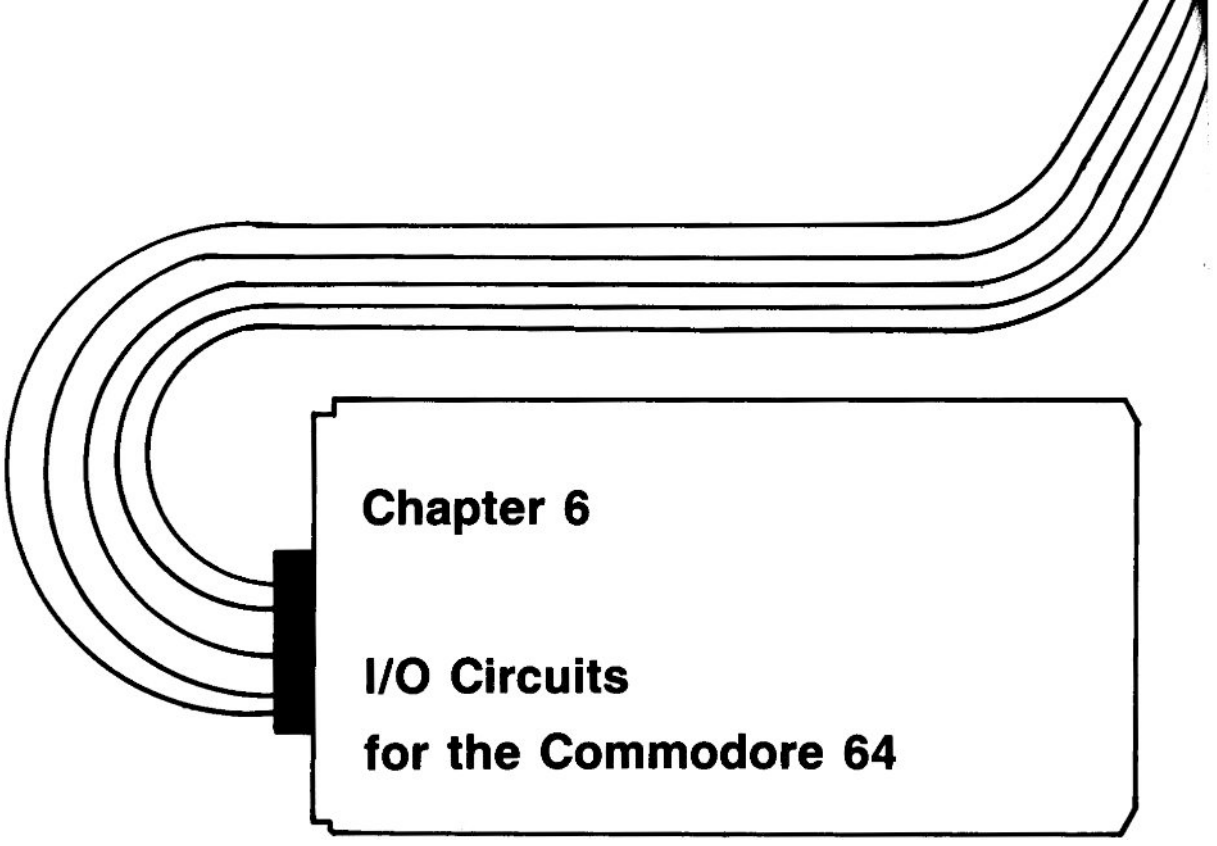


Fig. 5-3. The bottom-view of the *DIA* converter circuit board.

```
5 REM PROGRAM 5.2 FOR THE C-64
10 REM A/D PGM
15 PRINTCHR$(147)
20 POKE56579,255
30 INPUT " INPUT A VOLTAGE BETWEEN                    0 AND 4.99 VOLTS - ";A
40 B=INT(A/.01953125)
50 POKE56577,B
60 PRINT"THE D/A OUTPUT SHOULD BE ";:PRINT INT((B*.01953125)*100)/100;:
65 PRINT" VOLTS"
70 PRINT" ":GOTO30
```

Program 5-2. This BASIC program can be used to generate an analog voltage at the output of the DIA converter circuit board, which is controlled input data from the computer keyboard.



Chapter 6

I/O Circuits for the Commodore 64

THIS eHAPTER WILL DEseRIBE TWO eIReUITS that are designed to increase the *1/0* power of the C-64. The first circuit is a 6522 VIA circuit that will give the C-64 an additional 16 *1/0* lines, and the second circuit is an *AID* converter which will give the C-64 eight programmable analog-to-digital conversion channels. When these circuits are built and used, the C-64 can be turned into a waveform digitizing system for waveform recording.

INTERFACING 110 CIRCUITS TO THE C-64

The C-64's memory map is not at all like the VIC-20's because there are no unused areas of memory in the computer. There are two areas in the C-64's memory map that are intended for I/O functions. They are small areas of memory called I/01 at $DE00 to $DEFF and 1102 at $DF00 to $DFFF. Each of these memory blocks have address-select lines available at the expansion port on pin 7 for 1101 and pin 10 for *1/02.* When you use these two 110 memory blocks for your interface functions, you will not interfere with the general operation of the C-64.

THE 6522 VIA CIRCUIT

The 6522 VIA circuit in Fig. 6-1 is just about the same as the one in Chapter 4 with the exception that U2, a monostable multivibrator TTL chip, has been added to reshape the clock signal coming from the C-64. The VIA circuit can be built on an experimenter's board like most of the other projects in this book.

The complete circuit is really simple because it only uses two IC chips. There are a lot of connecting wires that must be cut and soldered into place, so one must take a little care and use good construction practices to avoid mistakes. The first circuit that should be constructed is the 74121 TTL circuit. When this circuit is finished, adjust trimpot RA to about the midpoint and then complete the rest of the VIA circuit on the board. When the VIA circuit is complete, check it over for solder shorts between the circuit IC pads and use an ohm-

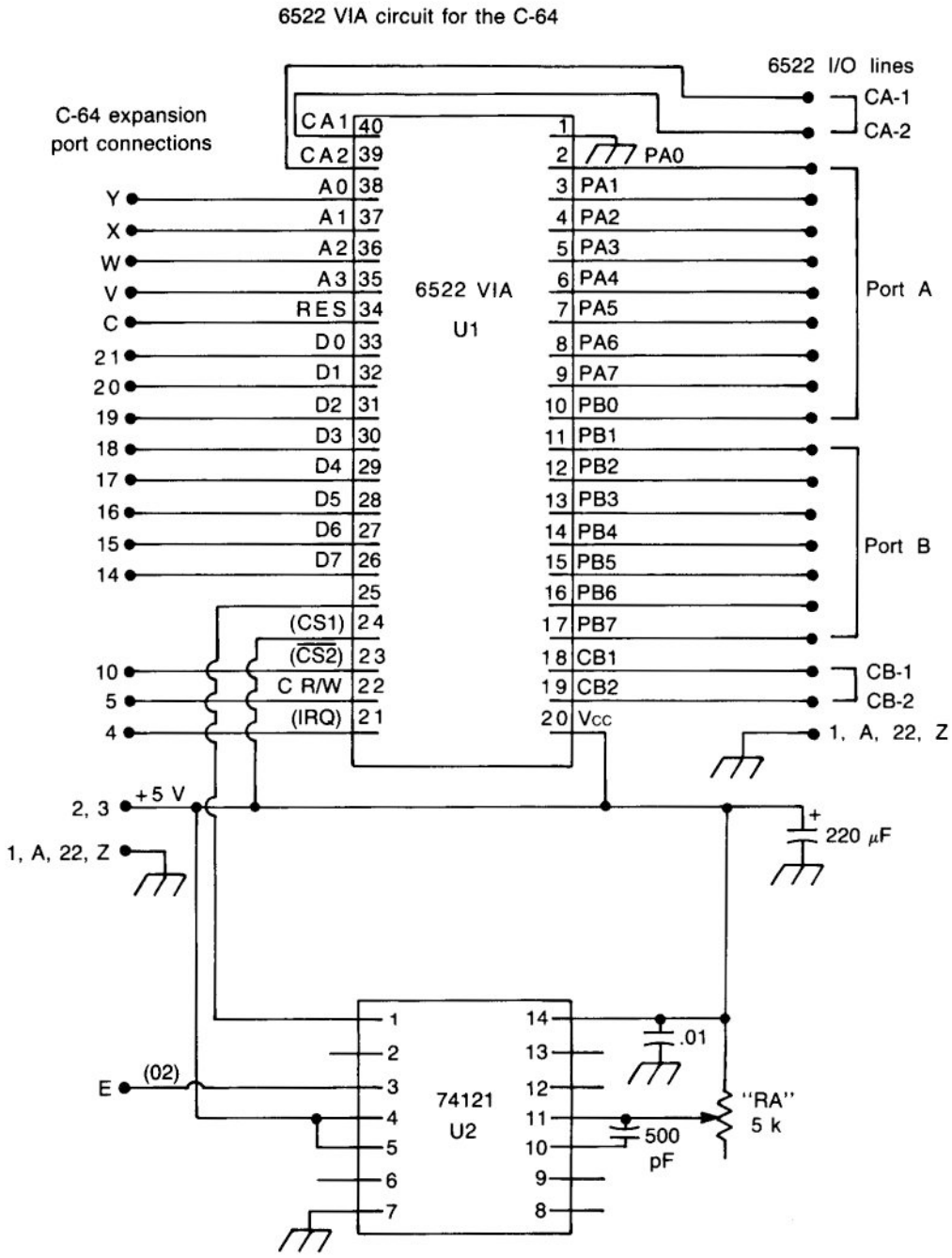


Fig. 6-1. This is the 6522 **VIA** circuit board schematic for the C-64.

meter to check the plus 5-volt lines on pins 2 and 3 for a shorted condition to the circuit's common ground. When you are sure that there are no wiring errors on the board, plug it into the C-64 while the computer is turned off. When you turn on the computer, the C-64 should come on as it generally does. If the video screen does not come up, check to make sure that the data bus wires are connected correctly on the VIA chip. The next step is to adjust trimpot RA. Load and RUN Program 6-1. When RA is adjusted properly, the number 255 shows all the time on the screen. When 255 appears on the screen, find the midpoint of this adjustment and set the trimpot at this point.

When the VIA is up and running, you will have a complete 6522 VIA and all of its extras available to you. I would like to bring up the fact that the timers in the 6526 in the C-64 are a little better than the timers in the 6522 because the 6526 timers can be stopped for a read operation and then restarted.

You can observe from the pictorial views of the VIA board that another experimenter's board with the correct edge-card pin spacings for the C-64 was used to make the plug-in part of the circuit board for the C-64's expansion port.

THE AID CONVERTER CIRCUIT

The analog to digital circuit shown in Fig. 6-2 is designed around an ADC0809, which is an eight bit-eight channel AID converter chip. The converter circuit is designed to complete a conversion in about 60 microseconds, which gives you the ability to record an analog signal with frequency components as high as 1000 hertz. Just for the information of you super experimenters, it is possible to purchase four or five *A/D* chips and select the unit that will operate the fastest. If you are going to do this, you will have to build your own clock circuit instead of using the C-64's clock signal. I have found IC chips that will operate with clock frequencies as high as 2 MHz, which reduces the conversion time to under 35 microseconds or so. This gives you the capability to record faster waveforms.

The main difference between this *AID* circuit and any of the other circuits that have been presented in this book so far is that this circuit uses an elementary form of address decoding. The circuit uses the chip-select signal from pin 10, which places the circuit in the $DF00 to $DFFF address block. IC U4, a 7430, is used to specifically place this circuit in the C-64's memory map at addresses $DFF0 to $DFF8. This address decoding leaves the rest of the memory block open for other I/O circuits.

The construction of this *AID* board is shown in Figs. 6-3 and 6-4. An experimenter's board was again used for this circuit, and one must employ good construction practices to avoid solder shorts between IC circuit pads. The edge-card plug-in for the C-64's expansion port was cut from another PC board that had the correct pin spacings. Another method of connecting to the expansion port is to use a plug-in card from a C-64 game module. These modules can be purchased on sale at very low prices. If some of the needed edge-connector cir-

```
1 REM  PROGRAM 6.1
2 REM  6522 VIA TEST PGM
5 PRINTCHR$(147)
10 PRINT" A 6522 VIA I/O BOARD TEST PGM"
15 PRINT" "
20 PRINT" ADJUST TRIMPOT RA UNTILL THIS PROGRAM   PRINTS '255' CONTINUOUSLY"
25 PRINT" "
30 PRINT" PRESS <T> TO TEST":PRINT" PRESS <S> TO STOP"
35 GET A$:IF A$="T" THEN GOTO 50
40 GOTO35
50 PRINTPEEK(57088)
60 GET A$:IF A$="S" THEN GOTO 5
70 GOTO50
```

Program 6-1. This program is used to test the 6522 VIA board.

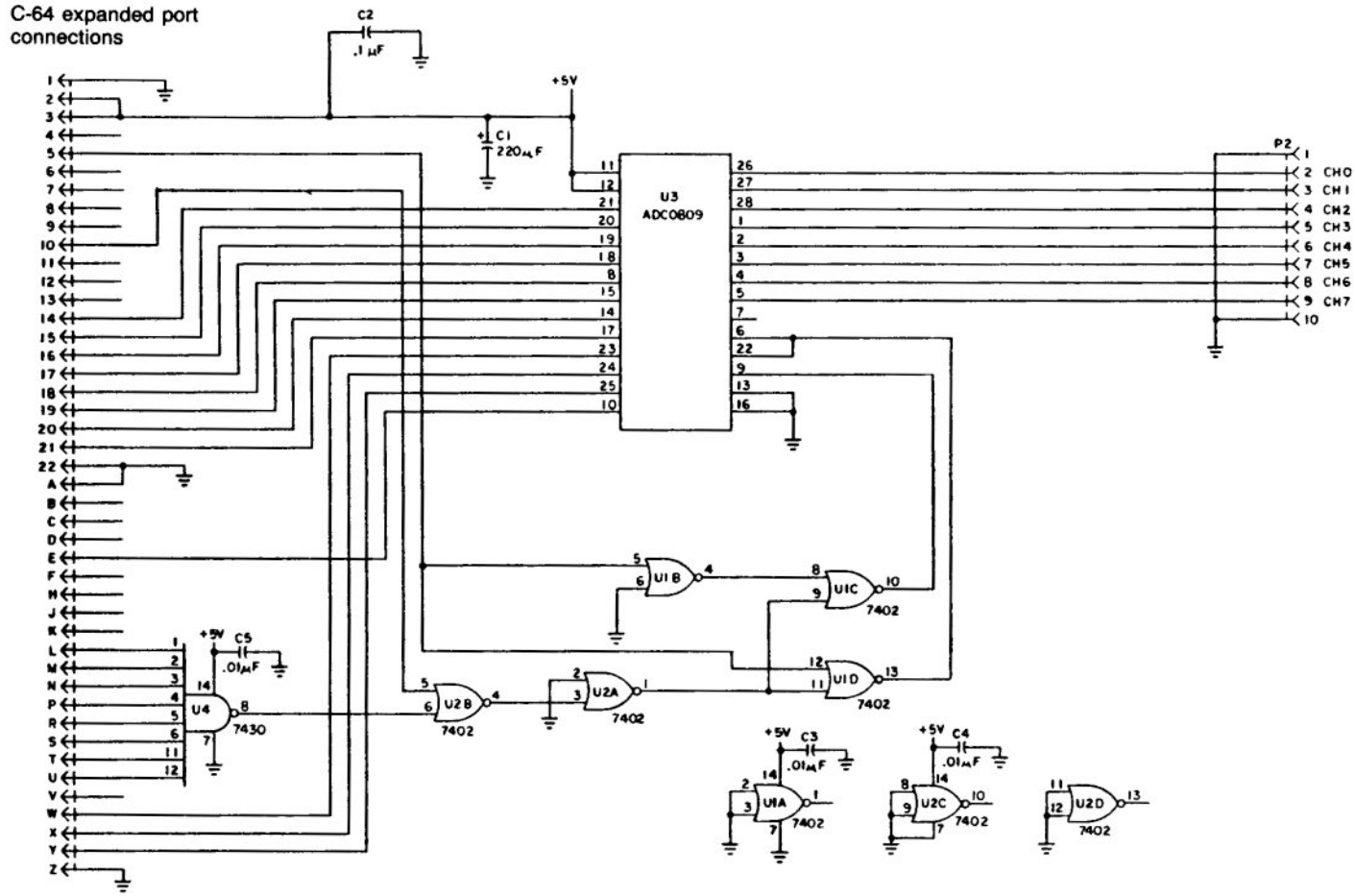


Fig. 6-2. This is the circuit diagram for the C-64 analog-to-digital converter circuit.

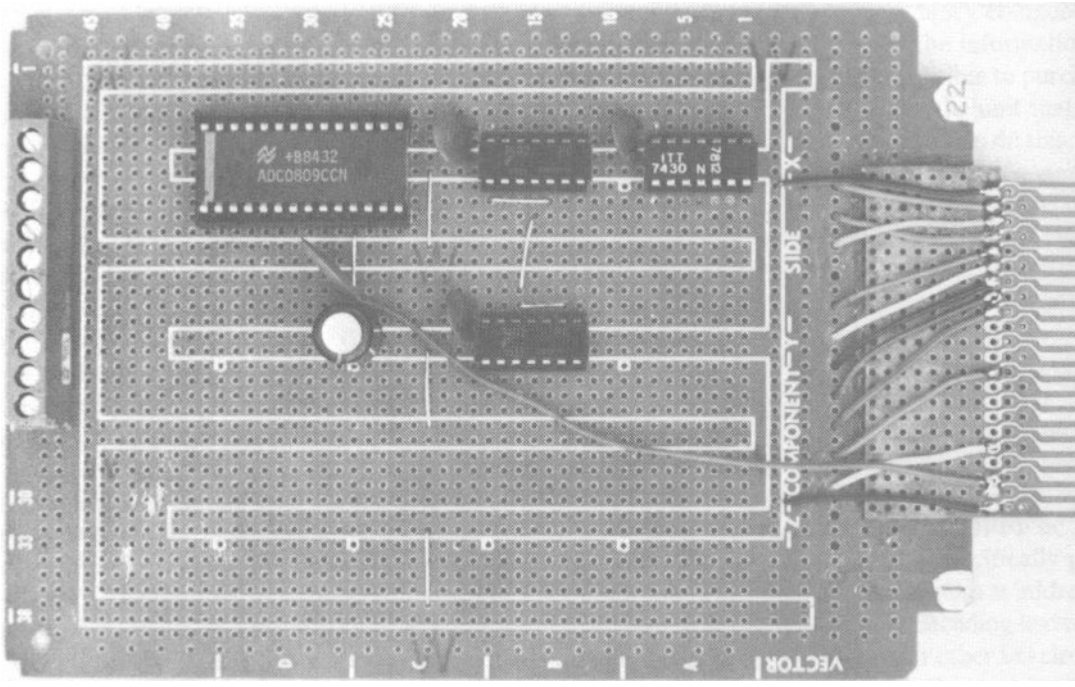


Fig. 6-3. The top-side of the C-64 AID converter board.

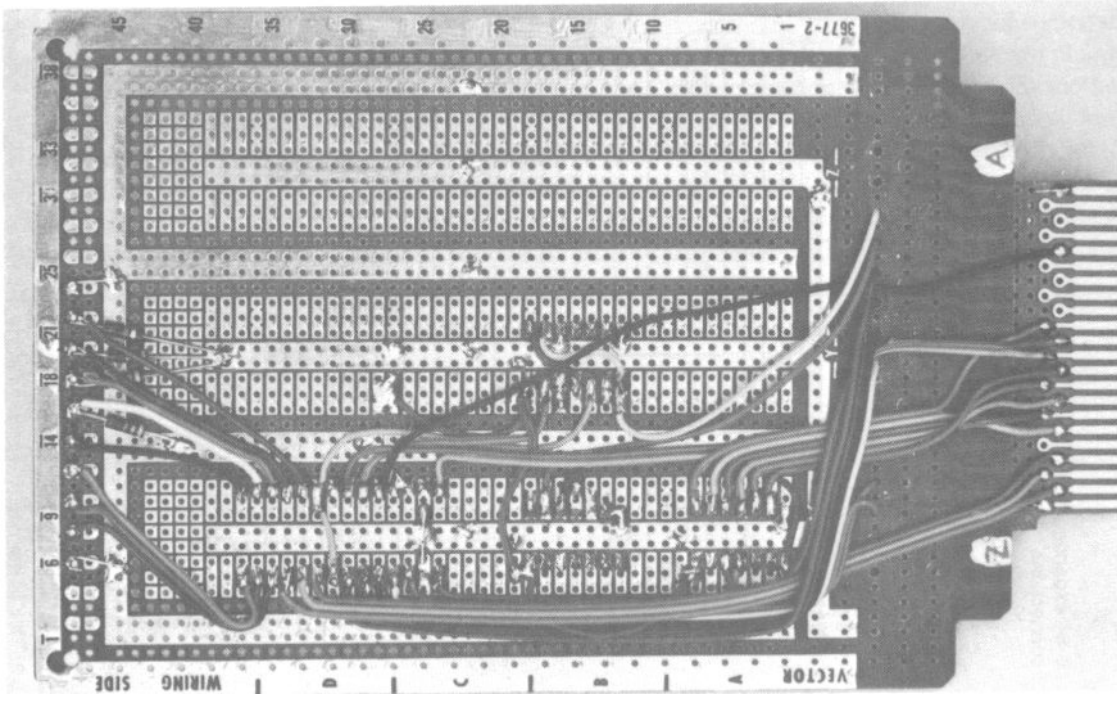


Fig. 6-4. The bottom of the C-64 AID converter board.

```
1 REM  PROGRAM 6.2
2 REM  A/D CONVERTER TEST PGM
5 PRINTCHR$(147):B=57328
10 PRINT"A/D TEST PROGRAM":PRINT" ":PRINT"PRESS (C) TO CHANGE CHANNELS"
20 INPUT"INPUT CHANNEL TO BE TESTED 0 TO 7 - ";A
30 POKE57328+A,00
40 PRINTPEEK(57328+A)
50 GETA$:IF A$="C" THEN GOTO 5
60 GOTO30
```

Program 6-2. This program is used to test the ADC0809 AID board.

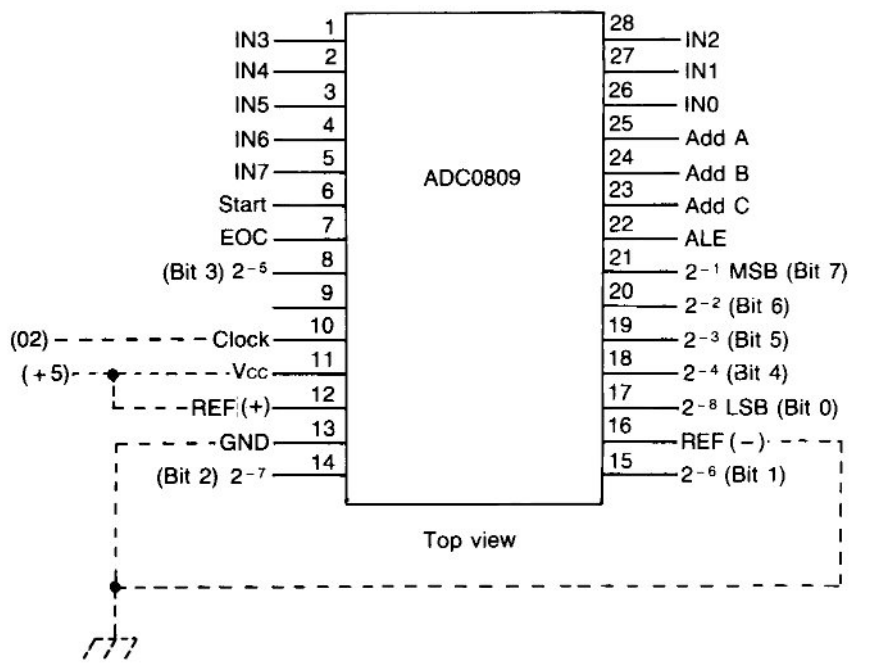
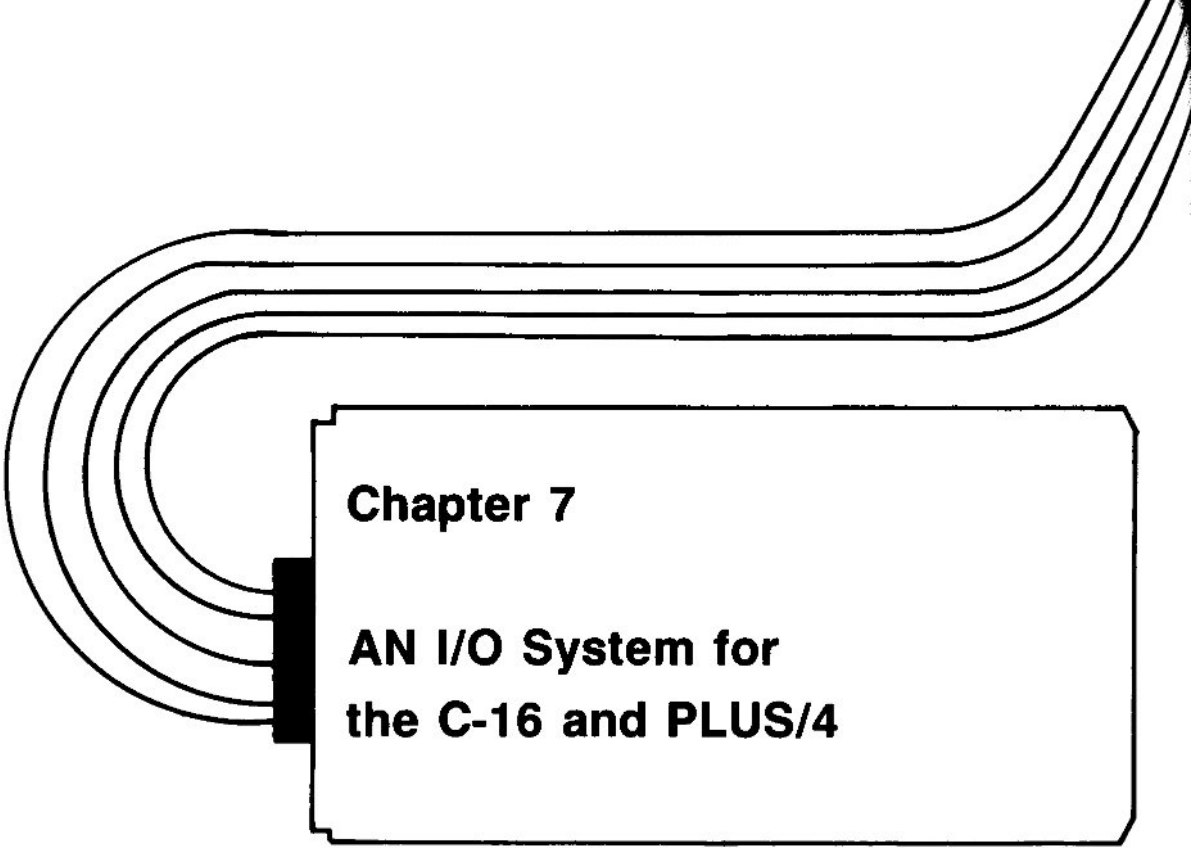


Fig. 6-5. This is the pinout data for the ADC0809 AID converter IC chip.

cuit pads are missing, such as pins 4 and 5, you can go to the local model airplane hobby shop and buy a sheet of thin brass or copper foil. You can then cut thin strips of this foil and glue them to the plug-in module board with five minute epoxy to make the missing copper pad. This method is described for the C-16 and PLUSI4 in detail in the next chapter.

Program 6-2 is presented to help you test the *AID* converter board. Figure 6-5 shows the IC pinout for the ADC0809. When you have your converter circuit up and running, Chapter 9 presents a series of waveform recording programs that use this converter circuit.



Chapter 7

AN I/O System for the C-16 and PLUS/4

THE PLUSI4 AND THE e-16 eOMPUTERS ARE functionally different from the VIC-20 and the C-64 in many ways, but all four of the computers still use the same 6502 machine language. At the writing of this book, the two new computers have only been on the market for about five months, and Commodore has been slow in releasing technical data about the computers. The 110 technical data presented in this chapter has been gained by opening the cases of the two computers and tracing out the circuits. This technical data was used to design the I/O system in this chapter. The I/O system is not a beginner's project because of the number of required 110 boards and the card-cage style construction that is used. You should plan on having an oscilloscope available for troubleshooting if needed or at least some kind of logic probe. If you have been able to understand everything in this book so far, you should be able to build this system if you proceed slowly.

The C-16 and the PLUSI4 are nice computers to use for science and engineering projects because of the computer's built-in graphic commands and machine-language monitor. The addition of the AID converter board and the 110 board that are presented in this chapter will turn either one of the computers into a hi-tech data gathering and displaying system. Even though the construction of this 110 system is a little more complex than the previously presented projects, the end results will justify the required construction effort. When the technical data becomes available in the C-16 and PLUSI4 programmer's guide, you should be able to convert the I/O boards of Chapters 4, 5, and 6 for use with the computers. But for now, we will use the full sixteen line address bus and the required address decoding to operate the *1/0* system. The C-16 and PLUSI4 Expansion Port pinout data that is required for this chapter is presented in Table 7-1.

THE CARD CAGE

A card-cage style of construction was selected for this project because of the number of circuits that were required to build the complete system.

Table 7-1. The Expansion-Port Pin Assignments for the C-16 and the PLUS/4.

1	Ground	A	Ground
2	5 Volts VCC	B	*
3	5 Volts VCC	C	*
4	IRQ	D	*
5	R/W	E	*
6	*	F	A15
7	*	H	A14
8	*	J	A13
9	*	K	A12
10	*	L	A11
11	*	M	A10
12	*	N	A9
13	*	P	A8
14	DB7	R	A7
15	DB6	S	A6
16	DB5	T	A5
17	DB4	U	A4
18	DB3	V	A3
19	DB2	W	A2
20	DB1	X	A1
21	DB0	Y	A0
22	*	Z	*
23	*	AA	*
24	*	BB	*
25	Ground	CC	Ground

NOTES: * - No Pin-out data available.

The Expansion Port pin-out data that is presented here has been secured by tracing the C-16 and PLUS/4 circuit board. Only the specific pin-out data required for the I/O circuits in this chapter is presented.

The complete 110 system requires one address decoder board, at least one I/O port board, and an AID board to complete a good 110 system. Some method of interconnecting and securing the three boards was needed, and a card cage was designed from G-10 copper-clad circuit board material to solve the problem. If you have the ability to build a large complex I/O board, you could build all of the presented I/O circuits on one large experimenter's board.

The completed card cage is shown in Figs. 7-1, 7-2,7-3, and 7-4. Four edge-card slots were used, which will give you the ability to use one address decoder board and three other 110 boards. Each edge-card slot has its own 44-pin edge-card connector. Using the standard 44-pin edge-card connector will allow you to use a number of different brands of experimenter's boards for use with this card cage. The card cage is built from six pieces of copper-clad circuit board material that form the right side, the left side, top support and bottom supports, card slide rails, and the back side, which contains the four edgelcard connectors. All of these circuit-board pieces are soldered together, which makes the cage assembly very easy. If you build these 110 circuits as shown in this chapter, you will need to buy or build a card cage set-up using Figs. 7-1 to 7-4 as a guide.

The hardest part of the card cage assembly will be soldering all of the edge-card connector pins together to form the card cage bus system. All of the same numbered or lettered pins must be soldered to each other as shown in Fig. 7-4. This means that all pin ones' of the four edge connectors along with the pin twos' through to pin 22s' must be soldered together in a bus line style. When all of the numbered pins are connected, solder all of the lettered pins in the same fashion.

THE ADDRESS DECODER BOARD

The address decoder board (build the decoder board first) presented in this chapter is used to select the specific memory locations in the C-16's and PLUSI4's memory for I/O functions. For the purposes of this chapter, the PLUSI4 and the C-16 have the same memory-map configurations with the exception of the limited memory of the C-16. There is an area in their memory map at $FE00 to $FEFF that is to be used for the DMA DISK SYSTEM. This is the memory area that will be used in the *1/0* system of this chapter. The actual memory locations that are selected by this decoder board are $DEF0 to $DEF8 or #65264 to #65273.

The decoder board schematic that is shown in Fig. 7-5 uses two 7430 TTL chips and a 74154 to do the memory decoding along with some 7402 chips for logic switching. The circuit is designed so the two 7430 chips are used to select the $FEFX part of the memory address (X means "don't care") while the 0 through 8 unit digits are selected by the 74154 chip. The two 7402 NOR gates U2A and U2B are used to buffer the readlwrite line. The power supply in the computer is too small to supply

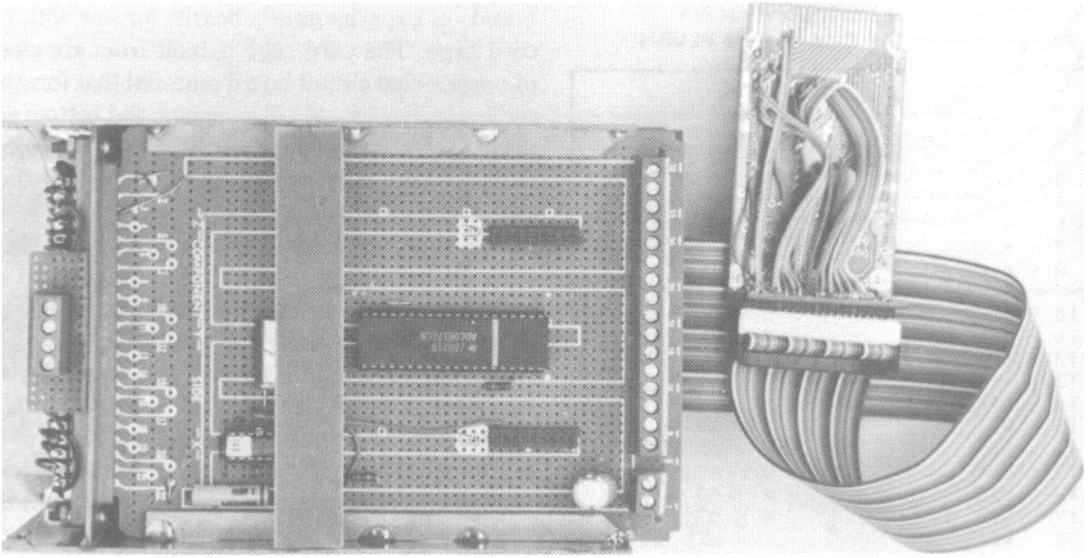


Fig. 7-1. Top-view of the card-cage system with all three boards inserted into the card cage. Note how the slide supports on the side of the AID converter board are soldered to the card-cage sides. Also note the flat forty-conductor ribbon cable with its connector plugged into the computer's Expansion Port plug-in board.

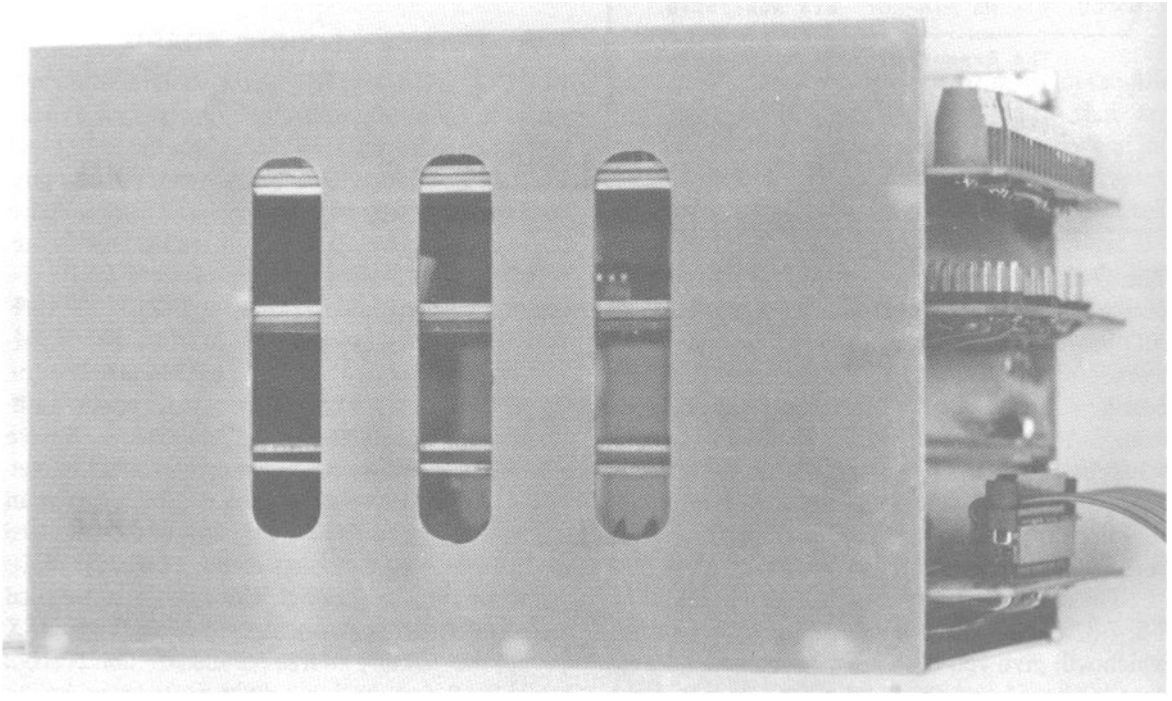


Fig. 7-2. Side-view of the card-cage assembly. The decoder board is plugged into the bottom slot, the 110board is plugged into one of the middle slots, and the AID board is plugged into the top slot. When you are inserting the circuit cards into the plug-in slots, you must use a little care so you will not break any connecting wires on the bottom of the board that is in the slot above.

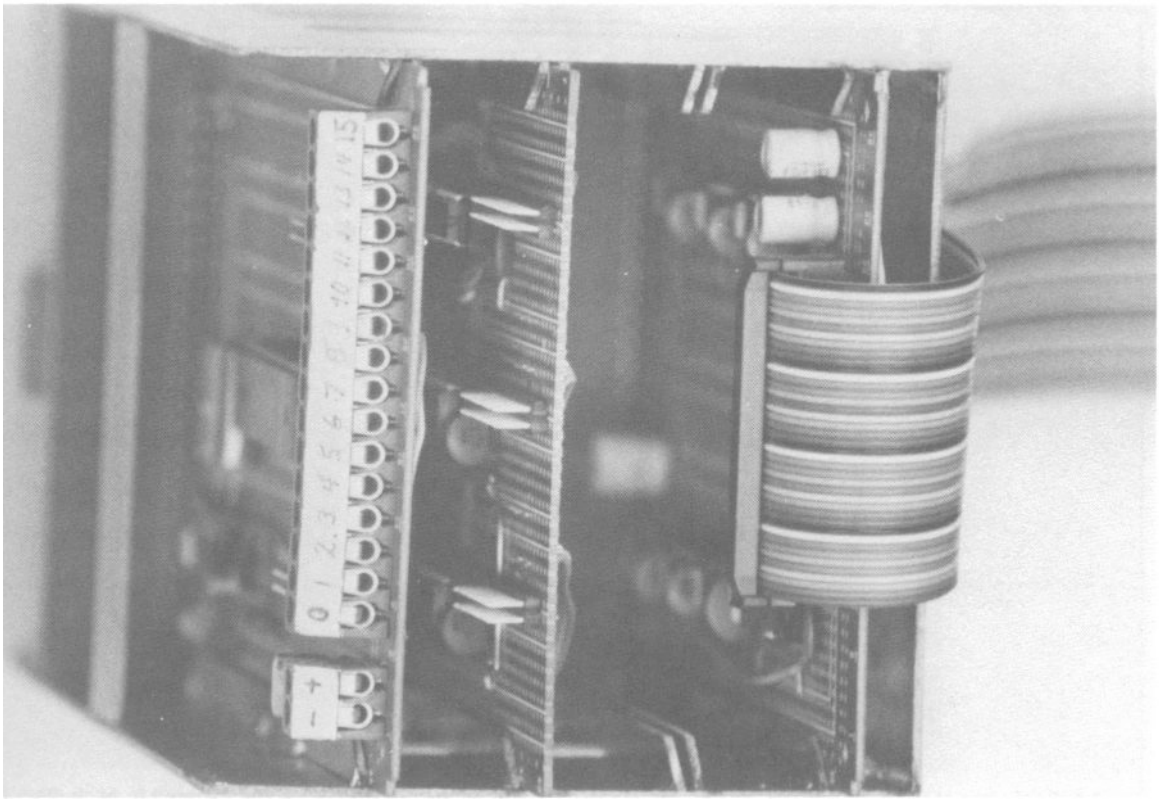


Fig. 7-3. Front-view of the card-cage system that shows how the circuit card slides are used.

the dc power for the card cage, so an external supply must be used. Capacitors CA and CB are used to ac-couple the supply in the computer to the supply for the card cage to prevent power-supply noise problems, while dc-isolating the two supplies from each other. The construction of the decoder board is shown in Figs. 7-6 and 7-7.

After the address decoder board is finished, the next project is to build the connecting cable and the Expansion Port plug-in connector board for the computer. The pinout connector spacing configuration for the PLUSI4 and C-16 are not compatible with any experimenter's board pin-out that one can buy so you will need to make your own plug-in board or modify a game-cartridge circuit board. We will modify a game cartridge circuit board for this project. To modify a game cartridge board, you will have to remove the ROM IC chip and add three edge connector circuit pads for the additional signals that are required for the *1/0* system.

If you look at Figs. 7-8 and 7-9 will see that three additional edge connector circuit pads were added to the game cartridge circuit board to complete the Expansion Port plug-in connector board. These three connector pad strips can be made by cutting thin strips from a sheet of brass or copper foil, which are the same size as the connecting pads on the circuit board. The metal foil material can be secured from a model airplane hobby shop along with some five-minute epoxy. Next, cut the foil strips to the proper length and width of the edge-connector pads, and mix-up some five-minute epoxy. Place a thin coat of five-minute epoxy on the circuit board where you want the additional copper strip and place the copper strip onto the board. Now take a piece of plastic sheet from a bread sack and fold it around the edge connector end of the circuit board (the plastic will not stick to the epoxy).

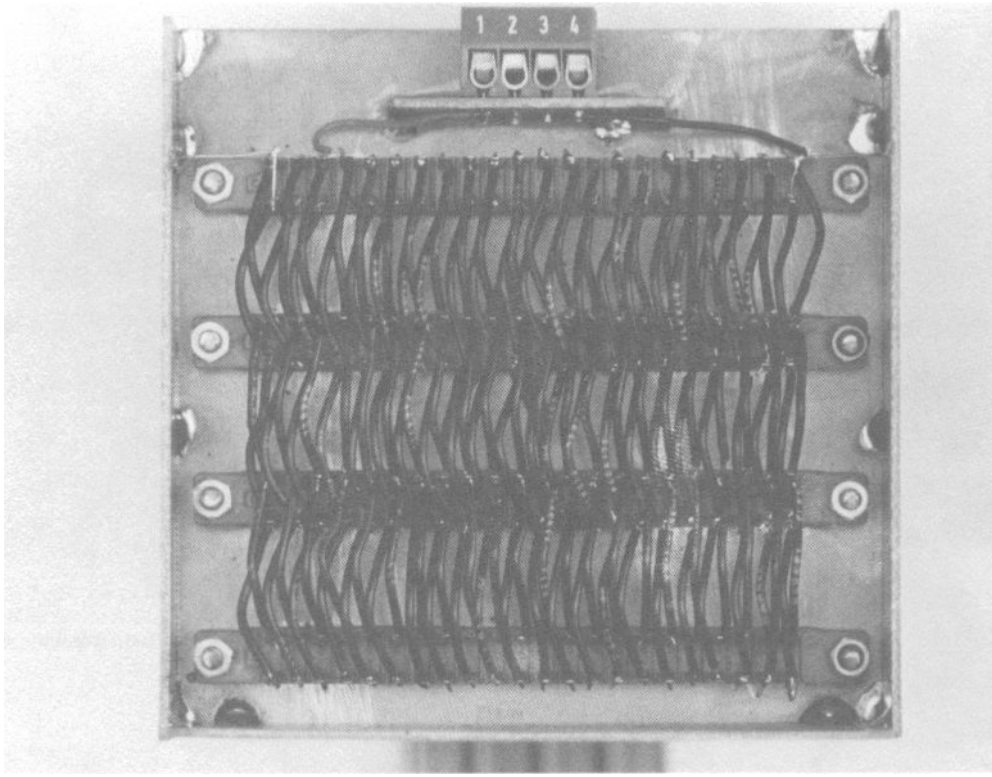
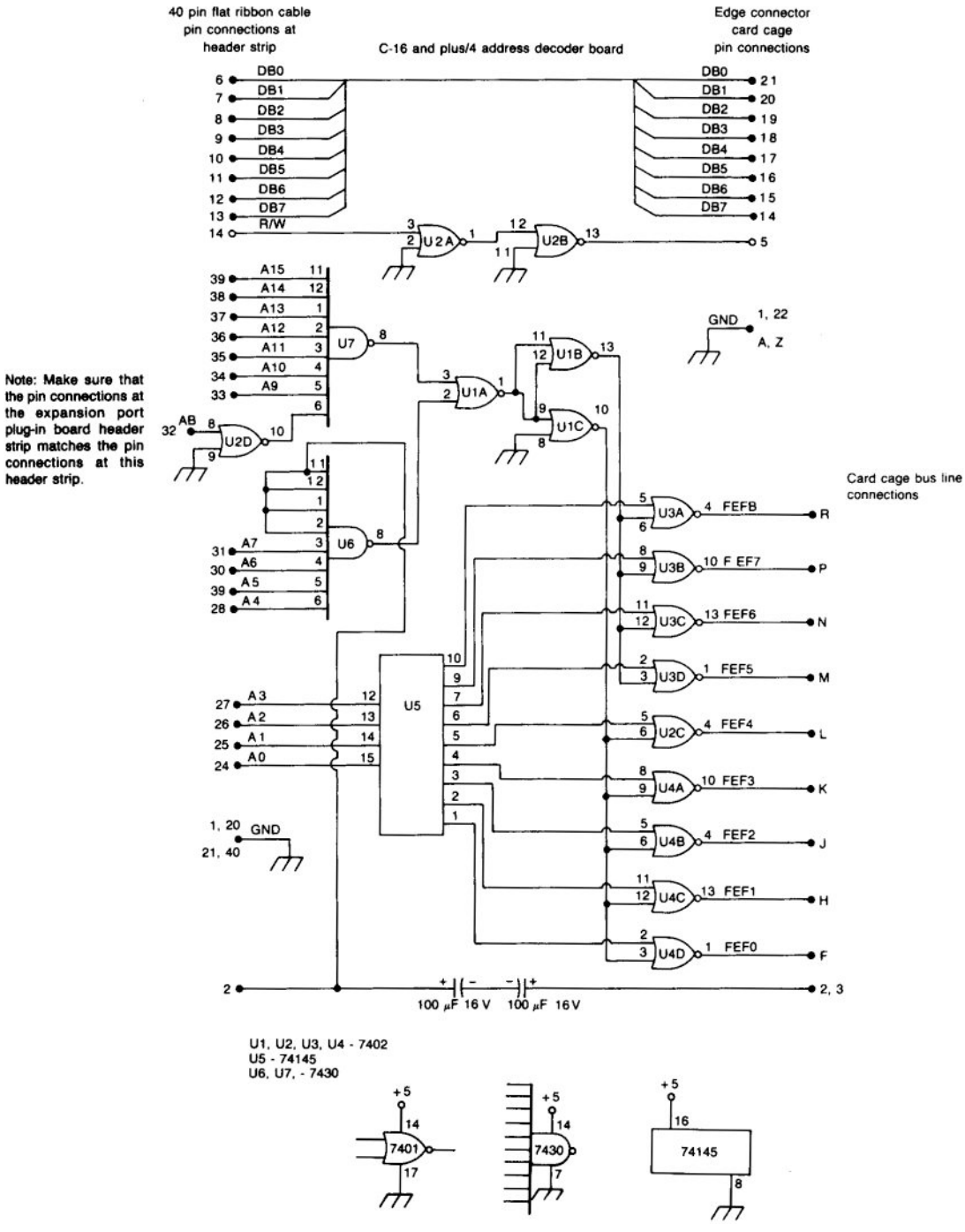


Fig. 7-4. Back-view of the card-cage assembly. Note how all of the edge-connector pins are soldered to each other to form the card cage bus system. The terminal strip at the top is used to connect the 5-volt Vee and the common ground from the power supply to the card cage. The 5 volts goes to terminal number one and then onto pins 2 and 3 of the card-cage bus system. Terminal number 4 is connected to the card-cage copper-clad material and all edge-card connector pins 1, A, 22, and Z.

Place the board in the middle of a large book making sure that the new connecting pad strip does not move on the board. Now, sit on the book for five minutes. After five minutes, the epoxy should be set up and you can remove the circuit board from the book and take off the plastic. You should now have an additional edge-connector circuit pad on the plug-in module that you can solder a wire to for edge connecting purposes. You can use a bench vice in place of a book if you have one. You may also need to clean off the excess epoxy with sand paper if any epoxy gets on top of the copper pads. Some other types of super glue can be used in place of the epoxy.

A forty-conductor flat-ribbon cable is used to take the computer signals from the Expansion Port plug-in connector board to the decoder board. Forty-pin header strip assemblies which matches the forty-pin flat ribbon cable connectors are used on the plug-in connector circuit board and the decoder board for connection purposes. Make sure that the pinouts of these connectors and header strips do not get reversed, although no computer damage will occur if they do. The connection diagram for the plug-in connector board is presented in Fig. 7-10.

When all parts of the address decoder board are completed, that is the decoder circuit board, the Expansion Port plug-in connector board, and the flat ribbon cable, you can test out the decoder circuit by using an oscilloscope or a logic probe and BASIC Program 7-1. This program sets up a simple loop operation that toggles each address decode select line on or off one at a time in the order that you



Note: Make sure that the pin connections at the expansion port plug-in board header strip matches the pin connections at this header strip.

Fig. 7-5. Schematic of the address decoder board.

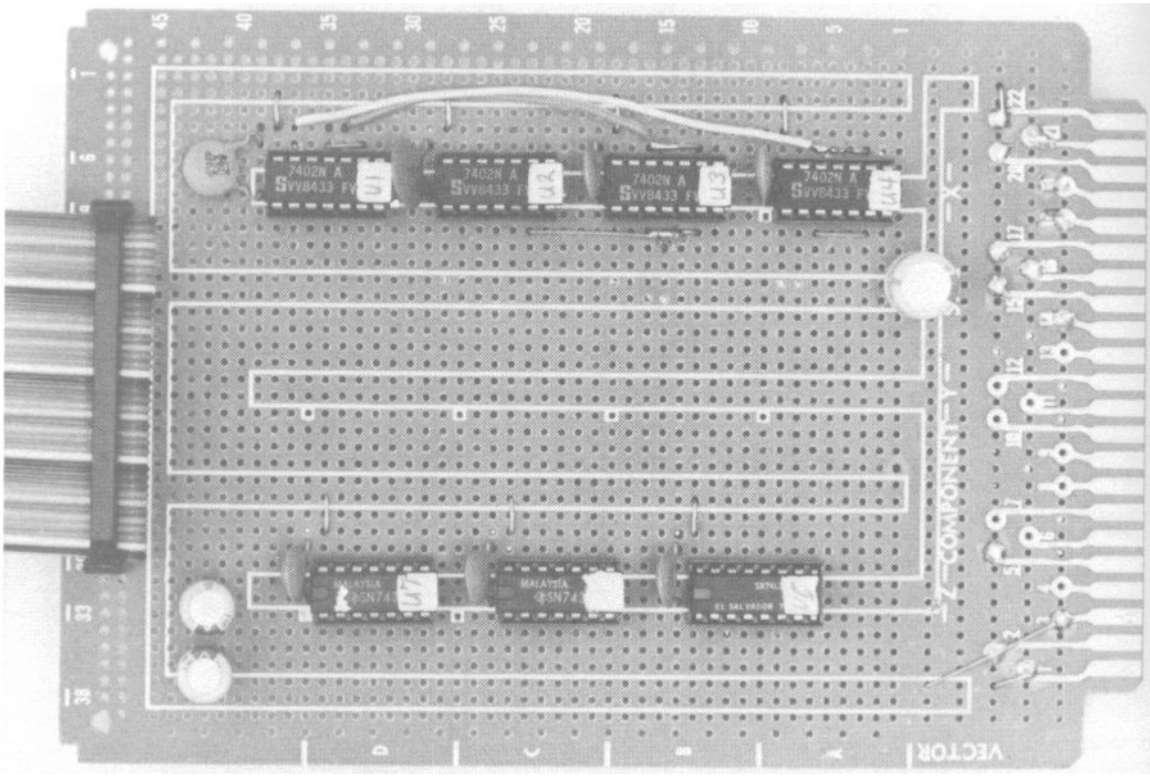


Fig. 7-6. Top-view of the address decoder board. Note the forty-conductor flat ribbon cable that is connected to the header strip on the board.

select. Using this program, you can observe the actual address select signal pulses at the output of the address decoder circuit (pins F to R).

THE I/O PORT BOARD

The I/O Port Board shown in Figs. 7-11 and 7-12 gives you two eight-bit input ports and one eight-bit output port. The best part about this I/O board is that you can buy all of the parts at any good electronics hobby shop. You also can design our own I/O port board in other configurations by using the number of input or output port circuits that is required. A maximum of 9-input or output port circuits can be addressed by the address decoder board. The I/O board schematic is shown in Fig. 7-13.

The input port is designed around a 74LS244, which is an octal based tri-state buffer chip. The term tri-state means that you can program the Ie chip's output buffer lines into a high-impedance state so they will not appear to be connected to the computer's data bus until the TTL chip is selected by the address decoder circuit. The basic circuit and operation of the 74LS244 is shown in Fig. 7-14. The output port is designed around a 74LS373 TTL chip, which is an octal based output latch circuit. When the output latch circuit is selected by the address decoder circuit, the data that is present on the computer's data bus is latched into the chips

output until the next address select pulse. The basic output latch circuit and its functional operation is presented in Fig. 7-15.

When you have your 110 board completed, you can test it out by using Program 7-2. Program 7-2 gives you the ability to read both input ports or toggle the output port off and on.

THE ANALOG-TO-DIGITAL CONVERTER CIRCUIT

The analog-to-digital converter circuit of Fig. 7-16 is designed to work with the *1/0* board that was described previously. It requires one output port circuit to run the *AID* chip's control functions and one inp'ut port to read the converted digital data. The *AID* converter board has two circuits built on it, which are the basic *AID* circuit and a clock circuit. The *AID* circuit uses the ADC0817, which is a sixteen channel *AID* chip that has been used in other projects in this book. The clock circuit is a standard 1 MHz CMOS clock circuit that can be turned off or on by transistor Q1. The actual frequency of this circuit is not too critical, but it should be kept around 1 MHz to secure the highest speed *AID* conversions. The physical construction of the *AID* board is shown in Figs. 7-17 and 7-18.

The AID circuit is designed using a control port and a data port. The control port is connected to an 110 board output port, and the data port is connected to an 110 board input port. Bits 4 through 7 of the control port are used to select the AID channel number. Bit 3 is used to control the output enable line, which places the converted data on data port pins when bit 3 is low and bit 2 is high. Bit 2 is used as the start conversion line. Anytime bit 2 is taken to a logic ZERO and back high, a conversion is started and completed 60 clock cycles later.

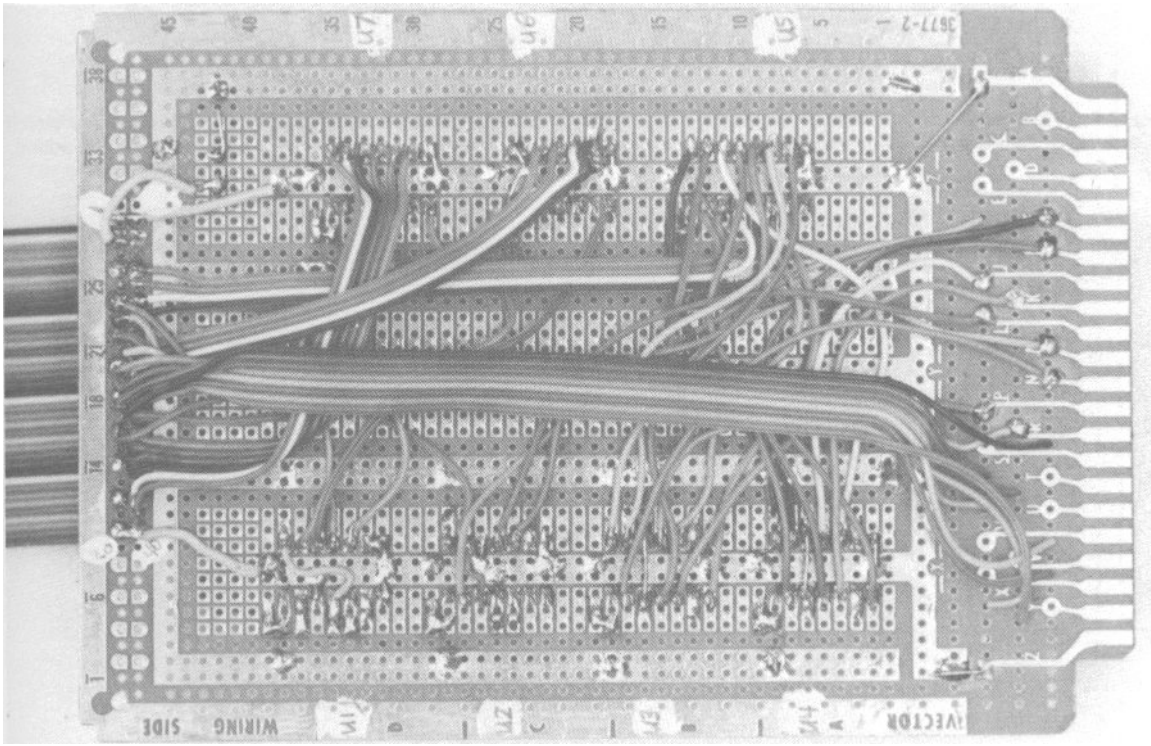


Fig. 7-7. Bottom-view of the address decoder board. Note how the data bus lines are taken directly from the header strip to the edge-card connector pads using flat ribbon cable to form the card-cage data bus lines.

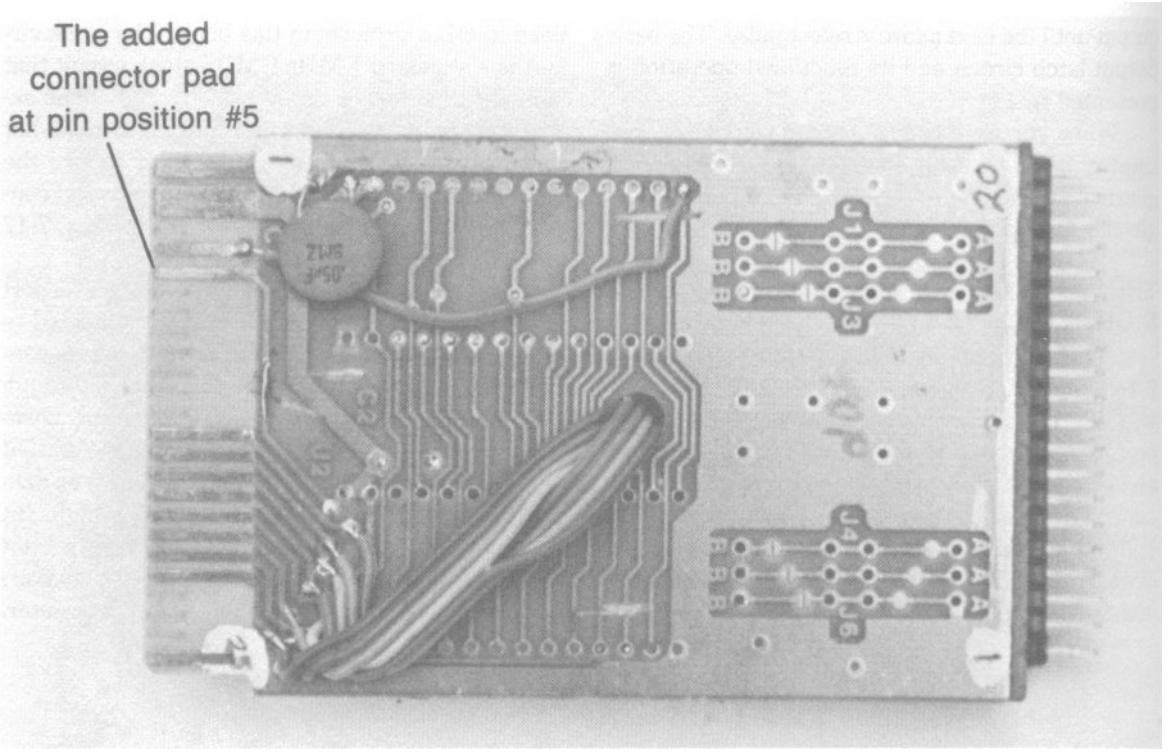


Fig. 7-8. Top-side of the Expansion Port plug-in connector board that was made by modifying a game cartridge circuit board. Note the added edge-connector plug-in circuit pad at pin 5. The text explains how this circuit pad was added.

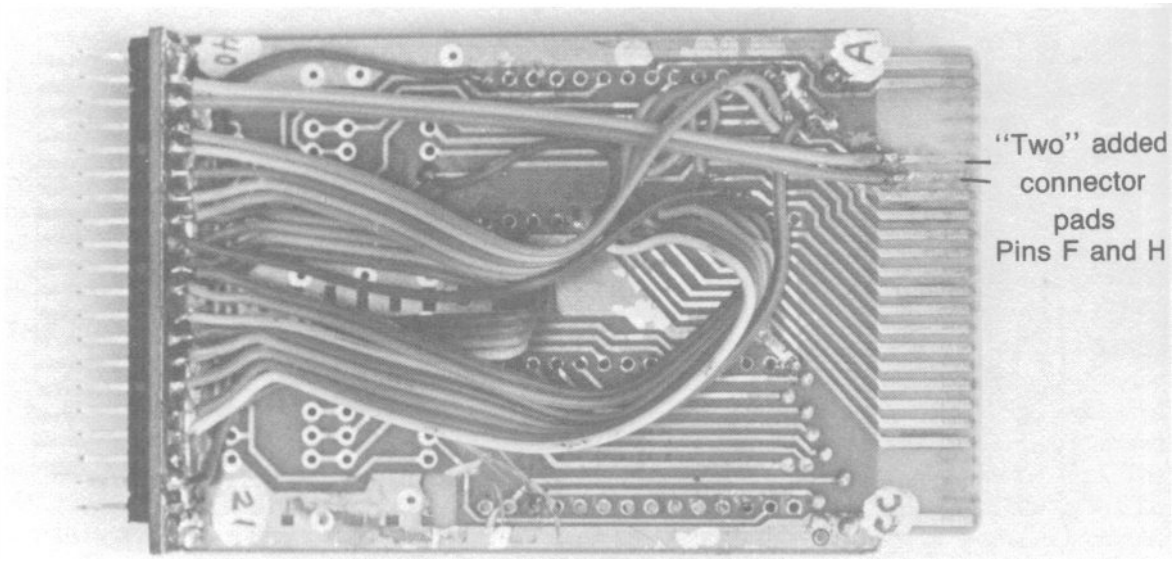


Fig. 7-9. Bottom-side of the Expansion Port plug-in connector board. Note the two added circuit pads at pins F and H. Also, note how the forty-pin header-strip assembly is used in this application.

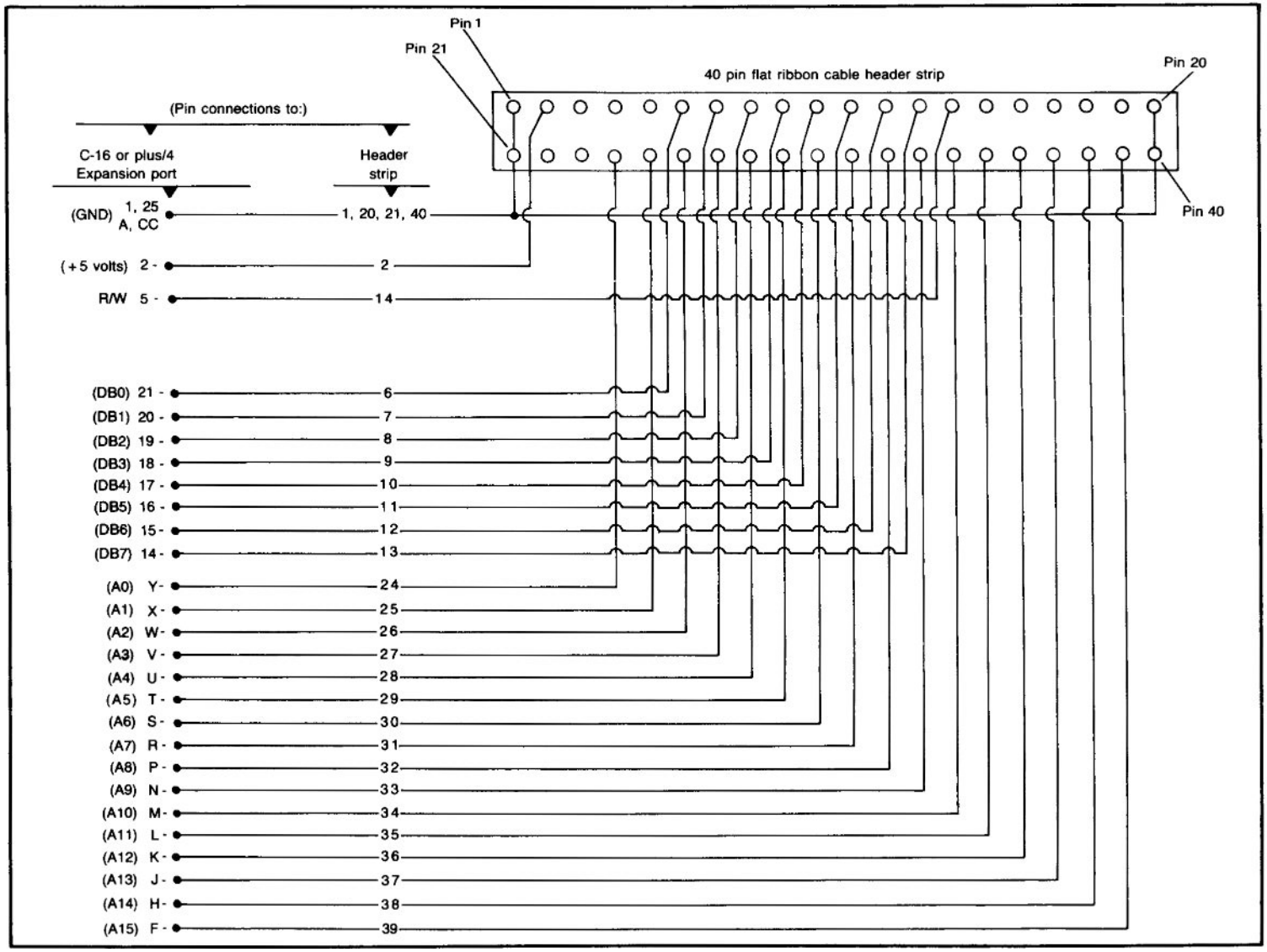


Fig. 7-10. Connection diagram for the Expansion Port plug-in connector board.

```
5 REM PROGRAM 7.1
10 REM DECODER BOARD TEST PROGRAM
20 POKE 28672,169:POKE28673,00:POKE28674,141:POKE28676,254:POKE28677,76
25 POKE 28678,00:POKE28679,112
30 PRINTCHR$(147)
40 PRINT"DECODER BOARD TEST PROGRAM":PRINT" "
50 PRINT"FOR ADDRESSES $FEF0 TO $FEF8":PRINT" "
60 PRINT"INPUT LAST DIGIT OF ADDRESS TO BE        DECODED - 0 TO 8":PRINT" "
70 INPUT "INPUT DIGIT - ";A
80 B=A+240:POKE28675,B
90 PRINT" ":PRINT"TO STOP PROGRAM PRESS THE RESET BUTTON";:
95 PRINT"  WHILE HOLDING DOWN THE RUN/STOP KEY"
100 SYS28672
```

Program 7-1. The decoder board test program.

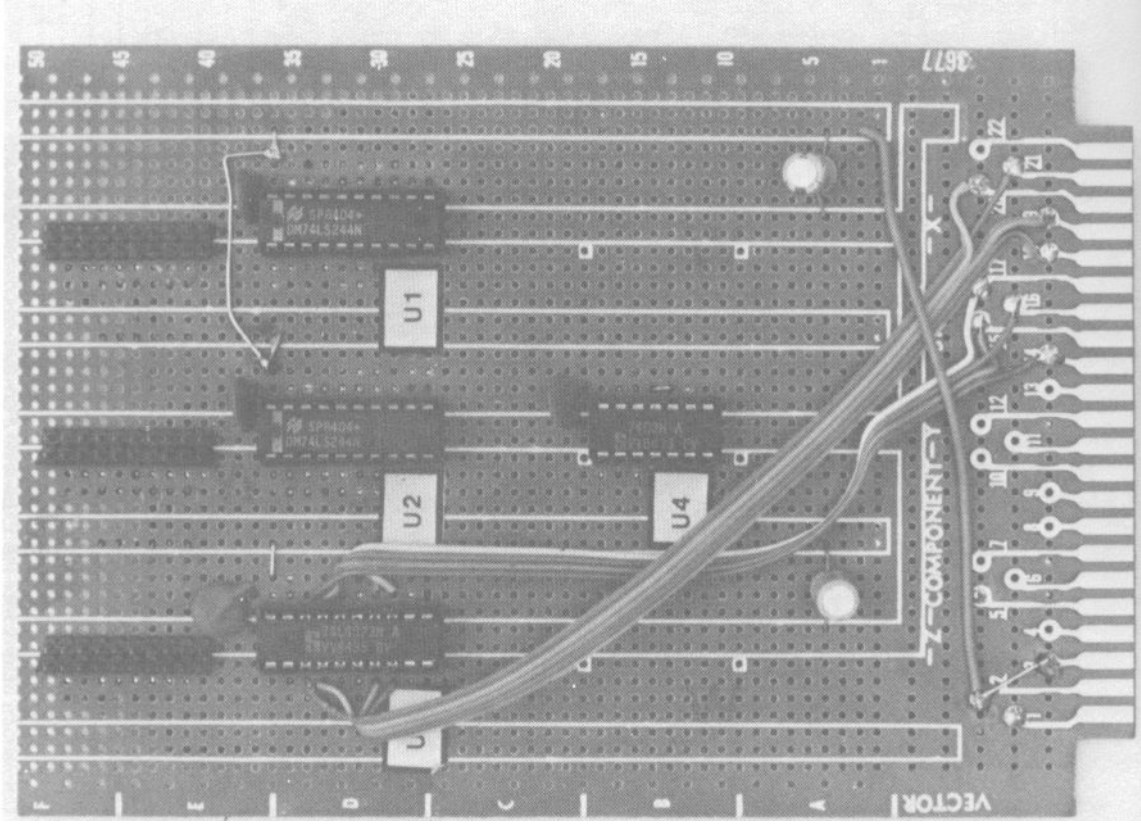


Fig. 7-11. A pictorial view of the top-side of the *1/0* Board.

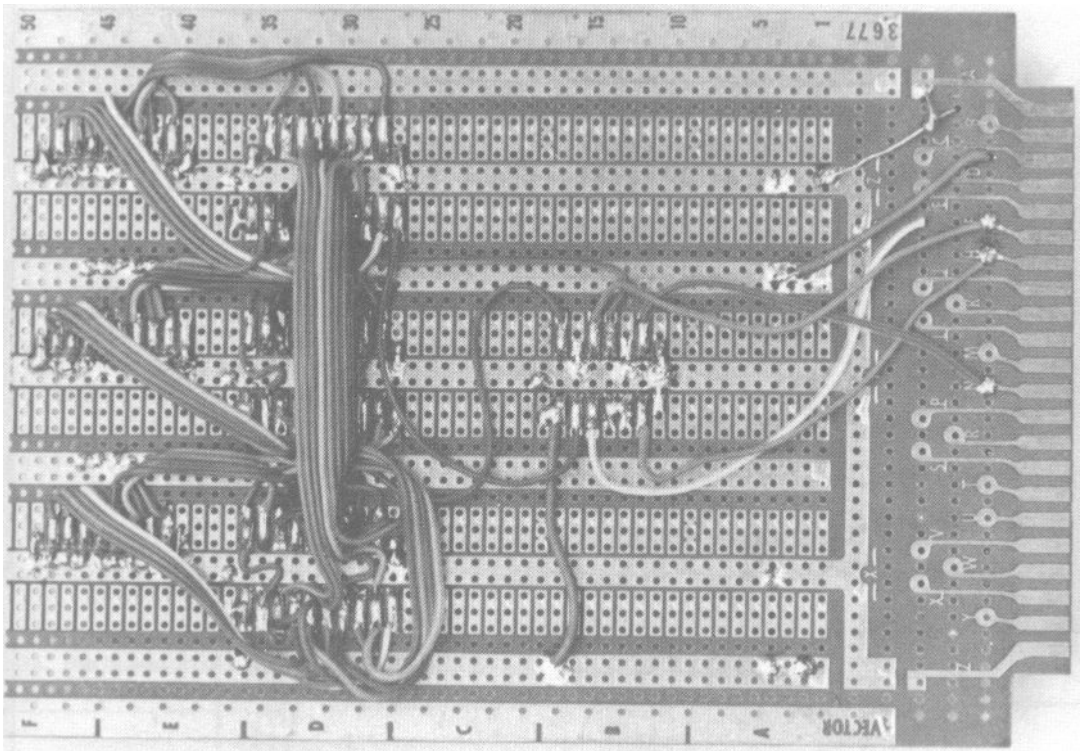


Fig. 7 12. Bottom-side of the *1/0* Board. Note how the multicolored flat cables are used to run the data bus lines.

After the conversion is completed, the converted data can be read by the computer by making control bit 3 a logic ZERO, control bit 2 a logic ONE, and PEEKing the input port memory location that is assigned to the 110 board input port that is connected to the *AID* data port. Control bit 1 is not used in this circuit, and bit 0 is used to turn the clock oscillator on and off.

BASIC Program 7-3 is presented to test out the *A/D* converter circuit. While testing out the converter board, you will want to connect the converter inputs to ground or Vee voltage so you will observe the operation of the *AID* converter. A zero voltage will generate a 000 while a 5 volts Vee voltage will generate a 255 on the video monitor.

CONCLUSION

Once you have this *1/0* system working, you can run any *AID* program in this book on the C-16 or the PLUSI4 with a little conversion work. You will not have to use any of the special plug-in ROMs for machine-language or graphics functions because the C-16 and the PLUSI4 have these functions built into their systems. It is actually easier to use graphics and text together on the PLUSI4 and the C-16 than it is with the C-64.

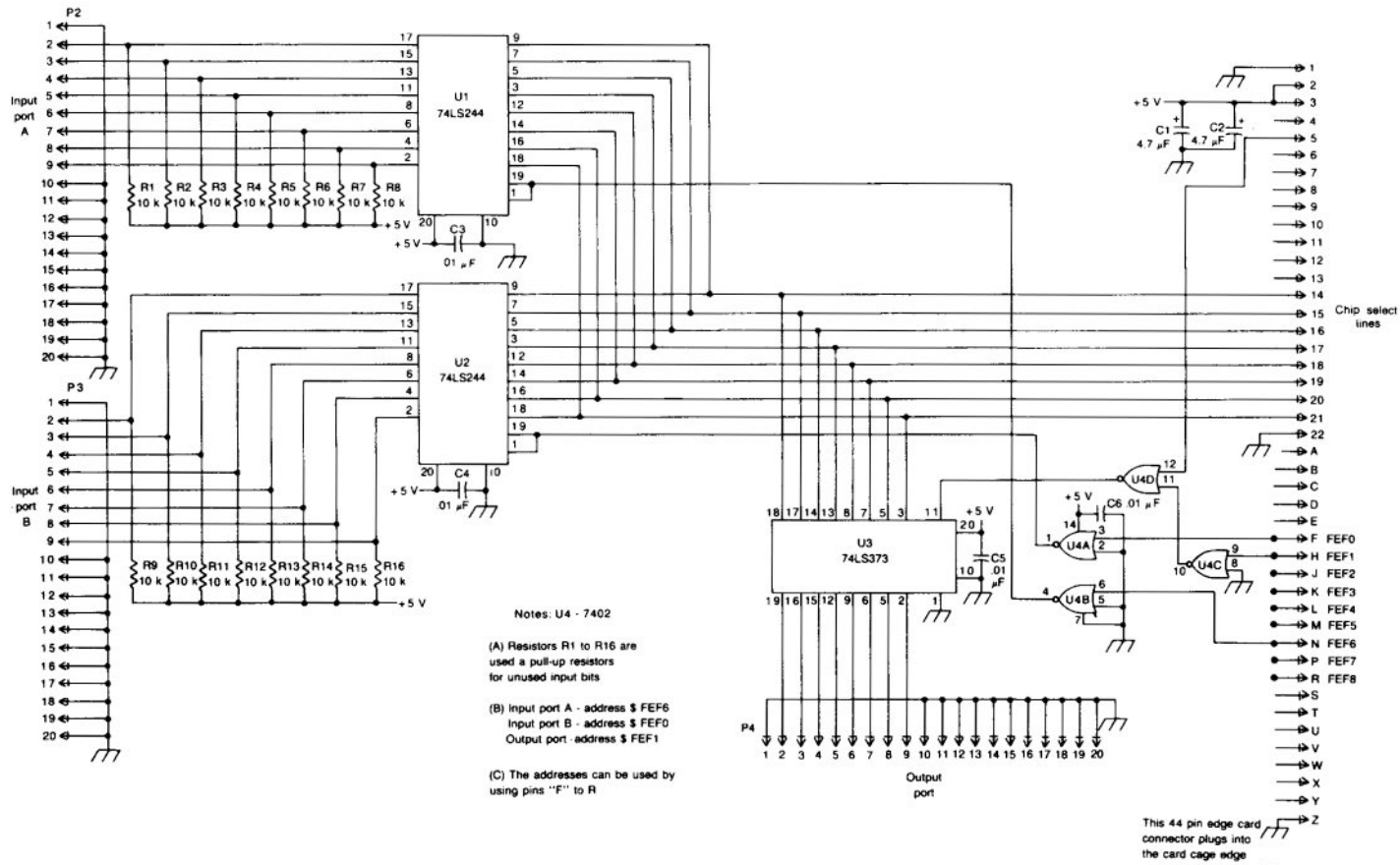


Fig. 7-13. Schematic diagram for the I/O board. The memory address that is assigned to the I/O ports can be changed by connecting U4A, U4B, and U4C to different address select lines which are pins F through R.

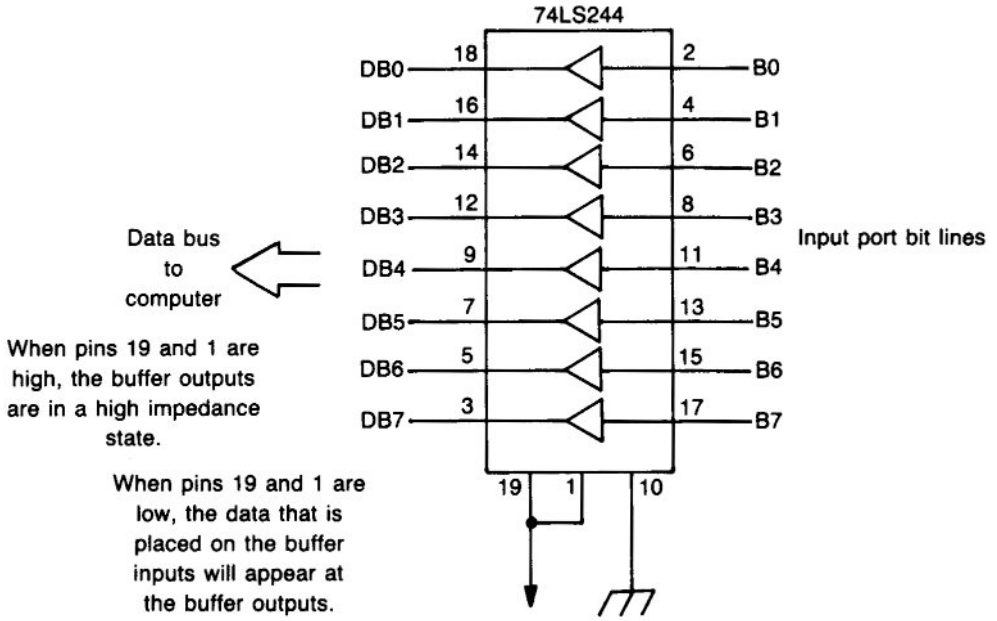


Fig. 7-14. The operation of the 74LS244 input IC chip.

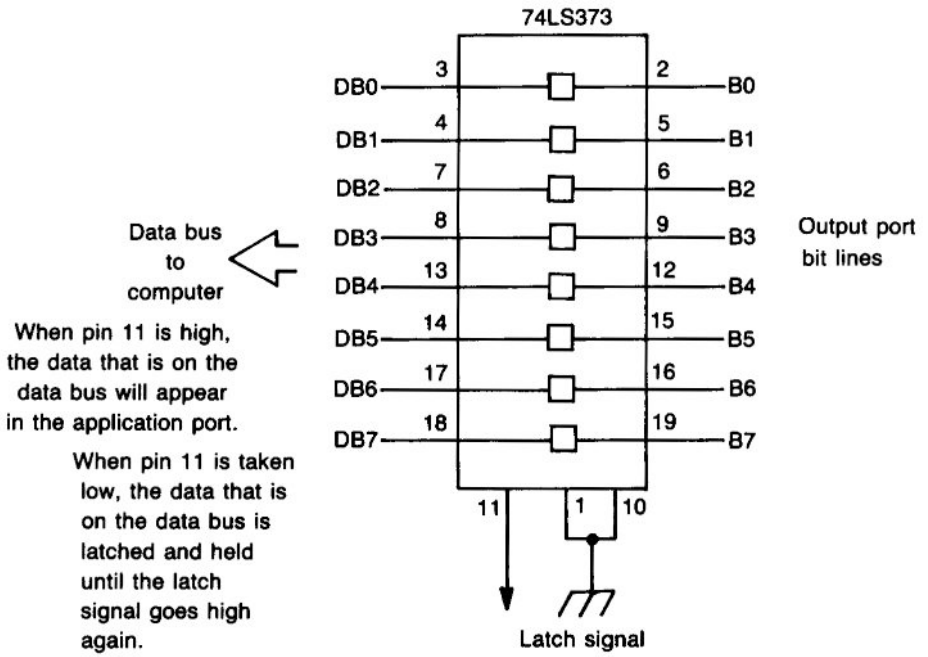


Fig. 7-15. The operation of the 74LS373 output IC chip.

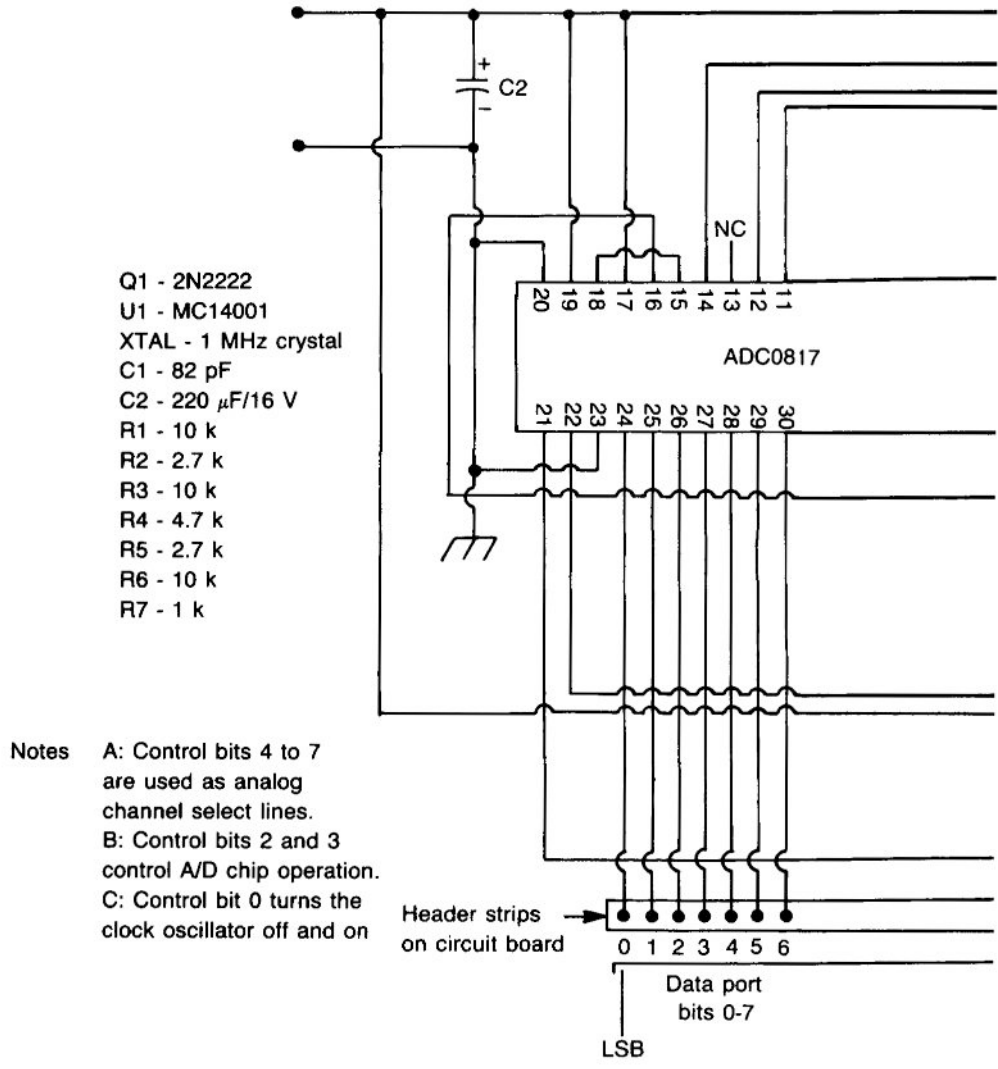
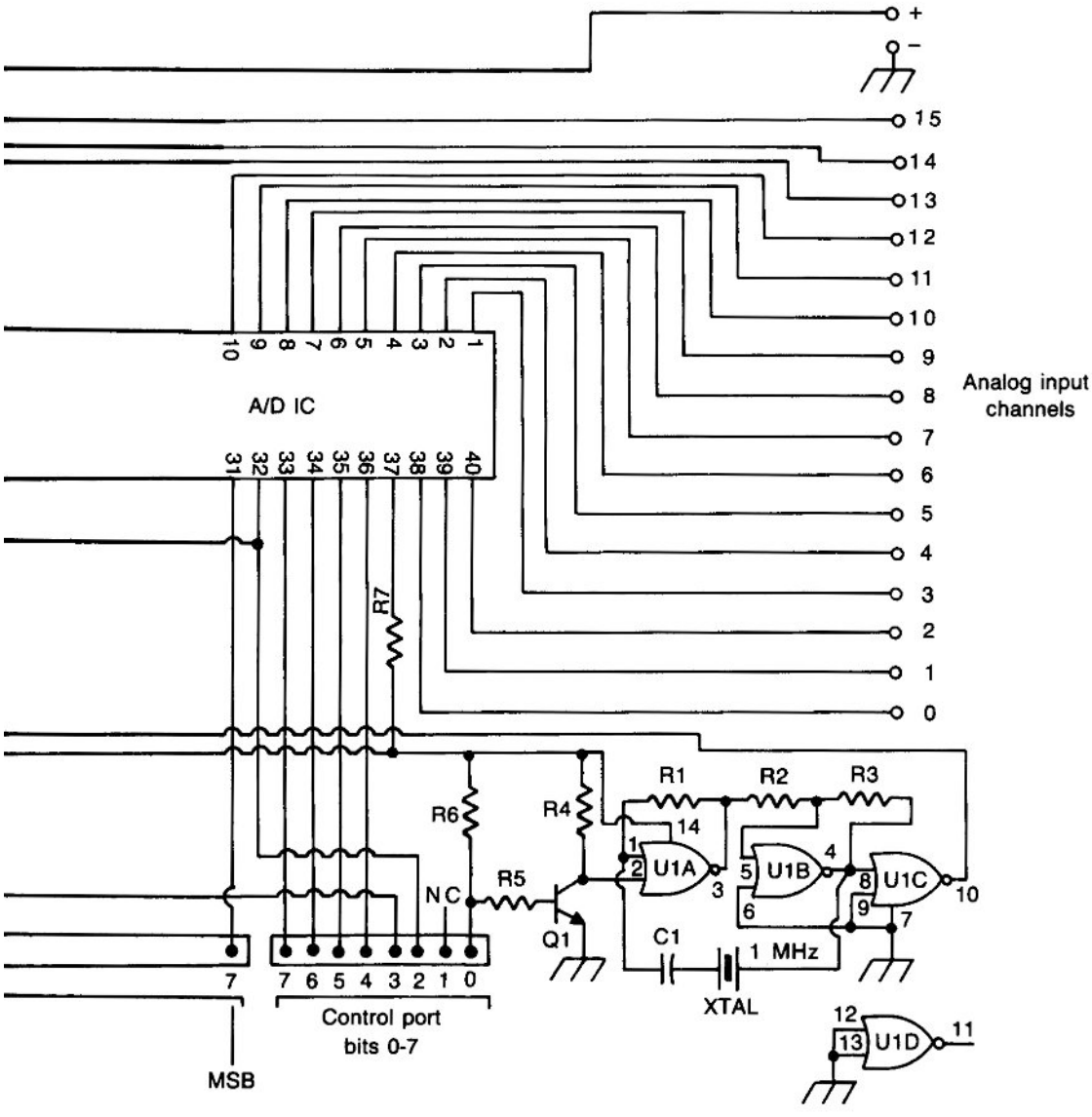


Fig. 7-16. Schematic for the card-cage analog-to-digital converter circuit.


+
−
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Analog input channels
A/D IC
10 9 8 7 6 5 4 3 2 1
31 32 33 34 35 36 37 38 39 40
R7
R6
R4
R5
R1
R2
R3
NC
Q1
U1A
U1B
U1C
U1D
14
1
2
3
5
4
6
8
9
7
10
12
13
11
C1
1 MHz
XTAL
7
7 6 5 4 3 2 1 0
Control port bits 0-7
MSB

```
5 REM PROGRAM 7.2
10 PRINTCHR$(147):PRINT"I/O BOARD TEST PROGRAM"
20 PRINT"PRESS <I> TO TEST INPUT PORTS OR <O> TO TEST OUTPUT PORT"
30 GET A$
35 IF A$="I" THEN GOTO 100
40 IF A$="O" THEN GOTO 200
50 GOTO 30
100 PRINTPEEK(65264);:PRINTPEEK(65270): GOTO 100
200 POKE 65265,00: FOR I=1 TO 500:NEXT
210 POKE 65265,255: FOR I=1 TO 500:NEXT
220 GOTO 200
```

Program 7-2. The *1/0* board test program. If you change the address locations of the input or output ports to something different from the *1/0* board schematic, you will have to change the PEEK and POKE locations in lines 100, 200, and 210 to use this program.

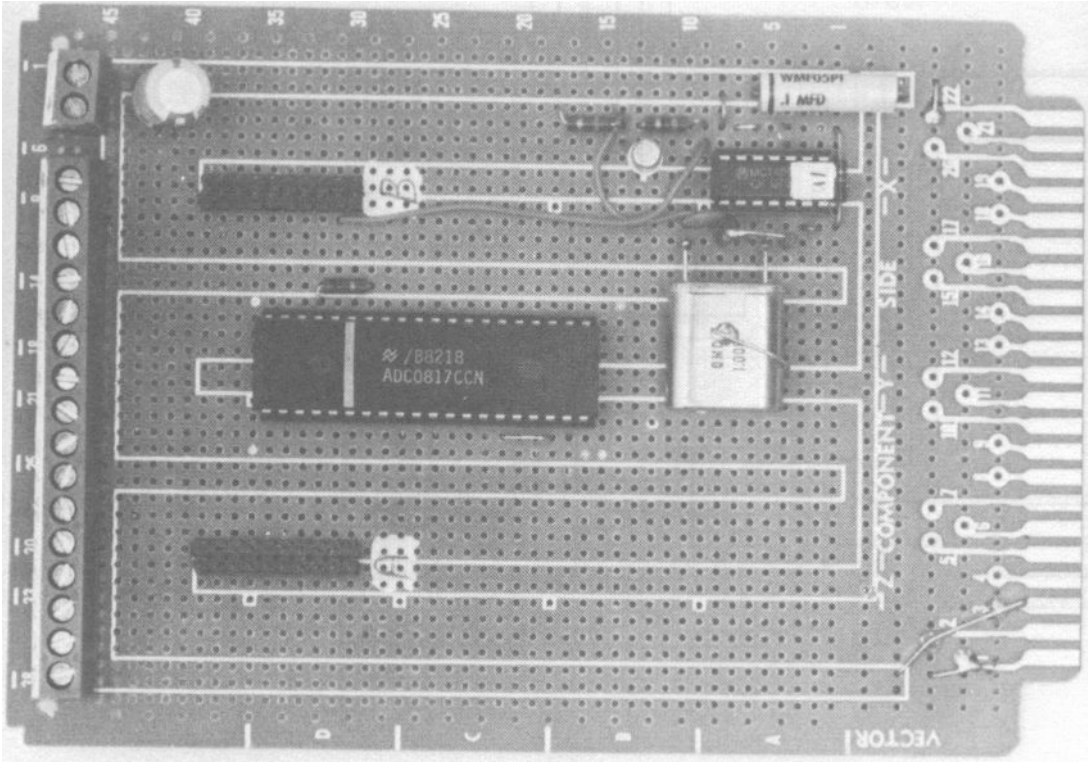


Fig. 7-17. Top-side of the AID circuit board. In this picture, the header strip marked "B" is the control-port pins and the header strip marked "A" is the data-port pins.

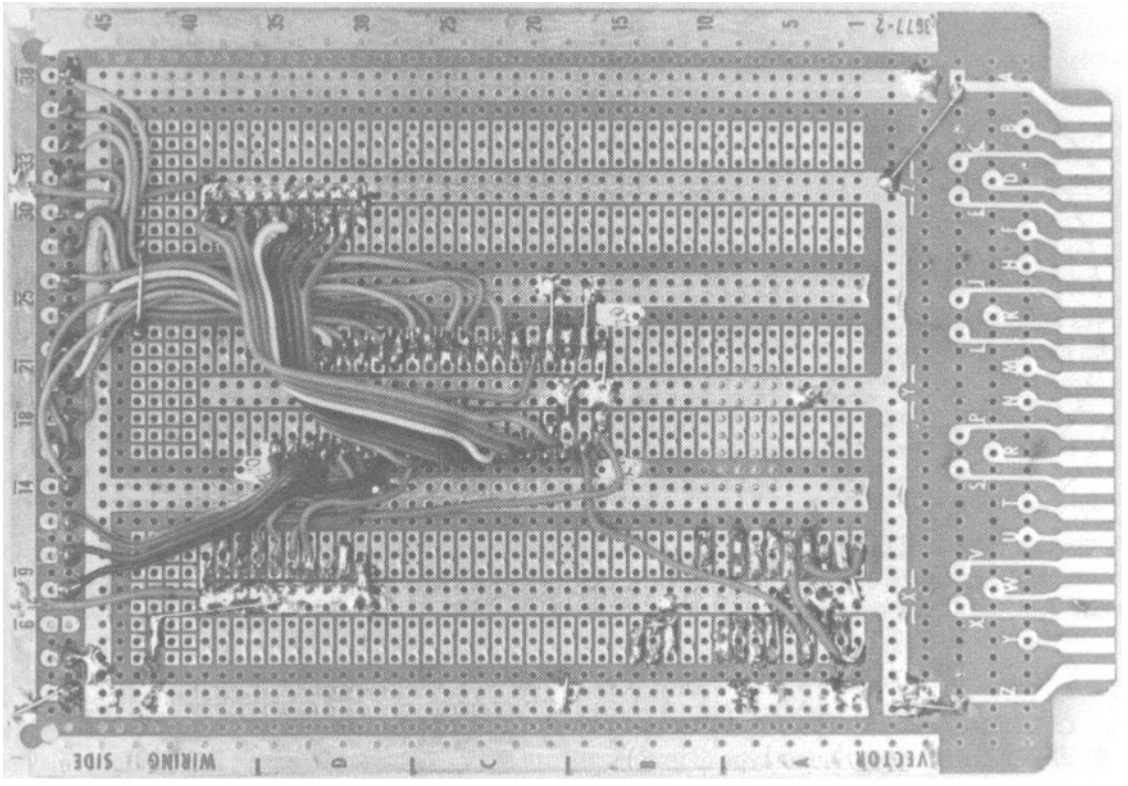
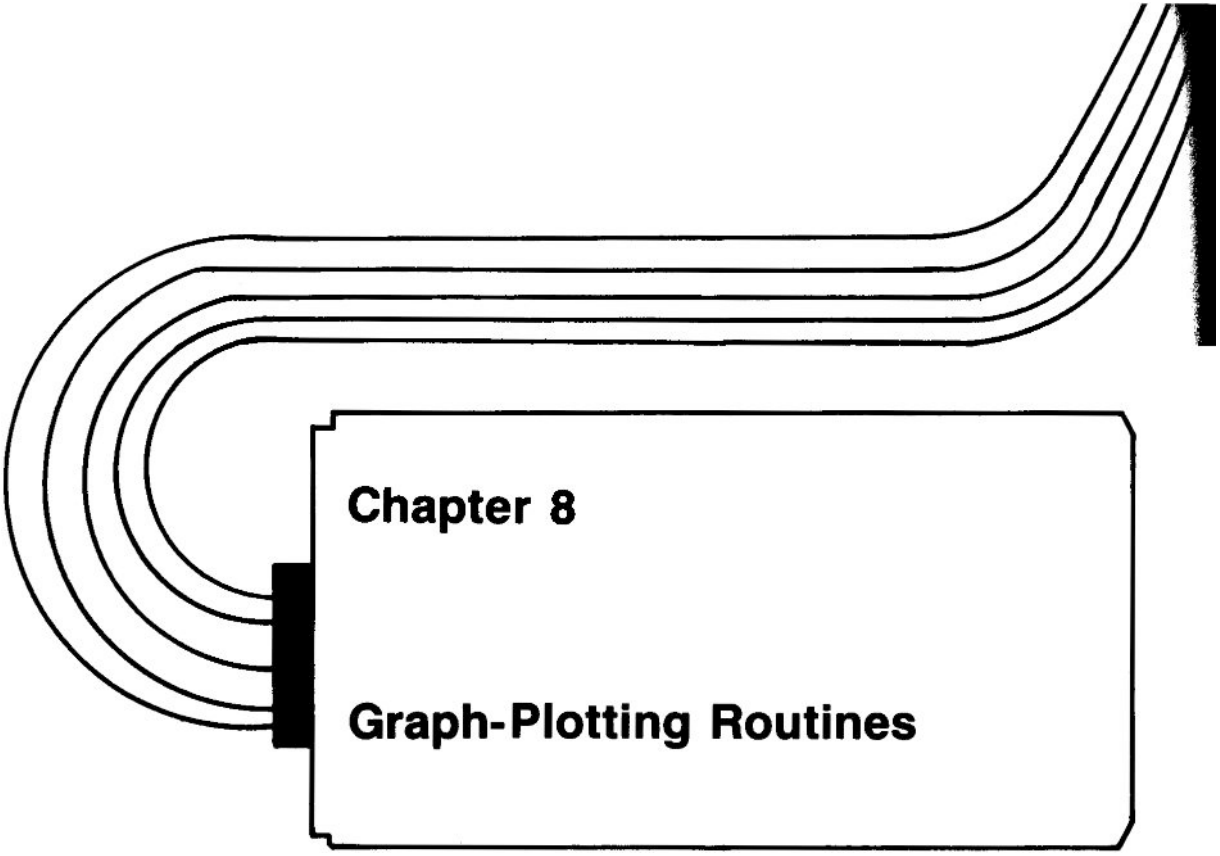


Fig. 7-18. Bottom-side of the AID circuit board.

```
10 REM PROGRAM 7.3
20 PRINTCHR$(147):POKE65265,255
30 PRINT"A/D TEST PROGRAM":PRINT" "
40 INPUT"SELECT CHANNEL TO BE TESTED: 0-15";A
45 B=A*16
50 POKE65265,015+B:POKE65265,011+B:POKE65265,015+B
60 POKE65265,007+B:PRINTPEEK(65264
70 GET A$:IF A$="S" THEN GOTO40
80 GOTO50

READY.
```

Program 7-3. The AID test program.



Chapter 8

Graph-Plotting Routines

A eOMPUTER PLOTTING ROUTINE IS GENERally used to generate some type of graphical display to summarize the way one numerical quantity "depends on" or "varies" with another quantity. The plotted graph generated on the video screen of the computer's monitor can be a video picture of a mathematical function or a technical display of recorded data. The main point of this chapter is to show you how to write a simple plotting program to display a mathematical function or recorded data that has been secured from an analog to digital converter. The ability to display data in graphical form will greatly increase the number of technical applications in which you can use your computer.

LOW-RESOLUTION PLOTTING ROUTINES

The plotting routines in Programs 8-1 for the C-64 and 8-2 for the C-16 and PLUSI4 will be used to present the basic method that can be used to generate a low-resolution video graph. These programs use the low-resolution graphic capabilities of the computers. Program line number 5 is used to clear the screen to plot the graph. The FOR .. NEXT loop that is started in line 10 is used to select the vertical column (the "X" value) that will receive the plot point. Since there are 40 vertical columns in the video format of the computers we are using, the FOR .. NEXT loop counts from 0 to 39 to plot graph points across the entire screen. Line 20 is a mathematical sine wave formula that is used to generate the "Y" value of the plot points. The Y values are the line positions in the vertical columns where the plot points will be placed. The number 12 in front of the "*SIN" is used to control the amplitude of the sine wave and the number 4 in "X/4" is used to control the number of waveform cycles that will be plotted. The values of 12 and 4 must be selected to keep the plotted graph points on the video screen.

Line 30 is the main event line in these programs. The formula in line 30 calculates the spot on the video screen where the plot point is placed.

```
1 REM PGM 8.1 FOR THE C-64
5 PRINTCHR$(147)
10 FOR X=0 TO 39
20 Y=INT(12*SIN(X/4))
30 B=(1504+(-40*Y)+X)
40 POKE B,87
50 NEXT
60 END
```

Program 8-1. A simple plotting routine for the C-64 computer.

```
1 REM PGM 8.2 FOR THE C-16 OR PLUS/4
5 PRINTCHR$(147)
10 FOR X=0 TO 39
20 Y=INT(12*SIN(X/4))
30 B=(3552+(-40*Y)+X)
40 POKE B,87
50 NEXT
60 END
```

Program 8-2. A simple plotting routine for the C-16 and the PLUSI4 computers.

Line 30 contains the formula:

Program 8-1: B = (1504 + (- 40 * Y) + X)

or

Program 8-2 B = (3552 + (- 40*Y) + X)

If you look at the video screen memory layout for the C-64, the C-16, or the PLUSI4 in your computer's guide, you will find that 1504 or 3552 is the memory location that controls what character is placed in the middle of column 0 on the video screen. Because there are 24 lines on the video screen, you can add the value of " - 40*Y" to 1504 or 3550 to calculate the vertical position in column 0 where the plot point will be placed. If you then add the value of "X", you can then place the plot point horizontally on the screen. Programs 8-3 for the C-64 and 8-4 for the C-16 and PLUSI4 demonstrates this formula.

Line 40 is the POKE instruction that does the work of putting the plot point in its calculated location. You must make sure that the calculated POKE address does not fall outside the video screen memory, because if it does, you can crash your program by POKING data into the wrong memory location. Line 50 loops the program back for the next math calculation and plot operation.

Low-resolution plotting has application limitations but it can still be used to display elementary graphs as one can observe by using Program 3-16 of Chapter 3. One can also learn the basic screen plotting fundamentals from this method.

HIGH-RESOLUTION PLOTTING

High-resolution plotting is required if one is going to attempt any serious graphing for science or engineering applications. The high-resolution graphing capabilities of the Commodore computers are as good or better than most other personal computers. You can generate a very sophisticated graphing program using the high-resolution graphics of these computers. Because this book is really about hardware projects and not software applications, all of the high-resolution graphic pro-

```
1 REM  PLOTTING DEMSTRATION PROGRAM 8.3
2 REM  FOR THE C-64
5 PRINTCHR$(147)
10 INPUT"INPUT THE ROW OF THE PLOT POINT - MUST  BE BETWEEN -12 AND +12 -";R
20 INPUT"INPUT THE COLUMN OF THE PLOT POINT -    MUST BE BETWEEN 0 AND 39 -";C
25 PRINTCHR$(147)
30 B=(1504+(-40*R)+C)
40 POKE B,87
50 PRINT"■":PRINTB,R,C
60 END
```

Program 8-3. This C-64 program shows how a point can be plotted on the video screen for graphing displays.

```
1 REM  PLOTTING DEMSTRATION PROGRAM 8.4
2 REM  FOR THE C-16 OR THE PLUS/4
5 PRINTCHR$(147)
10 INPUT"INPUT THE ROW OF THE PLOT POINT - MUST  BE BETWEEN -12 AND +12 -";R
20 INPUT"INPUT THE COLUMN OF THE PLOT POINT -    MUST BE BETWEEN 0 AND 39 -";C
25 PRINTCHR$(147)
30 B=(3552+(-40*R)+C)
40 POKE B,87
50 PRINT"■":PRINTB,R,C
60 END
```

Program 8-4. This C-16 and PLUSI4 program shows how a point can be plotted on the video screen for graphing displays.

grams will be written using Simon's Basic or the high-resolution graphics commands of the C-16 or PLUSI4 computers. Using the advanced graphics commands will make the job of writing the graphing program easier, but the actual time it takes to generate and display a graph with these BASIC commands can be quite long when compared to a machine-language graphing routine.

Programs 8-5 and 8-6 are presented as examples of high-resolution graphing programs.

```
1 REM PROGRAM 8.5
2 REM A PLOTTING PROGRAM FOR THE C-64 USING SIMON'S BASIC
5 PRINTCHR$(147):DIM AA(256)
95 PRINT" NOW COMPUTING DATA POINTS"
100 FOR AM=0 TO 251
110 AA(AM)=INT(100*SIN(AM/10))
120 NEXT AM
130 HIRES1,6:GOTO 2000
140 FOR II=0TO250:B=(AA(II)*.58):C=INT(104-B)
145 A=II+40
150 PLOT A,C,1
155 NEXTII
160 GET A$ : IF A$="C" THEN 170
165 GOTO160
170 COPY
180 GOTO 180
2000 TEXT 60,20,"         A SINE WAVE ",1,1,8
2020 LINE 40,45, 40,165,1
2025 LINE 40,165,291,165,1
2030 FOR I=45 TO 165 STEP 6
2035 LINE 35,I,40,I,1
2040 NEXTI
2050 FOR I=45 TO 165 STEP60
2055 LINE 30,I,35,I,1:NEXTI
2060 FOR I= 40 TO 296 STEP 10
2065 LINE I,165,I,170,1:NEXT I
2070 FOR I=40 TO 290 STEP50
2080 LINE I,165,I,175,1:NEXTI
2090 CHAR 20,102,48,1,1
2100 GOTO140
```

Program 8-5. A high-resolution graphing program for the C-64 using SIMON's BASIC.

```
1 REM PROGRAM 8.6
2 REM A PLOTTING PROGRAM FOR THE C-16 OR THE PLUS/4
5 PRINTCHR$(147):DIM AA(210)
95 PRINT" NOW COMPUTING DATA POINTS"
100 FOR AM=0 TO 200
110 AA(AM)=INT(075*SIN(AM/10))
120 NEXT AM
125 COLOR 2,2
130 GRAPHIC 2,1: GOTO200
140 FOR II=0TO200:B=(AA(II)*.58):C=INT(090-B)
145 A=II+44
150 DRAW1,A,C
155 NEXTII
170 GETA$:IF A$="S" THEN GOTO198
180 GOTO 170
198 GRAPHIC 0
199 END
200 DRAW 1,44,146 TO 244,146
205 DRAW 1,44,36 TO 44,146
210 FOR I=44 TO 244 STEP 20
215 DRAW 1,I,146 TO I,150
220 NEXT I
230 FOR I=36 TO 146 STEP 11
235 DRAW 1,39,I TO 44,I
240 NEXT I
250 CHAR 1,5,19,"0    2    4    6    8    10"
260 CHAR 1,3,11,"0-"
270 CHAR 1,3,4,"+"
280 CHAR 1,3,18,"-"
290 CHAR 1,9,2," A SINE WAVE"
300 GOTO140
```

Program 8-6. A high-resolution graphing program for the C-16 and the PLUSI4 using the computers' built-in high-resolution graphics commands.

These two programs are used to generate sine waves from a mathematical formula, but the general program plotting routine can be used to plot many other forms of data. In Chapter 9, the waveform recording programs will use plotting routines that are similar to Programs 8-5 and 8-6 to display the recorded waveform.

Programs 8-5 and 8-6 both perform similar functions but use different high-resolution graphic commands as required by the host computer. Lines 100, 110, and 120 in each program is used to calculate and store the sine wave data values in an array so the data can be recalled and plotted later. The ability to store your data in an array will give you the capability to secure the data, store it, and then plot it at a later time. Once you have the data stored, you can use that data for other things than just graphs.

Lines 140 to 155 in each program is used to calculate the position of the plot point. You can experiment with each of the values in these three lines and observe the effect that they have on the plotted graph. It will be easy to change the plotting program to meet your graph plotting needs after you learn the function of each line. The rest of each program (line numbers 200 and up) is used to generate the X and Y axis display. A little experimentation with these lines will show you how the X and Y axis can be modified to generate different graphing displays.

SUMMARY

The plotting routines that have been presented in this chapter will show you the general method that is used in this book to present graphical data. You can change the mathematical formula in each of the programs to display other math functions. When you are trying other math functions, remember to watch where you are POKING data. If your formula calculates a POKE address that is outside of the video screen's memory area, you can poke data into the wrong memory location and crash the computer program.

Chapter 9

A Practical Waveform Recording Program

IN THIS eHAPTER, AN ANALOG WAVEFORM recording program will be presented that will show the practical application capability of the circuits that have been presented in the previous chapters. This waveform recording program can be used for a variety of practical applications in physics, chemistry, and engineering. The presented program will give you the capability to record an electrical analog waveform and display the recorded waveform data on the computer's video monitor or printer. The complete waveform recording program uses the combination of a BASIC control routine and a high-resolution graphics display routine along with a machine-language subroutine that operates the analog-to-digital converter circuit.

The basic block diagram of the waveform recorder which is presented in this chapter is shown in Fig. 9-1. The waveform recorder is really a Commodore 64 computer system plus two circuits that were presented in previous chapters. A similar system can be made using a VIC-20. a C-16, or a PLUSI4 computer. Even if you connect all of the computer devices together, you will not have a waveform recorder until you load a waveform recording program into the computer. The waveform recorder is only as good as its control program.

As one can see by observing Fig. 9-1, the complete waveform recording system is assembled around the C-64. The universal op-amp circuit was presented in Project 3-3 of Chapter 3 and the analog-to-digital converter circuit is from Chapter 6. The high-resolution graphics portion of the waveform display uses SIMON'S BASIC commands, so you will need SIMON'S BASIC plug-in cartridge to use the presented program. The video monitor can be any that will work with your computer. The printer must be either a MPS801 or a VIC 1525 because these two printers will work with the SIMON'S BASIC high-resolution graphics.

The complete waveform recording program requires two programs; a BASIC program and a machine-language subroutine. Program 9-1 is the BASIC control program. This program gives you the options of selecting the time interval between the recorded waveform data points, starting the

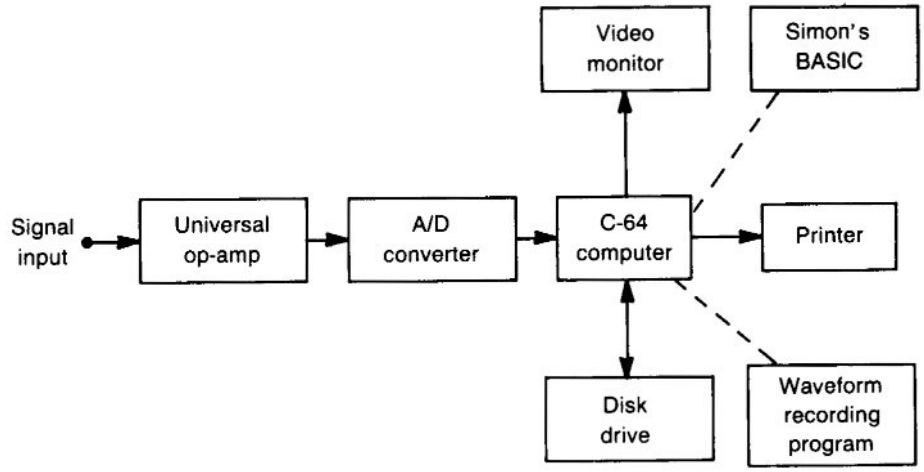


Fig. 9-1. Block diagram of a waveform recording system.

```
1 REM PGM 9.1 - USE WITH SIMON'S BASIC
2 DIM AA(256)
5 PRINTCHR$(147)
10 PRINT"  A WAVEFORM RECORDING PROGRAM"
15 PRINT" "
20 PRINT"ENTER THE TIME/DATA POINT INTERVAL. "
21 PRINT" "
22 PRINT" A =  64 MICROSECONDS"
23 PRINT" B = 100 MICROSECONDS"
24 PRINT" C = 500 MICROSECONDS"
25 PRINT" D =   1 MILLISECOND"
26 PRINT" E =  10 MILLISECONDS"
27 PRINT" F = 100 MILLISECONDS"
28 PRINT" G = 1/4 SECOND"
HROUGH H";:
31 GET A$
32 IF A$="A" THEN GOTO 42
33 IF A$="B" THEN GOTO 43
34 IF A$="C" THEN GOTO 44
35 IF A$="D" THEN GOTO 45
36 IF A$="E" THEN GOTO 46
37 IF A$="F" THEN GOTO 47
38 IF A$="G" THEN GOTO 48
39 IF A$="H" THEN GOTO 49
40 GOTO31
42 POKE 52250,1:POKE52254,4:GX$="64E-06 SECONDS":GOTO51
43 POKE 52250,1:POKE52254,6:GX$="100E-06 SECONDS":GOTO51
44 POKE 52250,1:POKE52254,31:GX$="500E-06 SECONDS": GOTO51
45 POKE 52250,1:POKE52254,62:GX$="1E-03 SECONDS": GOTO51
46 POKE 52250,10:POKE52254,115:GX$="10E-03 SECONDS":GOTO51
47 POKE 52250,100:POKE52254,124:GX$="100E-03 SECONDS":GOTO51
48 POKE 52250,150:POKE52254,207:GX$="1/4 SECNDS":GOTO51
```

Program 9-1. The BASIC waveform recording control program.

```
49 PRINT" ": PRINT" ENTER TA,TB -";:INPUT TA,TB
50 POKE 52250 ,TB:POKE 52254,TA: GX$=" TA-TB SECONDS"
51 PRINT" "
70 PRINT" ":PRINT" PUSH 'S' TO START RECORDING"
75 PRINT" ":PRINT" WHEN THE WAVEFORM DISPLAY IS COMPLETE,";:
76 PRINT"  PRESS <C> FOR A HARD COPY PRINT.":PRINT" "
80 GET A$:IF A$="S" THEN GOTO83
81 GOTO80
83 PRINT" ":PRINT" RECORDING":PRINT" "
90 SYS 52224
95 PRINT" ":PRINT"FINISHED":PRINT" "
100 PRINT" DO YOU WANT A DATA PRINT-OUT - Y/N"
101 PRINT" ":GG=0
102 GETA$:IF A$="N" THEN GOTO 120
103 IF A$="Y" THEN GOTO 105
104 GOTO102
105 PRINT"   PRINTER ON  Y/N"
106 GETA$:IF A$="N"THEN GOTO 110
107 IF A$="Y" THEN GOTO 109
108 GOTO 106
109 GG=1:OPEN4,4:CMD4
110 I=0
111 PRINTI;:PRINTINT((PEEK(52480+I)*.01953125)*100)/100;
112 I=I+1:IF I)255 THEN GOTO 116
113 PRINTTAB(19);I;:PRINT" ";:PRINTINT((PEEK(52480+I)*.01953125)*100)/100:I=I+1
114 IF I<255 THEN GOTO111
116 IF GG=0 THEN GOTO120
118 PRINT#4:CLOSE4
120 PRINT" ":PRINT" PRESS ANY KEY TO CONTINUE"
121 GET A$:IF A$="" THEN GOTO 121
122 PRINT" ":PRINT" PLEASE WAIT":PRINT" "
124 FOR AM=0 TO 256
125 AA(AM)=PEEK(52480+AM):NEXTAM
130 HIRES1,6:GOTO 2000
140 FOR II=0TO254
145 A=II+40 :B=(AA(II) *.58):C=INT(165-B)
150 PLOT A,C,1
155 NEXTII
160 GETA$:IF A$="C" THEN GOTO180
170 GOTO 160
180 COPY
190 GOTO190
1000 GOTO110
2000 :
2020 LINE 40,15, 40,165,1
2025 LINE 40,165,291,165,1
2030 FOR I=15 TO 165 STEP 6
2035 LINE 35,I,40,I,1
2040 NEXTI
2050 FOR I=15 TO 165 STEP30
2055 LINE 30,I,35,I,1:NEXTI
2060 FOR I= 40 TO 296 STEP 10
2065 LINE I,165,I,170,1:NEXT I
2070 FOR I=40 TO 290 STEP50
2080 LINE I,165,I,175,1:NEXTI
2090 CHAR 20,11,53,1,1
2100 CHAR 20,41,52,1,1
2110 CHAR 20,71,51,1,1
```

```
2115 CHAR 20,101,50,1,1
2120 CHAR 20,131,49,1,1
2125 CHAR 20,161,48,1,1
2130 CHAR 37,178,48,1,1
2135 CHAR 81,178,53,1,1
2140 CHAR 90,178,48,1,1
2145 CHAR 127,178,49,1,1
2150 CHAR 136,178,48,1,1
2155 CHAR 145,178,48,1,1
2160 CHAR 177,178,49,1,1
2165 CHAR 186,178,53,1,1
2170 CHAR 195,178,48,1,1
2175 CHAR 227,178,50,1,1
2180 CHAR 236,178,48,1,1
2185 CHAR 245,178,48,1,1
2190 CHAR 277,178,50,1,1
2195 CHAR 286,178,53,1,1
2200 CHAR 295,178,48,1,1
2210 TEXT 60,190,"   RECORDED DATA POINTS",1,1,8
2215 CHAR 5,40,09,1,1
2220 CHAR 5,50,14,1,1
2225 CHAR 5,60,16,1,1
2230 CHAR 5,70,21,1,1
2235 CHAR 5,80,20,1,1
2240 CHAR 5,100,22,1,1
2245 CHAR 5,110,15,1,1
2250 CHAR 5,120,12,1,1
2255 CHAR 5,130,20,1,1
2260 CHAR 5,140,19,1,1
2265 TEXT 30,05,"TIME/DATA POINT =",1,1,8
2270 TEXT 180,05,GX$,1,1,8
2400 GOTO 140
```

waveform recording when you press the "S" key, and a numerical data point display if needed. After the optional part of the program is finished, the program displays the recorded waveform on the video screen or printer. Program 9-2 is the machine-language program that controls the operation of the analog-to-digital converter circuit and stores the recorded data point information in a RAM memory location for later use by the BASIC program.

Looking at Program 9-1, shows you that lines 20 to 50 are used to select the time interval between the recorded data points. Lines 51 to 95 are used to start the waveform recording machine-language subroutine. Lines 100 to 120 are used to generate the optional data point display. Lines 120 to 190 are used for the SIMON'S BASIC high-resolution plotting routine, and lines 2000 to 2400 are used to draw the display graph on the video screen.

Program 9-2 is the *AID* machine-language control subroutine. Address lines CC00 to CC05 are used to turn off the keyboard interrupts. Line CC08 sets the X register to zero. Lines CC0A and CC0C are used to set memory location $CCFF to $FF. Lines CCOF and CC12 form a remote control start routine by checking BIT 7 of the USER PORT to see if it is a logic "1" or "0". If BIT 7 is a logic ZERO the program will wait in a loop until BIT 7 becomes a logic ONE. This BIT test routine only functions if you have a control line connected to USER PORT BIT 7. If nothing is connected to the USER PORT, the remote control routine will not effect the program operation. Lines CC14 and CC16 are used to start the *AID* conversion cycle. Lines CC19 to CC27 are the time-delay loop that receives its time delay information from the BASIC program. After the time-delay loop is finished, lines CC29 and CC2C reads and stores the *AID* converter's digital data in RAM memory. The RAM

```
PGM 9.2 FOR THE C-64


,CC00 A9 00    LDA #$00
,CC02 8D 0E DC STA $DC0E
,CC05 8D 0D DC STA $DC0D
,CC08 A2 00    LDX #$00
,CC0A A9 FF    LDA #$FF
,CC0C 8D FF CC STA $CCFF
,CC0F 2C 01 DD BIT $DD01
,CC12 10 FB    BPL $CC0F
,CC14 A9 00    LDA #$00
,CC16 8D F0 DF STA $DFF0
,CC19 A9 FA    LDA #$FA
,CC1B 85 FB    STA $FB
,CC1D A9 0B    LDA #$0B
,CC1F 85 FC    STA $FC
,CC21 C6 FC    DEC $FC
,CC23 D0 FC    BNE $CC21
,CC25 C6 FB    DEC $FB
,CC27 D0 F4    BNE $CC1D
,CC29 AD F0 DF LDA $DFF0
,CC2C 9D 00 CD STA $CD00,X
,CC2F E8       INX
,CC30 CE FF CC DEC $CCFF
,CC33 D0 DF    BNE $CC14
,CC35 A9 01    LDA #$01
,CC37 8D 0E DC STA $DC0E
,CC3A 60       RTS
,CC3B 00       BRK
,CC3C 00       BRK
,CC3D 00       BRK
,CC3E 00       BRK
```

Program 9-2. The machine-language subroutine for Program 9-1 that controls the AID converter operation.

memory location is calculated by adding the current value of the X register to $CD00, which stores all 256 data conversions between memory locations $CD00 and $CDFF. Line CC2F increments the X register. Lines CC30 and CC33 decrements memory location $CCFF and checks to see if it was decremented to zero. If it was zero, the program goes on to line CC35. If $CCFF was not zero, the program loops back for another *AID* conversion cycle until 256 conversion cycles have been completed. Lines CC35 and CC37 are used to turn the keyboard interrupt back on, and line CC3A is the return from subroutine instruction. This paragraph is a short explanation of the machine-language subroutine's operation. If you look up and read about all of the instructions in Chapter 13, you can easily understand the complete operation of this *AID* control subroutine.

When using the waveform recorder, you should be aware of the term aliasing, because aliasing can be encountered anytime you are using sampled data. Aliasing is a phenomenon that can cause high-frequency components to appear as low-frequency components in the sampled waveform. Figure 9-2 shows a graphical representation of aliasing where a waveform record might be used to record a 1000 Hz sine wave using data points that are spaced .001 seconds apart. The 1000 Hz sine wave would be sampled at the same point in its waveform each time, which would make the recorded amplitude-time visual display look like a straight-line de signal. This display would be a false waveform representation. You should not have any aliasing problems with this waveform recorder if you limit your input frequencies to lower than 750 Hz and make several waveform recordings at different sample rates. If aliasing does present a problem, you may have to precede the universal op-amp circuit with a low-pass filter circuit to limit the frequencies that could cause aliasing. You will find that the universal op-amp as presented has a frequency response that starts to roll-off at about 750 Hz.

You can check the waveform recording program and system by connecting a potentiometer to the input of the *AID* converter as shown in the previous chapters about *AID* circuits.

A PRACTICAL APPLICATION

The practical application that was chosen for this chapter is a pendulum. There are many different experiments that can be performed with a pendulum such as calculating the acceleration due to gravity and study of oscillations. The pendulum that was constructed for our project is shown in Figs. 9-3 and 9-4. It is constructed out of surplus G-10 copper-clad circuit board material by cutting out the pieces and soldering them together. The actual size of the pendulum is not a critical factor. The most important part of the pendulum is to find a l00-ohm multiturn potentiometer that turns very

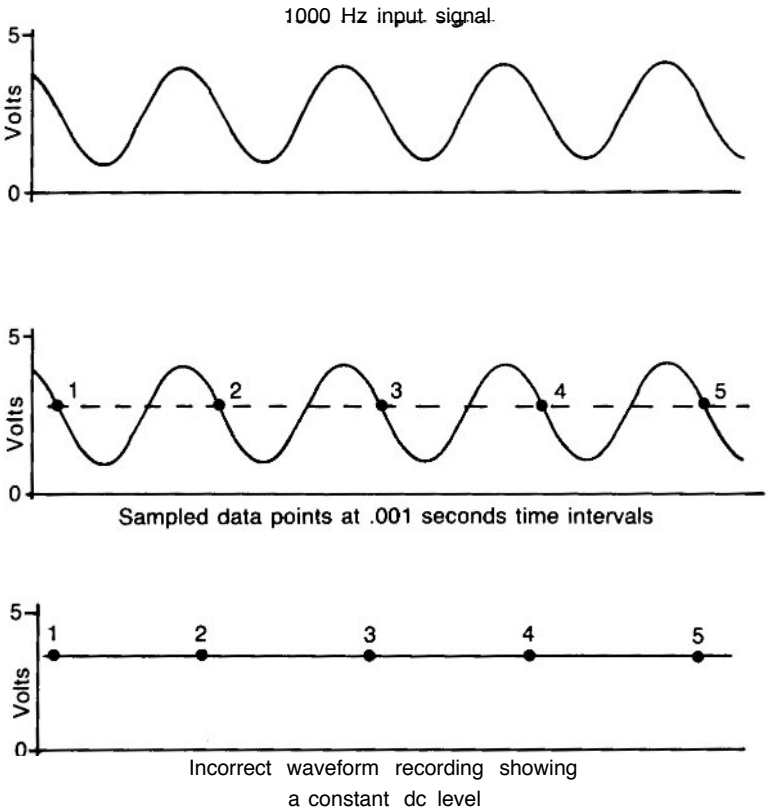


Fig. 9-2. Graphical presentation of aliasing (see text).

easy. The shaft of the multiturn (ten-turn) potentiometer is used as a support for the swinging pendulum arm, so a potentiometer that turns easy is needed. The pendulum arm is a long thin piece of G-10 board that is connected to the support shaft on one end and contains the pendulum weight on the other.

The shaft that supports the pendulum arm is made of brass tubing that was purchased from a local model airplane hobby shop. You must secure a brass tube that will just fit over the shaft of the potentiometer. (The next size over *1/4* inch) The other side of the support shaft is supported by a nylon bushing that can also be purchased at the same hobby shop. You may need to buy the next smaller size of brass tubing also so you can telescope the tubes together to get the tubing diameter back down to 114 inch so it will fit into a standard nylon bushing. The swing pendulum weight should be steel or lead which can be purchased at a fishing tackle supply store. When the pendulum frame is completed, it should be screwed to a flat wooden base board. Make sure that the complete pendulum framework assembly is mechanically stable while the pendulum arm is swinging.

The pendulum arm support shaft tubing can be connected to the potentiometer shaft with five minute epoxy or super glue after the pendulum is secured to its wooden base board. When you are

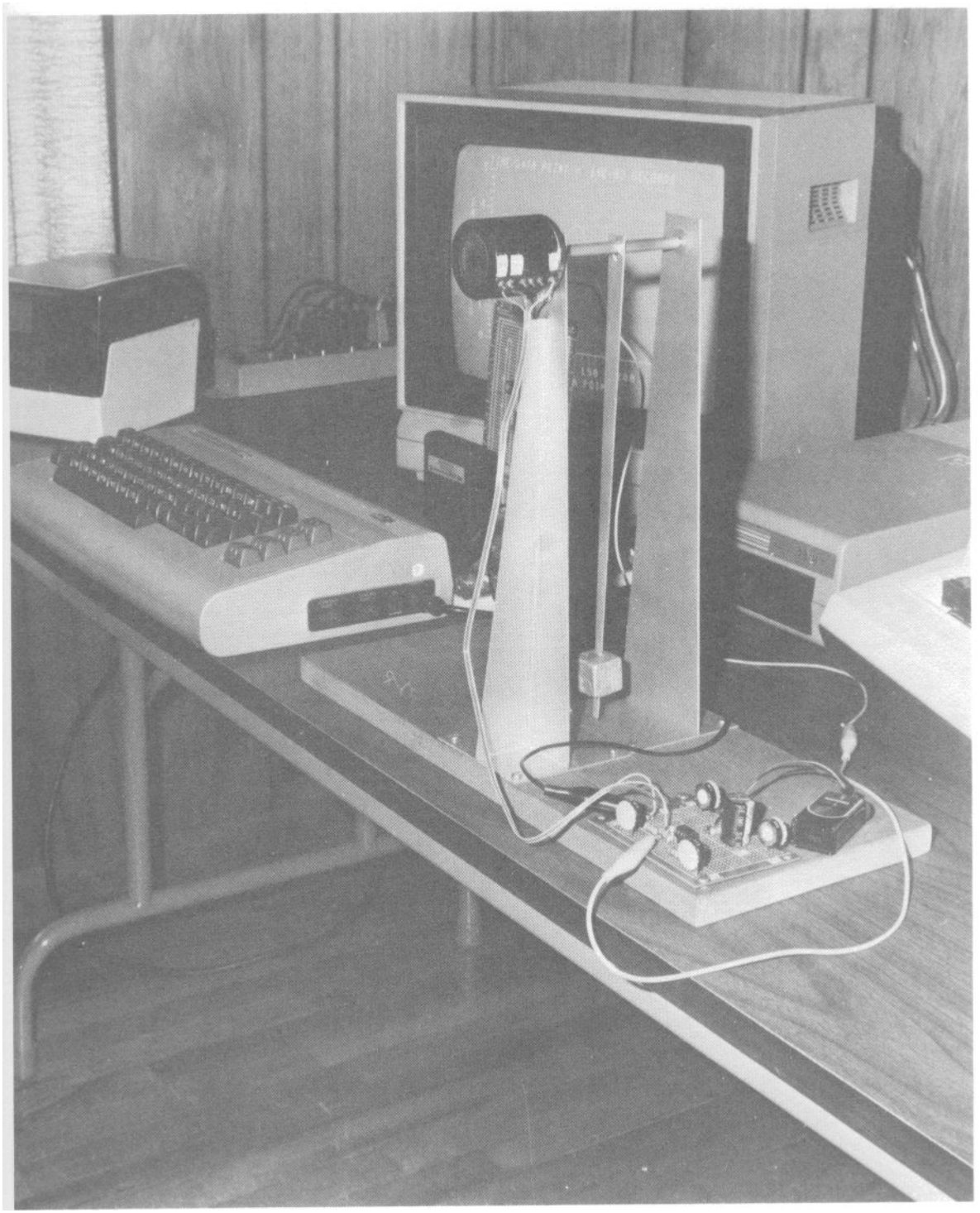


Fig. 9-3. Pictorial view of the pendulum.

Fig. 9-4. The pendulum and the waveform recorder in action.

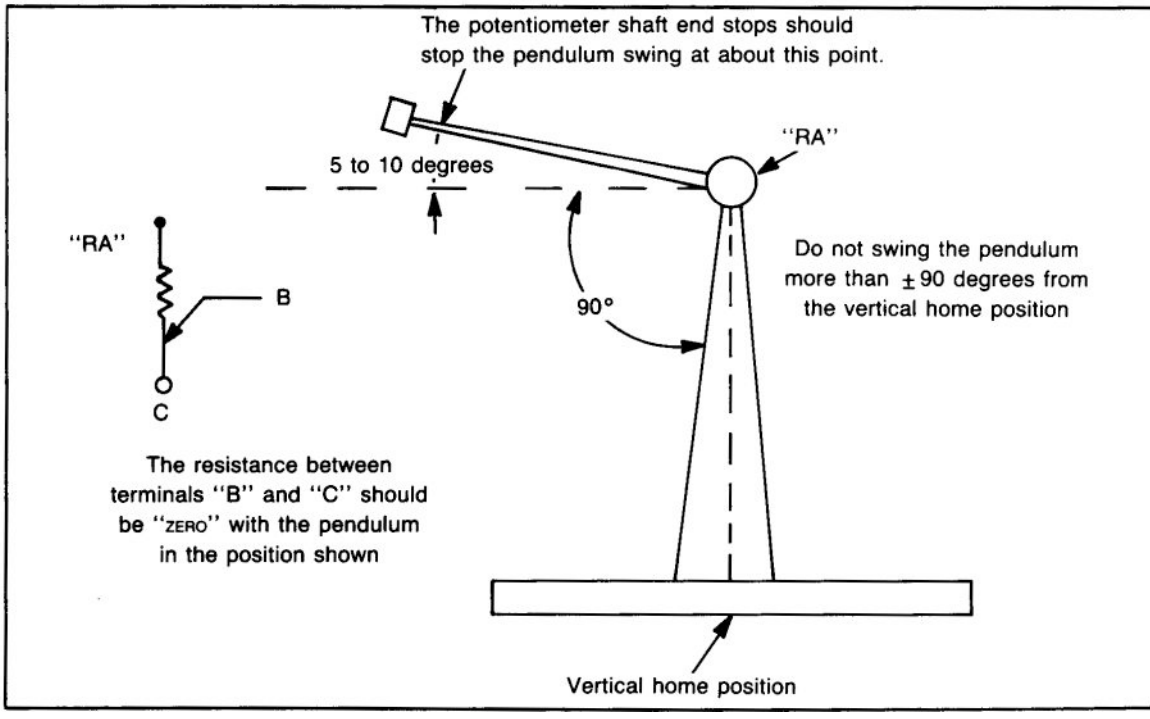


Fig. 9-5. The positional data for the pendulum arm and potentiometer "RA" connection.

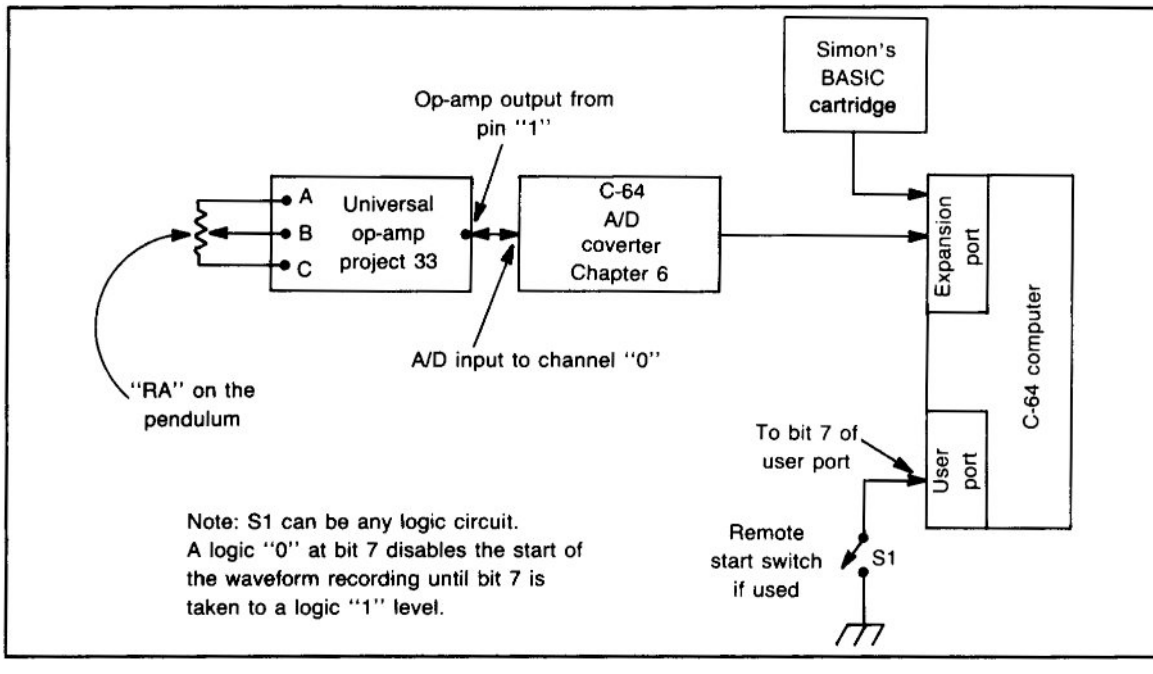


Fig. 9-6. Block diagram of the waveform recording system using the AID converter from Chapter 6 and the universal op-amp from Chapter 3.

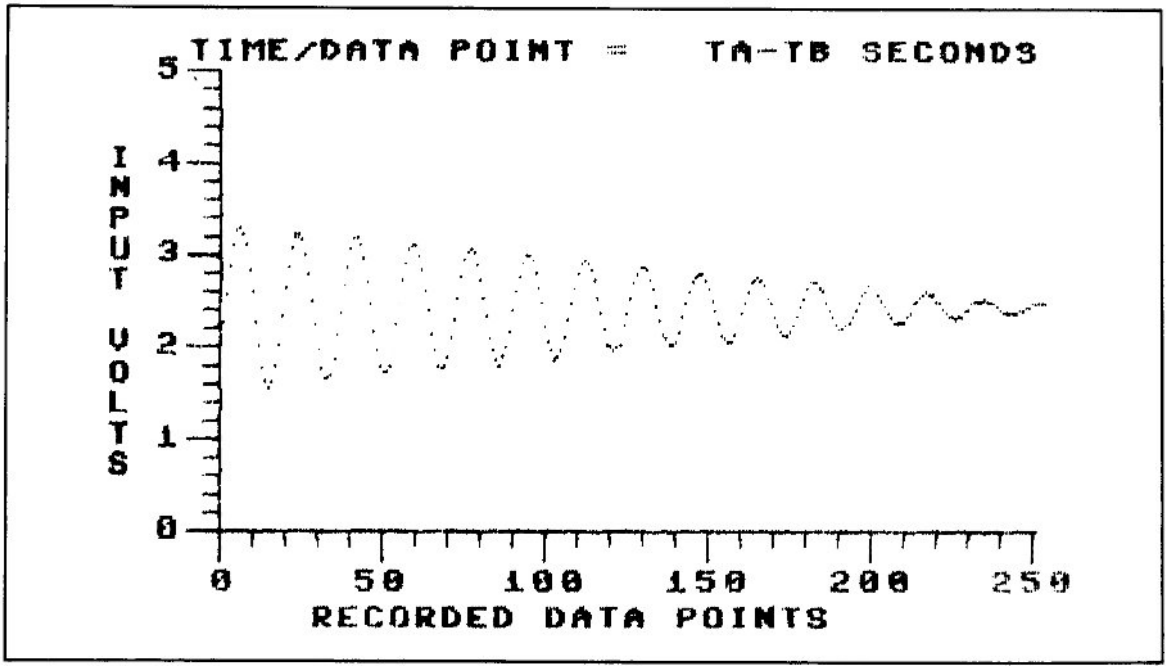


Fig. 9-7. Printout from the waveform recorder showing the decaying sine waves from the pendulum oscillation.

gluing the shafts together make sure that the potentiometer shaft is adjusted all the way to one end, and set the pendulum arm so the potentiometer end point and the pendulum arm is at 90 degrees to the vertical arm rest position as shown in Fig. 9-5. This will give you a full 180 degree pendulum swing.

A full180 degree pendulum swing will develop a potentiometer voltage output from zero to about 1/4 volt if you use a ten turn pot. Since the *AID* converter we are using requires a zero to 5 volts input range, the voltage range of zero to 114volt will have to be amplified in order to secure a zero to 5 volts input for the AID converter. You can use the universalop-amp circuit of Project 3-3 to amplify the 1/4 volt level to a full five volts for the *AID* converter input. A completed connection diagram of the waveform recorder is presented in Fig. 9-6.

Conclusion

When you have the pendulum completed and connected to the waveform recorder system, you should be able to pull the pendulum arm back and let it swing through a twenty degree arc and record a decaying sine wave as shown in Fig. 9-7. The constant decay of the sine wave shows that the potentiometer shaft turning resistance is constant and linear. Chapter 10 presents four computer programs that can be used to analyze the recorded waveform data.

Chapter 10

Elementary Signal Analysis

THE PROBLEMS OF WAVEFORM ANALYSIS were for many years an academic area of science and engineering that was left for the mathematically elite engineer or scientist to solve. Generally, waveform analysis requires the understanding and the use of rigorous mathematics, but if you use a computer and a signal analysis program, you can easily do elementary signal analysis without high-level math. This chapter will present four computer programs that will enable you to perform elementary signal analysis on waveforms that you have recorded with the waveform recording system that was presented in Chapter 9. A bibliography will be presented at the end of this chapter to help you in further research efforts.

Actually, most of the waveforms that you will want to analyze will be physical waveforms and not electrical waveforms, but the basic physical parameters can be converted into electrical waveforms by using transducers. Transducers can easily convert physical parameters such as light level, sound, pressure, acceleration, temperature, and weight into electrical waveforms that can be recorded on a waveform recording system so you can completely analyze the dynamic waveforms. The analysis of a dynamic waveform can lead to a better understanding of the physical system that generated the waveform.

The waveform analysis that will be presented in this chapter will take complex waveform that has been recorded in the *time domain* and transform that waveform into the *frequency domain*. This means that a waveform that has been recorded using amplitude-time data points will be transformed into the various signal frequencies that can be added together to generate the physical waveform. A transformation of time-amplitude waveform data into amplitude-frequency data is called *Fourier transformation* (see references). A pictorial that will help you visualize Fourier transformation is presented in Fig. 10-1.

Now, you will be presented with three computer programs that will let you perform Fourier transformations on waveform data that you collect with the waveform recording system of Chapter 9. The three programs are Fourier Series Program

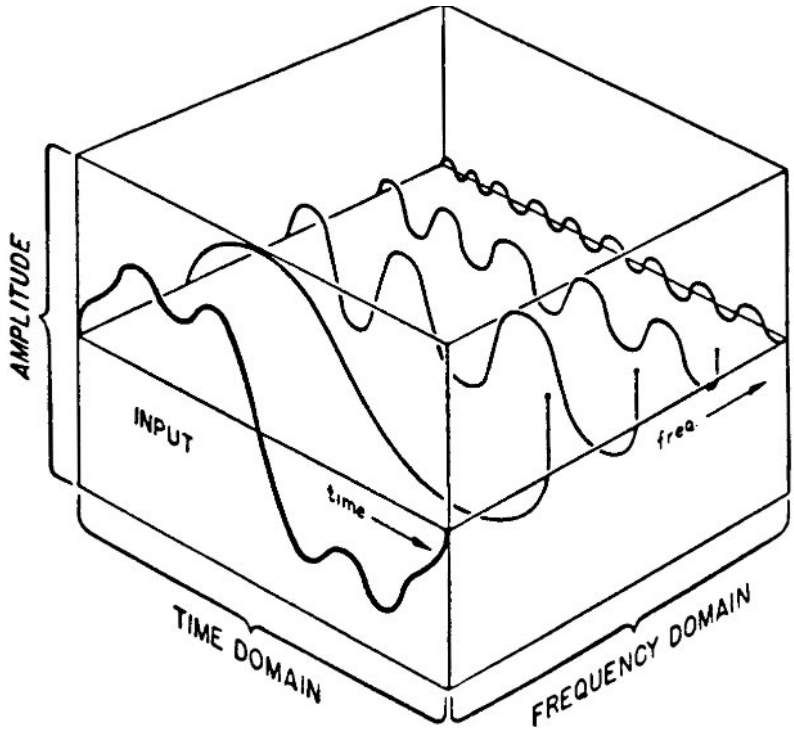


Fig. 10-1. This is a graphical representation of the Fourier series where sine waves can be added together to give a complex waveform time history. Fourier first published the series in 1807. It states that any periodic waveform in the Time domain can be expressed as the sum of a series of cosines and sines of different amplitudes and frequencies. From this, Fourier developed the Fourier Transform also based on the concept of cosine and sine waves 2 (courtesy of Hewlett Packard Company).

10-1. Discrete Fourier Transform Program 10-2,2 and Fast Fourier Transform Program 10-3.3 All three of these programs are use to transform amplitude-time waveform data into amplitude-frequency data. This book will not go into a technical discussion of Fourier transform theory because any college or university library will have many books on the subject, and you can use these three programs to help you understand any of the books.

If you look up a book on Fourier theory, it will tell you that the frequency spectrum of a square wave contains its fundamental frequency and all of its odd harmonics. A square wave is really a complex composition of a fundamental frequency sine wave and a large number of odd-frequency harmonically related sine waves. The higher the harmonic content of the square wave, the sharper the square wave corners will be. Generally a good sharp 1 MHz square wave will contain detectable odd harmonics to over 100 MHz. This fact can be demonstrated by entering data into the three Fourier transform programs for a square wave. This is shown in Figs. 10-2, 10-3, and 10-4. Actually the Fourier series program is to be used for continuous repeated waveforms and the two Fourier transforms programs are to be used for transient type waveforms. Once you have looked up read some of the reference material, you will easily understand the discussion in this paragraph.

```
1 REM - PGM 10.1  FS
10 REM - FOURIER SERIES PGM BY HOWARD M BERLIN, FROM ' CIRCUIT DESIGN PROGRAMS
12 REM - FOR THE TRS-80 ' PUBLISHED BY H.W. SAMS, INDIANAPOLIS, IND 1980
14 REM - CONVERTED FOR THE C-64 BY R. LUETZOW
50 PRINTCHR$(147)
100 DIM F(100),C(100),D(100),A2(100)
500 PRINT"A FOURIER SERIES PROGRAM"
1040 Z9=1
1060 INPUT"ENTER THE TOTAL NUMBER OF DATA PTS.      (MUST BE AN ODD NUMBER)";N
1080 PRINT" "
1081 PRINT" NOTE - THE NUMBER OF HARMONICS MUST BE  2 LESS THAN 1/2 OF THE";
1082 PRINT" TOTAL NUMBER OF  DATA POINTS ENTERED":PRINT" "
1085 INPUT"NO. OF HARMONICS";N8
1090 IF N8>(N/2)-2 THEN GOTO 1080
1100 IF Z9=1 THEN GOTO 1110
1102 GOTO1150
1110 FOR I=1 TO N
1120 PRINT"POINT";I;
1130 INPUT F(I)
1140 NEXT I
1149 PRINT" "
1150 PRINT" "
1151 INPUT"SIGNAL PERIOD";T
1160 N3=N-1
1170 A=2*3.1415927/T
1180 T3=T/N3
1190 A0=0
1200 FOR I=1 TO N3 STEP 2
1210 A0=A0+F(I)+4*F(I+1)+F(I+2)
1220 NEXT I
1230 A0=A0*T3/(3*T)
1240 IF ABS(A0)<.001 THEN A0=0
1250 FOR X=1 TO N8
1260 PRINTCHR$(147):PRINT"CALCULATING"
1270 PRINT"HARMONIC # ";X
1280 C=0:D=0:E=0:A2=0
1290 T1=-T3
1300 FOR J=1TO N3 STEP 2
1310 T1=T1+2*T3
1320 CZ=(F(J)*COS(X*A*(T1-T3)))+(4*F(J+1)*COS(X*A*T1))
1322 CX=(F(J+2)*COS(X*A*(T1+T3)))
1324 C=C+CZ+CX
1330 DZ=(F(J)*SIN(X*A*(T1-T3)))+(4*F(J+1)*SIN(X*A*T1))
1332 DX=(F(J+2)*SIN(X*A*(T1+T3)))
1334 D=D+DZ+DX
1336 NEXT J
1350 C=2*C*T3/(3*T):D=2*D*T3/(3*T)
1360 IF ABS(C)<.005 THEN C=0
1370 IF ABS (D) <.005 THEN D=0
1380 E= SQR(C*C+D*D)
1390 IF C>0 THEN GOTO1470
1392 GOTO 1400
1400 IF C<0 AND D<0 THEN GOTO 1480
1402 GOTO 1410
1410 IF C=0 AND D>0 THEN GOTO 1460
1412 GOTO 1420
```

Program 10-1. This is a Fourier Series program (see reference 3).

```
1420 IF C=0 AND D=0 THEN GOTO 1430
1422 GOTO 1440
1430 A2=0:GOTO1500
1440 IF C=0 AND D<0 THEN GOTO 1450
1442 GOTO 1490
1450 A2=-90:GOTO1500
1460 A2=90:GOTO1500
1470 A2=(ATN(D/C))*180/3.14159:GOTO1500
1480 A2=-180+(ATN(D/C))*180/3.14159:GOTO1500
1490 A2=180+(ATN(D/C))*180/3.14159
1500 C(X)=C:D(X)=D:E(X)=E:A2(X)=A2
1510 NEXT X
1520 PRINTCHR$(147):PRINT"DC TERM =";:PRINT INT(A0*100)/100
1529 PRINT"HARMONIC"
1530 PRINT" #  COS     SIN        MAG        PHASE"
1531 PRINT" "
1536 FOR I=1 TO N8
1541 PRINTI;
1542 Q1=INT(C(I)*100)/100:PRINTQ1,
1543 Q2=INT(D(I)*100)/100:PRINTQ2,
1544 Q3=INT(E(I)*100)/100:PRINTQ3,
1546 Q4=INT(A2(I)*100)/100:PRINTQ4
1559 NEXT I
1560 F=1/T
1565 K=INT(F*1000)/1000
1570 PRINT"FREQ.=";K;"HZ"
1580 END
2000 Y=1:FORI=0TO1STEP.025
2001 F(Y)=162*COS(6.283185*I)
2020 Y=Y+1:NEXT I:N=Y-1:GOTO1080
```

```
1 REM  PROGRAM 10.2   DFT PGM
2 REM - DFT PROGRAM BY H M BERLIN, FROM ' CIRCUIT DESIGN PROGRAMS
3 REM - FOR THE TRS-80 ' PUBLISHED BY H.W. SAMS, INDIANAPOLIS, IND 1980
4 REM - CONVERTED FOR THE C-64 BY R. LUETZOW
6 DIM A(100),B(100),ZZ(100),X(100),Y(100),FZ(100),QQ(100)
8 PRINTCHR$(147)
10 PRINT" A DISCRETE FOURIER TRANSFORM PROGRAM"
12 PRINT" ":PRINT"YOU MAY ENTER 16, 32, OR 64 DATA POINTS. HOW MANY POINTS- "
14 INPUT N
16 PRINT"NOW ENTER THE DATA POINTS":PRINT" "
18 IF N = 16 THEN J=4
20 IF N = 32 THEN J=5
22 IF N = 64 THEN J=6
24 FOR I=0 TO N-1
26 PRINTI+1;:PRINT" ";
28 INPUT FM
30 FZ(I)=FM
32 NEXT I
34 PRINT" ":PRINTCHR$(147)
36 PRINT"THE ";N;" DATA POINTS ARE -"
38 PRINT" "
40 FOR I=0 TO N-1
```

Program 10-2. This is a Discrete Fourier Transform program (see reference 3).

```
42 X=FZ(I)
44 A(I)=X
46 QQ(I)=A(I)
48 PRINTQQ(I),;
50 NEXTI
52 IF SQ= 1 THEN GOSUB 208
54 PRINT"PRESS -S- TO GO ON"
56 GET A$: IF A$="S" THEN 60
58 GOTO056
60 PRINT" "
64 PRINT"DISCRETE FOURIER TANSFORM IN PROGRESS":FOR I=0TO1000:NEXT
68 PRINT" ":PRINTCHR$(147)
70 P=3.1415927
72 FOR Z= 0 TO 63
74 A(Z)=A(Z)/N
76 NEXT Z
80 PRINT" "
82 PRINT" "
84 C=N/2:D=1:E=P*2/N
86 FOR I=1 TO J
88 F=0:G=C
90 FOR S=1 TO D
92 H=INT(F/C)
94 GOSUB  184
96 Q=R
98 A1=COS(E*Q):A2=-SIN(E*Q)
100 FOR K= F TO G-1
102 A3=A(K):A4=B(K)
104 B1=A1*A(K+C)-A2*B(K+C)
106 B2=A2*A(K+C)+A1*B(K+C)
108 A(K)=A3+B1:B(K)=A4+B2
110 A(K+C)=A3-B1:B(K+C)=A4-B2
112 NEXT K
114 F=F+2*C:G=G+2*C
116 NEXT S
118 C=C/2:D=D*2
120 NEXT I
122 PRINT"            RE(Z)      IM(Z)      (Z)"
124 :
126 U=0
128 Z=0
130 :
132 :
134 H=U
136 IF U=0 THEN PA$="DC VALUE"
138 IF U=1 THEN PA$="FDM FREQ"
140 PA=U
142 GOSUB  200
144 PB=INT (A(R)*1000)/1000
146 PC=INT(B(R)*1000)/1000
148 PD=INT(ZZ(R)*1000)/1000
150 IF U=0 THEN  156
152 IF U=1 THEN  156
154 GOTO 158
156 PRINTPA$,PB,PC,PD:GOTO160
158 PRINT PA,PB,PC,PD
160 Y(U)=ZZ:X(U)=U
162 U=U+1:Z=Z+1
```

```
164 IF Z=8THEN 168
166 GOTO 172
168 :
170 GOTO128
172 IF U> 12 THEN 176
174 GOTO 134
176 PRINT" ":PRINT"FINISHED"
178 IF SQ=2 THEN GOTO 182
180 :
182 END
184 R=0:N1=N
186 FOR W=1 TO J
188 N1=N1/2
190 IF H<N1 THEN 196
192 R=R+2↑(W-1)
194 H=H-N1
196 NEXT W
198 RETURN
200 GOSUB 184
202 ZZ(R)= SQR(A(R)↑2+B(R)↑2)
204 RETURN



1000 REM PROGRAM 10.3 - FFT PGM
2000 DIMA(128),B(128),P(20),I(20),R(20),V(20):PRINTCHR$(147)
2002 PRINT"     A SIMPLE FFT PROGRAM":PRINT" "
2004 REM* ORIGINAL PGM BY P. L. EMERSON
2006 REM* CONVERTED AND EXPANDED FOR THE
2008 REM* C-64 R. H. LUETZOW
2010 PRINT" THIS PROGRAM REQUIRES 16, 32, OR 64     DATA INPUTS"
2012 PRINT" ":INPUT" ENTER THE NUMBER OF INPUTS - ";N:PRINT" "
2014 M=1:L=0
2016 L=L+1
2018 M=2*M
2020 IF M<N THEN GOTO 2016
2022 T1=2*3.1415927:S1=-1
2024 FOR I=1 TO N/4+1:P(I)=COS(T1*(I-1)/N):NEXT I
2026 PRINT"TYPE IN DATA"
2028 FOR I= 1 TO N
2030 PRINT" ";I" ";
2032 INPUT A(I)
2034 B(I)=0
2036 NEXT I
2038 GOSUB 2064:GOSUB 2108:GOSUB 2132
2040 PRINTCHR$(147)
2042 PRINT"THE FFT DATA DISPLAY"
2044 PRINT"HARMONIC - REAL  -  +/- J  -  VEC SUM"
2046 PRINT" ": NC=13: IF N=16 THEN NC=9
2048 FOR I=1 TO NC
2050 Q2=A(I):Q3=B(I):Q4=V(I)
2052 Q2=INT (Q2* 62.499)/1000
2054 Q3=INT(Q3*62.499)/1000
2056 Q4=INT(Q4*1000)/1000
2058 PRINTI-1,Q2,Q3,Q4
2060 NEXT I
```

Program 10-3. This is a Fast Fourier Transform program (see reference 4).

```
2062 PRINT:PRINT" 0=DC VALUE":PRINT" 1=FDM FREQUENCY":PRINT" ":END
2064 KD=SQR(A(1)↑2+B(1)↑2)
2066 M=N/2:R=1
2068 FOR I = 1 TO L
2070 FOR J=1 TO R
2072 T=1-R:S=2*M*(J-1)
2074 FOR K=1 TO M
2076 S=S+1:X1=A(S):X2=B(S):Y1=A(S+M)
2078 Y2=B(S+M):A(S)=X1+Y1:B(S)=X2+Y2
2080 X1=X1-Y1:X2=X2-Y2:T=T+R
2082 IF T<>1 THEN GOTO 2088
2084 A(S+M)=X1:B(S+M)=X2
2086 GOTO 2102
2088 IF T-1>=N/4 THEN GOTO 2094
2090 U=P(T):V=P(N/4+2-T)
2092 GOTO 2096
2094 U=-P(N/2+2-T):V=P(T-N/4)
2096 IF S1>= 0 THEN GOTO 2100
2098 V=-V
2100 A(S+M)=U*X1-V*X2:B(S+M)=U*X2+V*X1
2102 NEXT K:NEXT J
2104 M=M/2:R=R+R:NEXTI
2106 RETURN
2108 KR=2↑16:KC=2↑4
2110 FOR I=2 TO N-2
2112 R=I-1:J=0:T=1:M=N/2
2114 D=INT(R/M):J=J+D*T
2116 IF M=1 THEN GOTO 2122
2118 T=T+T:R=R-M*D:M=M/2
2120 GOTO 2114
2122 J=J+1:IF J<=I THEN GOTO 2128
2124 X1=A(I):X2=B(I):A(I)=A(J):B(I)=B(J)
2126 A(J)=X1:B(J)=X2
2128 NEXT I
2130 RETURN
2132 FOR I=0 TO 12
2134 R(I)= A(I)*.062475
2136 I(I)= B(I)*.062475
2138 V(I)= SQR((R(I)*R(I))+(I(I)*I(I)))
2140 NEXT I
2142 RETURN
```

Before proceeding on with the next program, we will repeat a specific point from Chapter 9 about aliasing problems when using sampled data with these analysis programs. Aliasing is a phenomenon that can happen when the waveform sampling rate is not fast enough to record all waveform characteristics. Aliasing causes the complex high-frequency waveform components to appear as low-frequency waveform signals when the waveform is not sampled at rate that is at least twice the highest frequency waveform signals when the waveform is not sampled at rate that is at least twice the highest frequency of the complex waveform. If you limit the input waveform frequencies to less than 750 Hz that are recorded with the waveform recording system of Chapter 9 or at least record enough data points to generate a usable time-domain display, you will not have any aliasing problems.

CURVE-FITTING PROGRAM

Along with transform analysis of the recorded waveform, you may need to develop an equation

```
DC TERM = 2.5
HARMONIC
 *   COS      SIN         MAG         PHASE

 1  -.42     -3.19        3.21       -97.46
 2   0         0          0           0
 3  -.42     -1.08        1.15       -111.2
 4   0         0          0           0
 5  -.42     -.73         .84        -119.74
 6   0         0          0           0
FREQ.= 999.999 HZ
```

Fig. 10-2. This is a Fourier Series data display from Program 10-1 for a square wave.

to fit the recorded waveform. Another use for the curve-fitting program is to develop an equation to display nonlinear data on the linear waveform data display. The curve-fitting program is presented in Program 10-4.

This program has been modified to only generate coefficients for a 7th degree polynomial equation. A 7th degree polynomial was selected because the degree of accuracy that is needed for our application requires this level of an equation. The program is also fixed to request lOX -Y data points. The degree of the equation can be adjusted in line 20 and the number of required data points can be changed in line 40 if other requirements come up. The curve-fitting program is easy to use, just load it up and follow the program instructions.

CONCLUSION

In this chapter, you have been shown how a

```
            RE(Z)       IM(Z)       (Z)
DC VALUE    2.499       0           2.5
FDM FREQ   -.313        1.571       1.601
 2           0         -1E-03       0
 3          -.313       .467        .562
 4           0         -1E-03       0
 5          -.313       .208        .375
 6           0         -1E-03       0
 7          -.313       .062        .318
 8           0          0           0
 9          -.313      -.063        .318
 10          0          0           0
 11         -.313      -.209        .375
 12          0          0           0

FINISHED
```

Fig. 10-3. This is a DFT data display from Program 10-2 for a square wave.

```
1 REM PGM 10.4 - CURVE FITTING PROGRAM
2 REM SEE REFERENCE #4
4 PRINTCHR$(147):PRINT"  A WAVEFORM CURVE FITTING PROGRAM TO      GENERATE";
5 PRINT" COEFFICIENTS FOR A 7TH";
6 PRINT" DEGREE  POLYNOMIAL EQUATION USING TEN  X,Y       DATA  POINTS";
8 PRINT" FROM A KNOWN WAVEFORM       CURVE.":PRINT" "
10 PRINT" ENTER TEN X,Y DATA POINTS"
15 PRINT" "
20 D=7
30 DIM A(2*D+1),R(D+1,D+2),T(D+2)
40 N=10
50 A(1)=N
60 FOR I=1 TO N
100 PRINT"X,Y OF POINT ";I;
110 INPUT" :";X,Y
120 FOR J=2 TO 2*D+1
130 A(J)=A(J)+X↑(J-1)
140 NEXTJ
150 FOR K=1 TO D+1
160 R(K,D+2)= T(K)+Y*X↑(K-1)
170 T(K)=T(K)+Y*X↑(K-1)
180 NEXTK
190 T(D+2) = T(D+2) + Y↑2
200 NEXTI
205 PRINTCHR$(147):PRINT"NOW COMPUTING"
210 FOR J=1 TO D+1
220 FOR K= 1 TO D+1
230 R(J,K)=A(J+K-1)
240 NEXT K
250 NEXT J
260 FOR J = 1 TO D+1
270 K=J
280 IF R(K,J)< > 0 THEN 320
290 K=K+1
295 IF K=D+1 THEN 280
300 PRINTCHR$(147):PRINT"NO UNIQUE SOLUTION"
310 GOTO790
320 FOR I=1 TO D+2
330 S=R(J,I)
340 R(J,I)=R(K,I)
350 R(K,I)=S
360 NEXT I
370 Z=1 /R(J,J)
380 FOR I=1 TO D+2
390 R(J,I)=Z*R(J,I)
400 NEXT I
410 FOR K= 1 TO D+1
420 IF K= J THEN 470
430 Z= - R (K,J)
440 FOR I=1 TO D+2
450 R(K,I) = R (K,I)+Z*R(J,I)
460 NEXT I
470 NEXT K
480 NEXTJ
490 PRINTCHR$(147)
495 PRINTTAB(13)"CONSTANT = "; R(1,D+2)
```

Program 10-4. This is a polynomial curve-fitting program (see reference 4).

```
500 FOR J=1 TO D
510 PRINTJ;" DEGREE COEFFICIENT = "; R(J+1,D+2)
520 NEXT J
530 PRINT" "
540 P=0
550 FOR J=2 TO D+1
560 P=P+R(J,D+2)*(T(J)-A(J)*T(1)/N)
570 NEXT J
580 Q=T(D+2)-T(1)↑2/N
590 Z=Q-P
600 I=N-D-1
620 PRINT" "
630 J=P/Q
670 PRINT" ":PRINT"ENTER 9999 TO EXIT":PRINT" "
680 :
690 P=R(1,D+2)
700 INPUT "VALUE OF X";X
710 IF X=9999 THEN 790
730 FOR J=1 TO D
740 P=P+R(J+1,D+2)*X↑J
750 NEXT J
760 PRINT TAB(10) "Y= ",P
770 PRINT" "
780 GOTO 680
790 END
READY.
```

computer program can be used to help you analyze a recorded waveform. One suggested project would be to use the recorded waveform data points from the pendulum project in Chapter 9 as input data for the Fourier Series Program 10-1. Program 10-1 will tell you the frequency spectrum and period of the pendulum. As you study the applications of changing time-domain measurements into frequency-

```
THE FFT DATA DISPLAY
HARMONIC - REAL   -   +/- J   -   VEC SUM

 0          2.499      0          2.498
 1         -.313       1.571      1.601
 2          0          0          0
 3         -.313       .467       .562
 4          0          0          0
 5         -.313       .208       .375
 6          0          0          0
 7         -.313       .062       .318
 8          0          0          0

 0=DC VALUE
 1=FDM FREQUENCY
```

Fig. 10-4. This is a FFT data display from Program 10-3 for a square wave.

domain measurements, you will see that difficult analysis problems in one domain may be understood more clearly when transformed into the other domain.

References

1. Hewlett Packard, Inc., *Fourier Analyzer Training Manual*. Application Note 140-0.
2. Programs 10-1 and 10-2: (Program Routines courtesy of:) Howard M. Berlin, *Circuit Design Programsfor the TRS-80*. Howard W. Sams, Indianapolis, Indiana, 1980, pg. 38.
3. Program 10-3: Phillip L. Emerson, Ph.D., "Fast Fourier Transform Fundamentals and Applications," *Creative Computing,* July 1980, pg. 58.
4. Program 10-4: Lon Poole, et al. *Some Common BASIC Programs, Apple II Edition*. OSBORNE/McGraw-HiII, pgs. 156-157, Nth Order Regression Program.

Bibliography

Poole, Borchers, and Castlewitz, *Some Common BASIC Programs, Apple II Edition*. OSBORNE/McGraw-Hill, 1980.

Berlin, H.M., *Circuit Design Programs For the TRS-80*. H.W. Sams, 1980.

Emerson, Ph.D., P.L., "Fast Fourier Transform Fundamentals and Applications," *Creative Computing,* July 1980.

Hewlett Packard, Inc., *The Fundamenta13of Signal Analysis* Application Note 243.

Malmstadt, Enke, and Crouch, *Instrumentation For Scientist Series*. Benjamin/Cummings, 1973.

Oppenheim, A.V., *Digital Signal Processing*. M.I.T. Press, 1969.

Roxburgh, *A.,* "Fast Fourier Comes Back." *BYTE,* May 1981.

Stanley, W.D. "Fast Fourier Transforms on Your Home Computer," *BYTE,* Dec. 1978.

Chapter 11

Power Supplies

THE POWER SUPPLY IS A COMMON ELECTRONIC circuit which is used in almost all electronic instruments. Even the electronic instrument that uses a battery will usually have a voltage-regulator circuit in it to supply a constant current or a constant voltage to the electronic circuits. Most generally, the power supply is a simple circuit which can be built easily or can be purchased at a low cost. Unless you are a school student who cannot afford fifty dollars for a power supply, I would advise that you buy commercially made units.

A lot of good books have been published about designing power supplies, and so we will not compete with them by presenting a long and dry essay on power supply designing. It will be, however, the aim of this chapter to describe some commercial power supplies and their characteristics, plus supply you with several power supply schematics that you can build. All of these power supplies can be built with electronic components that are usually available at a local electronics hobby store. Now, go out and buy a good supply of one-amp fuses, and we will continue on this chapter!

COMMERCIALL Y MANUFACTURED POWER SUPPLIES

The power supplies that I will be describing are manufactured by Standard Power Incorporated. I am not selling Standard power supplies, but all three of the electronic supply houses that I buy from sell Standard Power power supplies and I also just happen to have a few around, one of which is shown in Fig. 11-1. There are several other competing companies which manufacture equivalent power supply units that probably function as well and are competitively priced. I have had very little trouble with the purchased power supplies that I have used, and I can generally state that if they work the first twenty-four hours, they will work for the next two or three years unless you put it to them in some way.

The commercially manufactured power supplies are generally protected from overloads and it is really hard to bum one of them up if they are used properly. The overload protection does not mean that you can run a power supply into a short circuit because sooner or later the power supply will

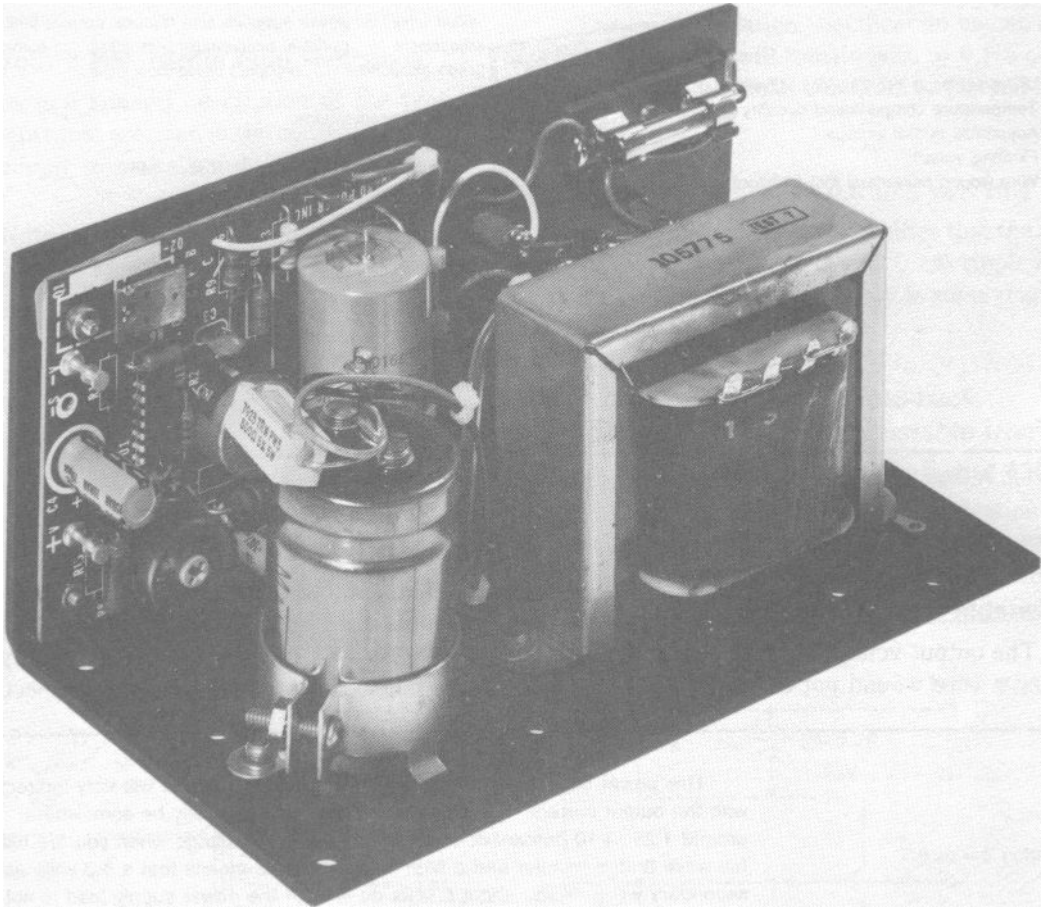


Fig. 11-1. This is a commercially manufactured power supply (courtesy Standard Power Inc.).

heat up and the higher leakage currents that are caused by the heat will wipe out one of the circuit components. If the power supply that you are using is rated at 3 amps and you are only pulling three-quarters of an amp in your circuit, put a one amp fuse in the dc output line and protect everything; some day you will be glad you did. We will now go through a short discussion of what each of the power supply specifications means using Fig. 11-2.

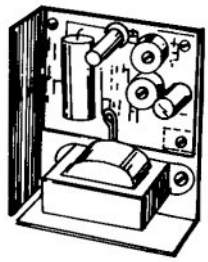
Universal Input *115/230* Vac, 47-440 Hz

The *universal input* means just that. You can connect the power supply to either a 115 Vac or a 230 Vac line voltage, using the proper input connections, and the power supply outputs will be within specification. The input line frequency can be anything from a 47Hz sine wave to a 440 Hz sine wave which covers about all of the input line frequencies that you will ever see.

Temperature Compensated Circuitry

The temperature compensation comes from the integrated-circuit chip that is used in the voltage regulation circuit. Temperature compensation means that the output specifications will be maintained as long as the power supply is operated

- Universal input 115/230 Vac, 47-440 Hz
- Temperature compensated circuitry
- Adjustable output voltage
- Floating output
- Wire wound pots/metal film resistors



"Blue Line" dc power supplies also feature: current limit adjustment . . . optional overvoltage protection . . . computer grade capacitors . . . compact design/low cost.

General Specifications:

Input voltage: 115/230 Vac ± 10%, 47-440 Hz
Line regulation: ±0.1%
Load regulation: ±0.1% 0 to full load
Ripple: 0.1%, typically 0.5 to 2 mv rms
Short circuit protection: fold-back type, current limiting, adjustable from 20% to 150% of load. Factory set at 110%
Response time: 50 microseconds
Temperature coefficient: 0.02%/°C
Temperature rating: 0° to 50°C (to 70° derated)
Adjustable voltage range: ±10%

Fig. 11-2. A Blue Line power supply specification. (courtesy Standard Power Inc.).

within its temperature range.

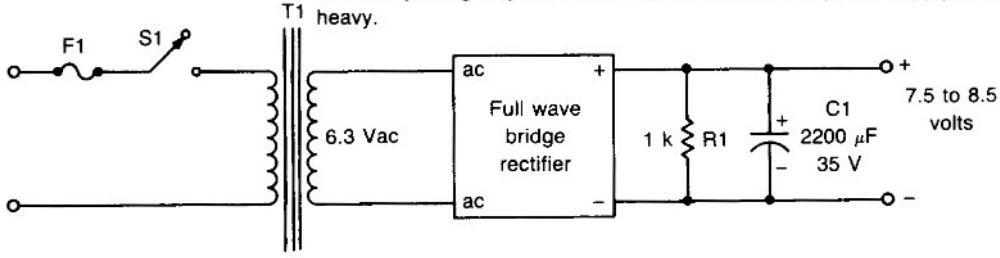
Adjustable Output Voltage

The output voltage may be controlled by adjusting a wire wound pot on the pc board.

Floating Output

The floating output means tht neither of the dc output terminals (+ or -) are connected to ground. So, you can use this power supply as either a positive or negative supply by connecting the

This power supply is unregulated and the output voltage will vary indirectly with the output current. The dc output voltage will generally be somewhere around 1.25 (±10 percent or so) times the ac input voltage when you are using a full wave bridge rectifier and a filter capacitor. This means that a 6.3 volts ac secondary will give you about 8 volts dc or so if the power supply load is not too heavy.



Parts List: Unregulated Positive Voltage Power Supply

F1 - Fuse 1 amp
S1 - 3 Amp 120 Vac switch
T1 - Radio Shack 273-1505
B1 - Radio Shack 276-1180
R1 - 1k 1/2 watt resistor
C1 - 220 μF at 35 Vdc

Fig. 11-3. A basic unregulated power supply.

proper output terminal to ground.

Wire Wound Pots/Metal Film Resistors

This is a general description of the type of components that are used in the construction of the power supply to make it a dependable unit.

Input Voltage: 115/230 Vac ± 10%,47-440 Hz

The input voltage specifications means that you can vary the power-supply line voltage ± ten percent from the specified values and the output will remain within the specified limits.

Line Regulation: ± 0.1 %

The line regulation specification means that the output voltage will remain within ± .01% of its set value if the input line voltage is varied over the +/- 10% range.

Load Regulation: ± 0.1 % 0 to Full Load

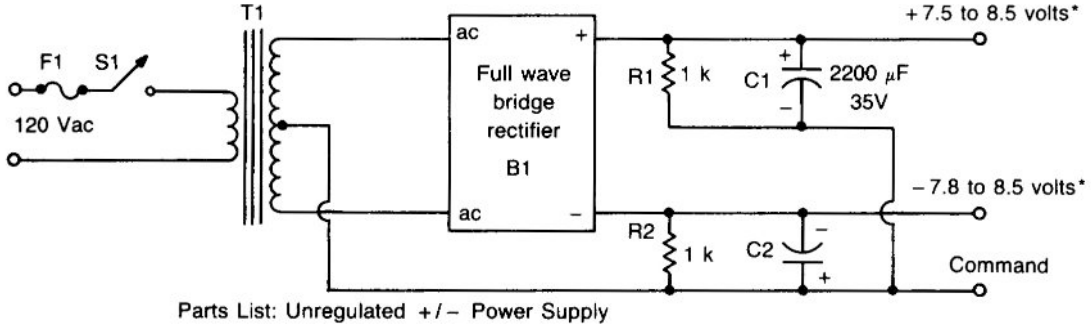
The load regulation specification means that the output voltage will remain with ± 0.1% of its set value from a no-load condition to the full load condition.

Ripple: 0.1 %, Typically 0.5 to 2 mV rms

The ripple specification specifies that the output voltage will not have over a 0.1 % ripple content on it when the power supply is supplying the full load current.

Short Circuit Protection: Fold-back Type, Current limiting, Adjustable from 20% to 150% of Load. Factory Set at 110%.

The over load protection specification indicates the type of protection circuitry that is used, and specifies that you can adjust the output current

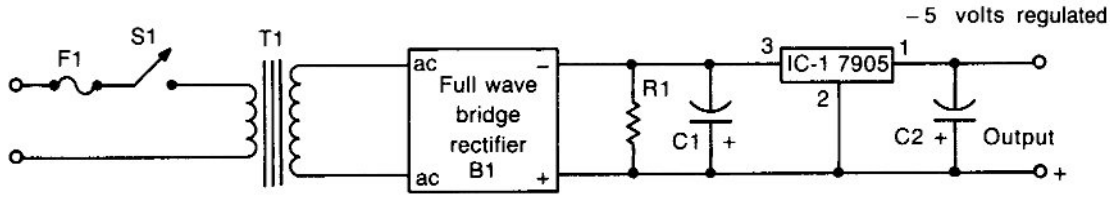


Parts List: Unregulated +/– Power Supply

F1 - Fuse 1 amp
S1 - 3 Amp 120 Vac switch
T1 - Radio Shack 273-1505
B1 - Radio Shack 276-1180
R1, R2 - 1 k 1/2 walt resistor
C1, C2 - 2200 μF at 35 Vdc

This power supply is unregulated and the output voltage will vary indirectly with the output current. The dc output voltage will generally be somewhere around 1.25 (± 10 percent or so) times the ac input voltage when you are using a full wave bridge rectifier and a filter capacitor. This means that a 6.3 volts ac secondary will give you about 8 volts dc or so if the power supply load is not too heavy.

Fig. 11-4. A basic positive and negative unregulated power supply.



Parts List for Regulated – 5 Volts Power Supply

F1 - Fuse 1 amp
S1 - 3 Amp 120 Vac switch
T1 - Radio Shack 273-1505
B1 - Radio Shack 276-1180
R1 - 1 k *1/2* watt resistor
IC1 - 7905 - Radio Shack 276-1773
C1 - 2200 μF at 35 Vdc
C2 - 1 to 5 μF at 16 Vdc

This power supply will deliver over 1 amp of dc current at a regulated output voltage of negative 5 volts. The IC regulator contains a circuit for short circuit protection and a thermal overload protection. The unit must have a proper heat sink if it is going to regulate at maximum current levels. You can estimate the IC heat dissipation by multiplying the dc current through the IC times the voltage drop across the IC.

Fig. 11-5. A negative 5 volt power supply.

level from 20% to 150% of the power-supply current rating. It also indicates that the factory sets the current limiting control to 110% of the power-supply rating when the unit leaves the factory. Short circuit protection *does not mean* that you can leave a short circuit across the power supply output continuously without damaging the power supply.

Response Time: 50 Microseconds

The response time tells you how long it will take for the voltage regulation circuit to react to a changing load condition.

Temperature Coefficient: 0.01 % Per Degree C

The temperature coefficient specifies that the output voltage will not change more than 0.02% from its set voltage value per degree of temperature change. (Temperature is in degrees centigrade.)

Temperature Rating: 0 Degrees to 50 Degrees C (to 70 Degrees C Derated)

The temperature rating indicates that the power supply will operate over a temperature range of 0 to 50 degrees centigrade, and can function at a temperature of 70 degrees centigrade if its output is derated by a given amount.

Adjustable Voltage Range: ± 10%

This specification indicates the output voltage may be varied plus or minus ten percent from the specified output voltage by turning an on-board wire-wound pot control.

The above specifications pretty well characterize the performance that you can expect from the power supply. If these specifications leave anything unmentioned, all of the power supply manufacturers have application engineers that will be happy to answer any questions you may have

about one of their products.

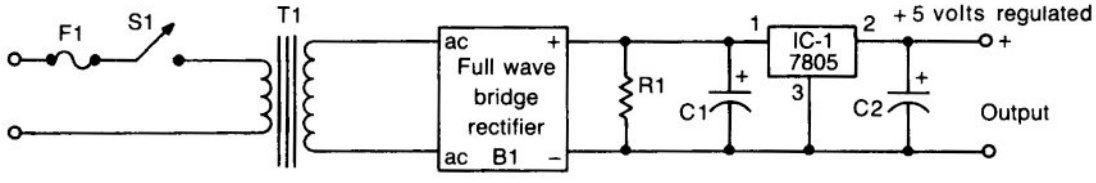
POWER SUPPLY CONSTRUCTION PROJECTS

The power supplies that are described in the rest of this chapter can be constructed with hobby store parts. But, they are still very good power supplies and will do just as good of a job for you as a commercially manufactured unit with the same voltage and current ratings. The main thing that you will have going if you build your own power supply is the fact that you will be able to easily repair it with parts that you can buy. Sometimes it is not very easy to find the parts that are in the commercially manufactured power supplies.

All of the required parts for the power supplies that are shown in Figs. 11-3 through 11-7 can be purchased at Radio Shack or most other hobby electronics shops.

I have built each one of the power supplies at one time or another and you should have no problem building them. Make sure that all of the power transistors and IC regulators are connected to a heatsink that has been coated with a silicon for heatsink compound for good heat transfer between the active circuit element and the heatsink. The majority of all problems that you will encounter in a power supply will usually be heat related in some way.

Always be sure that you fuse your power supply for its protection and your protection. An unfused power supply can cause large amounts of smoke and pungent smells. Also, an unfused ac power line can burn off the tip of a small screwdriver or burn a pretty good hole in a pair of wire cutters (which then can be used to strip number 10 and larger insulated wire).

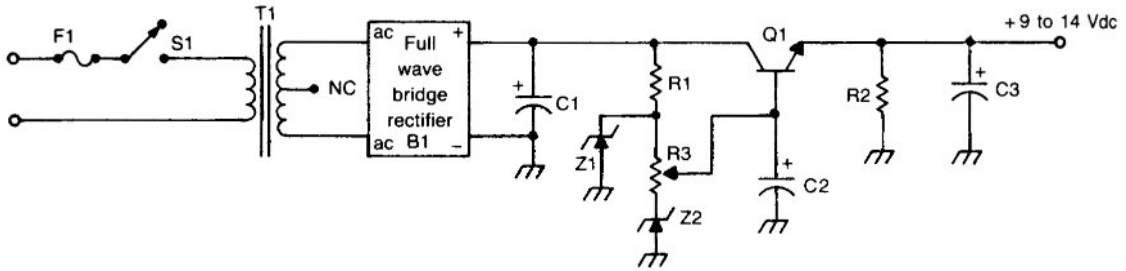


Parts List for Regulated 5 Volts Power Supply

F1 - fuse 1 amp
S1 - 3 Amp 120 Vac switch
T1 - Radio Shack 273-1505
B1 - Radio Shack 276-1180
R1 - 1 k *1/2* watt resistor
IC1 - 7805 Radio Shack 276-1770
C1 - 2200 μF Vdc
C2 - 1 to 5 μF at 16 Vdc

This power supply will deliver over 1 amp of dc current at a regulated output voltage of 5 volts. The IC regulator contains a circuit for short circuit protection and a thermal overload protection. The unit must have a proper heat sink if it is going to regulate at maximum current levels. You can estimate the IC heat dissipation by multiplying the dc current through the IC times the voltage drop across the IC.

Fig. 11-6. A positive 5 volt power supply.



Parts List for the Junk Box Power Supply

F1 . Fuse 1 amp
Sl ·3 amp 120 Vac switch
T1 • Radio Shack 273-1515
B1 . Radio Shack 276-1180
C1 . 2000 μF at 35 volts
C2· 1000 μF at 16 volts
C3 - 1 μF at 16 volts
R1 - 390 ohms at 1 watt
R2 . 1 k at 1/2 watt
R3 • 500 ohm 2 watt pot
Q1 • Radio Shack 276-2020
Zl . 14 zener
Z2 • 9V zener

This power supply is adjustable between 9 and 14 volts and can deliver up one amp of dc current. The voltage regulation of this power supply is not as good as a power supply with a commercial IC regulator unit, but this power supply can usually be built from junk box parts and little else.

Fig. 11-7. An adjustable power supply.

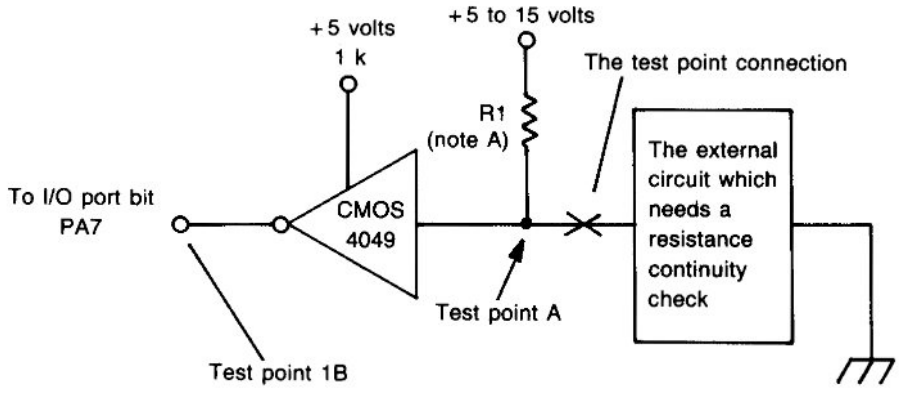
Chapter 12

Computer-Controlled Electronic Measurements

THIS CHAPTER IS ABOUT THE ELECTRONIC measurement of resistance, capacitance, inductance, and continuity that one will encounter in the manufacturing of an electrical part on a production line. If you are a school student, this chapter will give you a good view of a style of testing called go/no-go that will be encountered in an industrial environment. Just about all electrical or electronic items that are manufactured require an electronic or electrical measurement check at some point during their manufacturing trip down the production line. This chapter will show you how to make these measurements using a computer to control the measuring process. The measurements can be as simple as a few continuity checks during the manufacturing process, or when the produced item is finished, a complete series of resistance measurements to verify the product's integrity. The faster that these measurements can be made, the lower the cost of producing the item. It is very easy to make a computer-controlled measurement system that can measure the resistance of a resistor or coil of wire in a time period that is much shorter than the display set-up time of a LED display ohmmeter. When the computer-controlled measurement system is combined with a robotic handling system, you will have a high-speed automatic manufacturing test system.

CONTINUITY CHECKS

Continuity checks are used to verify that a complete electrical circuit exist between points. The check does not show whether the electrical circuit is a high-resistance or low-resistance circuit, but only that an electrical current path does exist between two points. Most of the time, a continuity check is the only check that is needed to completely satisfy an in-process type of production resistance measurement. A continuity check test is really not too involved. You can easily use a simple ohmmeter and two leads to make the needed test. This type of testing is ok if you only need a few checks in an hour's time. But, if you require 10,000 check per hour or multipath checks, it is time to use a computer to automate the continuity testing. A low cost



NOTE: The resistor R1 determines what dc current will flow through the external circuit which is receiving the continuity check. The current that flows through the external circuit must be enough to cause a voltage drop across R1 that leaves no more than 1 volt at test pOint A. If the external circuit has the required continuity, a logic one will be generated at test point B.

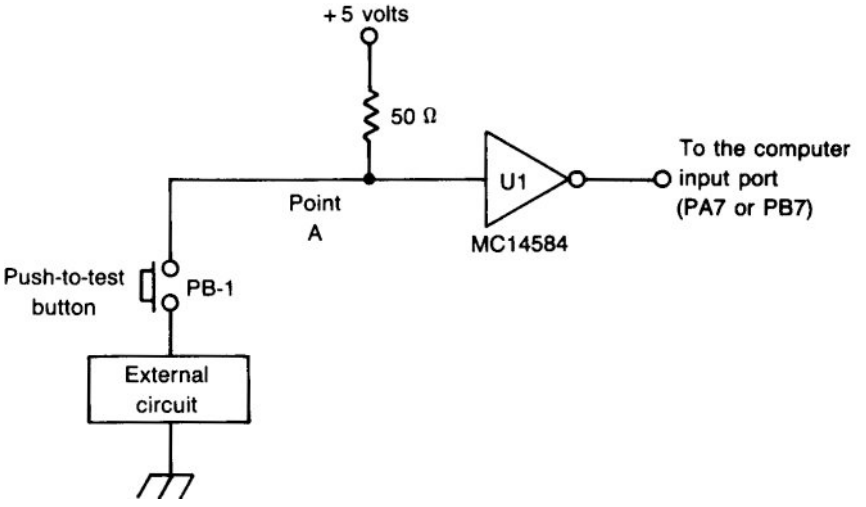
Fig. 12-1. The simple dc continuity test circuit.

VIC-20 computer can easily be used to build a high-speed continuity test system.

A continuity test circuit is shown in Fig. 12-1 that is simple and does a very good job. The circuit uses a CMOS 4049 hex inverter IC which detects the voltage drop through R1, a lK resistor. The value of resistor R1 should be at least 10 times larger the resistance of the circuit that is being tested. Also the resistance of R1 must be high enough to keep the resistor from overheating or overloading the 5 to 15 volts power supply. The value of R1 also must permit a voltage drop across itself that will set-up a voltage at test point A of no higher than 1 volt when the circuit continuity is considered GOOD. When the voltage at test point A is 1 volt, the 4049 will generate a logic ONE at it's output, which is test point B. This logic ONE signal can be used by the computer to generate a part GOOD signal. If only one or two continuity tests are being made, an LED can be connected to the 4049 at test point B to make a hand held go/no-go type tester. Program 12-1 shows a simple BASIC

```
10 REM : PROGRAM 12.1
20 REM : A CONTINUITY TEST PROGRAM FOR FIGURE 12.1 USING A C-64
30 REM : USE THE 6522 VIA BOARD OF CHAPTER 6
40 REM : CONNECT THE TEST CIRCUIT TO PORT LINE PA7
50 PRINTCHR$(147)
60 A=PEEK(57089)
70 IF A>127 THEN GOTO 90
80 IF A<128 THEN GOTO 120
90 PRINTCHR$(19)
100 PRINT"NO CIRCUIT CONTINUITY
110 GOTO 60
120 PRINTCHR$(19)
130 PRINT"CIRCUIT CONTINUITY IS GOOD"
140 GOTO 60
```

Program 12-1. A continuity test program for Fig. 12-1.



The resistance of the circuit under test must be low enough to draw 50 mA so point A will go low to a logic 0 when the push-to-test button is pressed

Fig. 12-2. The push-to-test continuity circuit.

program that can be used with the continuity test circuit of Fig. 12-1. A continuity testing routine is very similar to a routine which looks for an open or closed switch.

Most of the time in production style testing, you will have many different test measurements that must be made to a product before it can be shipped. One of these tests will usually be a continuity test of a ground lug or connecting bracket. It is very easy to use a continuity test circuit as shown in Fig. 12-2 in a push-to-test function to check the circuit's continuity and, at the same time, start the test sequence if the continuity test is good. If the circuit's continuity is bad, the testing sequence can not start. A customer is generally impressed when they see that the product that you are producing can not pass through a final test system if a specified ground bracket or something similar is not made correctly. Program 12-2 is an example of how to start a *push-to-test routine*.

```
10 REM : PROGRAM 12.2
20 REM : A PUSH-TO-TEST PROGRAM FOR FIGURE 12.2 USING A C-64
30 REM : USE THE 6522 VIA BOARD OF CHAPTER 6
40 REM : CONNECT THE TEST CIRCUIT TO PORT LINE PA7
50 PRINTCHR$(147)
60 A=PEEK(57089)
70 IF A>127 THEN GOTO 90
80 IF A<128 THEN GOTO 120
90 PRINTCHR$(13)
100 PRINT"PRESS PUSH BUTTON TO START TEST "
110 GOTO 60
120 PRINTCHR$(19)
130 PRINT"CIRCUIT CONTINUITY IS GOOD        "
140 GOTO 60
```

Program 12-2. A push-to-test continuity test program.

AN EIGHT-CIRCUIT CONTINUITY TEST METHOD

The VIC 20 or the C-64 will function nicely as an eight-circuit continuity tester with the addition of a 6522 I/O board such as those described in Chapters 4 and 6. It is possible to use both of the PA and PB ports 6522 VIA to make a continuity test that can test the continuity of eight independent circuits while also checking to see if there are any shorts between the independent circuits. This testing method is great for checking cables and small circuit boards.

The continuity test method that will be presented uses the 6522 Port A 110 lines as the continuity test drivers and detectors, and the Port B 110 lines as current-sinking lines. Figure 12-3 shows that a 6.8K pull-up resistor is added to each Port A 110 line to furnish the driving current for the continuity test. The Port B 110 lines are set up in the output mode and used to pull the voltage on the corresponding Port A line to a logic ZERO by setting each Port B line to a logic ZERO. If a continuity path exist between each corresponding port line, all Port A 110 lines will be pulled to a logic ZERO. Any Port A 110 line that is not pulled low will indicate that a bad continuity path exists between Port A and Port B.

Figure 12-3 also shows how to connect an external circuit to the Port A and Port B 110 lines so you can use the continuity test Program 12-3 to do continuity testing. To secure the continuity checks, all Port A lines must be set up as input lines and all Port B lines must be set up as output lines. All Port A lines will supply five volts to each of the external circuits that are to be tested. When you set all Port B lines to logic ZEROs, all Port A lines should go to a logic ZERO if the continuity path is good. You can also test for shorts between the external circuits by setting all Port B line to a logic ONE and then setting each one to a logic ZERO one at a time. If any other line goes to a logic ZERO besides the one that should, you will have a short between those two external circuit paths. You now have a method of testing for continuity and electrical shorts.

The only problem that might be experienced is with external circuit capacity. Sometimes long multiwire cables can have high levels of stray capacitance between the wires. This stray capacitance can cause problems if the continuity testing cycle is ran too fast. This problem can be overcome by using a FOR-NEXT loop time delay of .1 seconds between applying the continuity check voltage and checking for continuity.

WINDOW COMPARATORS

Most of the testing problems that you will encounter in a production operation will be the go/no-go type of tests. One of the most useful circuits that you can have in your arsenal of test circuit weapons to fight the go/no-go problem is the window comparator circuit. This circuit is built around two operational-amplifier circuits that are arranged so they will turn off and on at specific voltage levels. You can adjust the two operational amplifiers so both amplifiers will put out a logic one signal when the same input signal to both amplifiers is in between two given voltage levels which forms the voltage window. The window comparator circuit can be used for a large number of applications some of which are checking resistance, pulse measurements, linear slope measurements, and thermistor temperature-controller, and so on.

The window comparator circuit, shown in Fig. 12-4, is designed to detect a voltage window that can be adjusted anywhere between 2.1 to 2.9 volts with a five-volt supply voltage. The narrowest window voltage that can be reasonably detected is about .010 volts. The circuit uses a UA358 dual operational amplifier IC (UIA and UIB) that can operate from a single supply voltage source. The logic circuits are all CMOS chips, so you can use a supply voltage of 5 to 15 volts with this comparator circuit. The circuit is designed for a voltage window that is in the middle of the supply voltage range, but you can adjust the resistors RA, RB, RC, RD, RE, and RF to operate a voltage window about anywhere within the supply voltage range. You should stay about one-half volt away from the rail voltages.

The upper window-voltage limit is controlled by minipot RA and the lower voltage limit is con-

Use the 6522 I/O board of Chapter 6

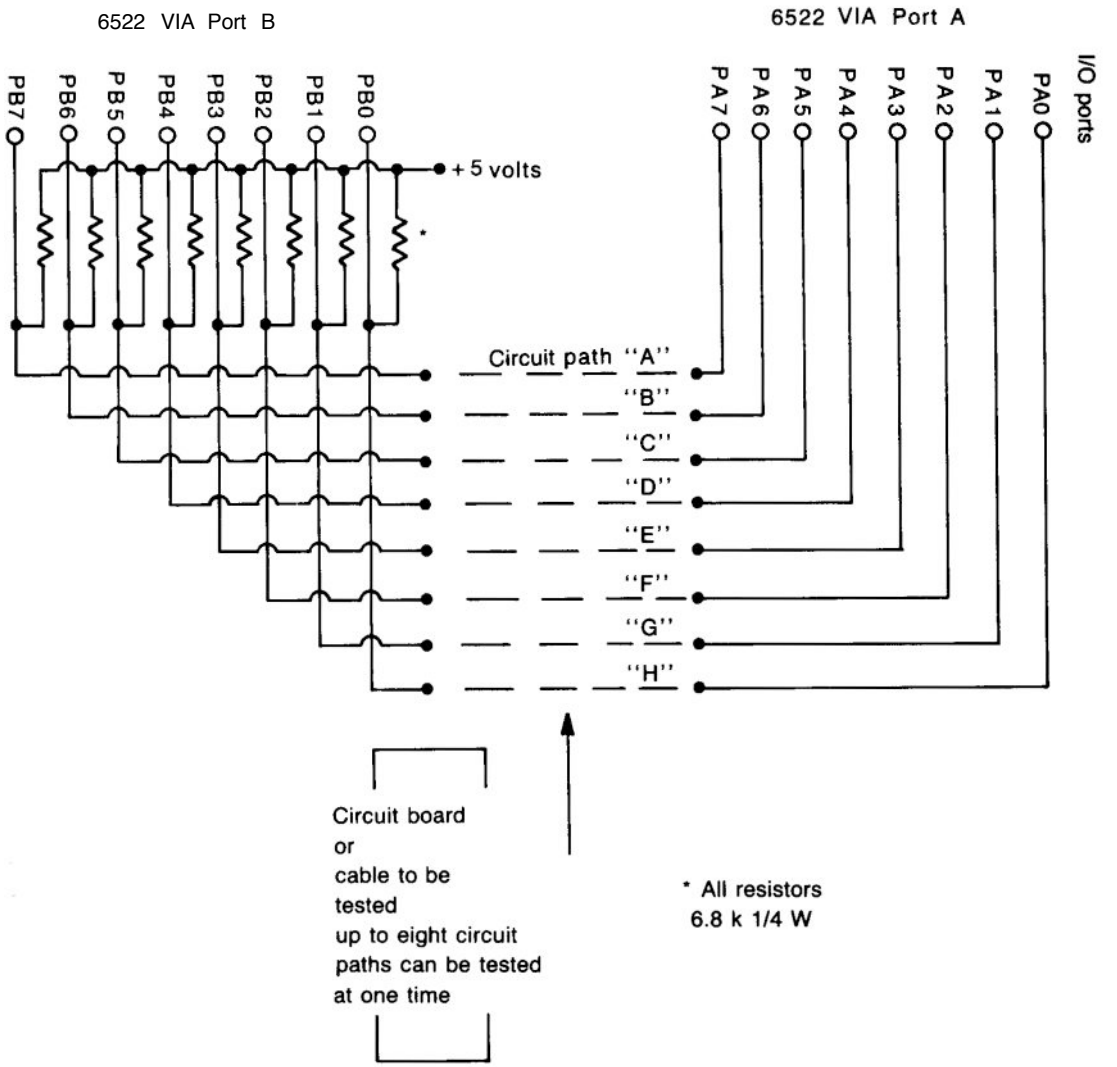
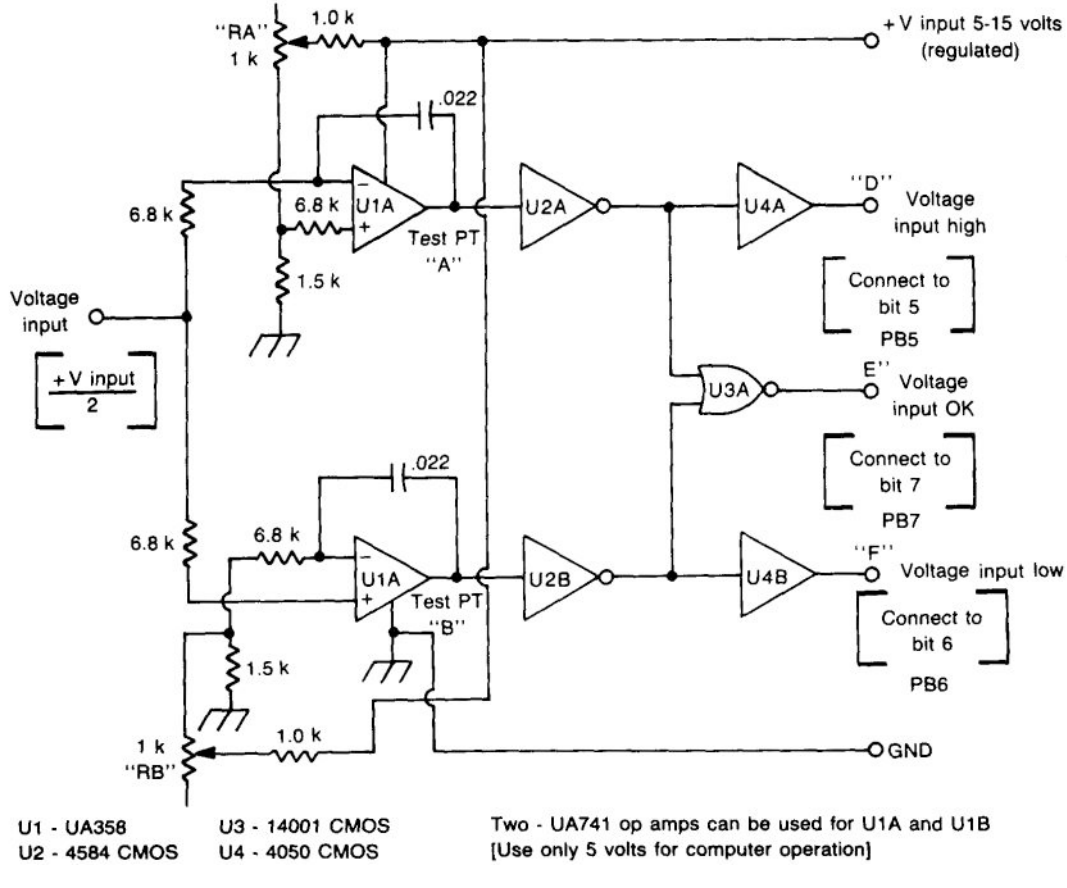


Fig. 12-3. The eight-circuit continuity test schematic.

trolled by minipot RB. When the input voltage is within the voltage window limits, both operational amplifiers will generate logic one output signals are inverted by the two inverter circuits of U2A and U2B (4584). The inverted output signals will both be logic zeros that are then applied to the two inputs of the NOR circuit of U3A. When both inputs to U3A are zeros, the output will be a logic one which indicates that the input voltage is within the window-voltage limits. If the input voltage is outside either one of the window-voltage limits, the output of one of the operational amplifiers will be

```
10 REM : PROGRAM 12.3
20 REM : AN 8 CIRCUIT CONTINUITY TEST PROGRAM FOR FIGURE 12.3 USING A C-64
30 REM : USE THE 6522 VIA BOARD OF CHAPTER 6
40 REM : CONNECT THE TEST CIRCUIT TO PORTS PA AND PB AS SHOWN IN FIGURE 12.3
50 PRINTCHR$(147):PRINT"PRESS C TO CHECK CONTINUITY"
60 POKE 57090,255:POKE57088,00
70 GET A$: IF A$="C" THEN GOTO 90
80 GOTO 70
90 A=PEEK(57089)
100 IF A>000 THEN GOTO 120
110 IF A=000 THEN GOTO 160
120 PRINTCHR$(19)
130 PRINT"CONTINUITY TEST IS BAD            "
140 GOTO 170
150 PRINTCHR$(19)
160 PRINT"CIRCUIT CONTINUITY IS GOOD        "
170 FOR I = 1 TO 3000:NEXT: GOTO 50
```

Program 12-3. An eight-circuit continuity test program.



Fia. 12-4. The voltage-window comparator circuit.

a logic zero. That logic zero will be inverted to a logic one which will turn off the NOR circuit indicating that the input voltage is outside of the window voltage limits. The logic one signal will also be sent on to one of the input voltage-high or input voltage-low output pins by the noninverting buffer circuits U4A or U4B (4050).

It is very easy to interface a voltage comparator to a computer. The comparator circuit will give you one of three logic output signals, which are voltage high, voltage low, or voltage ok. These three logic signals can be connected directly to the USER PORT lines so the computer can use the comparator circuit to test for the three voltage levels. Program 12-4 shows you how to test for four possible conditions which are voltage high, voltage low, voltage OK, or to check the test circuit in case the wrong logic signals are received if the test circuit fails. The program assumes that your voltage comparator circuit is connected and supplying a logic signal when the program is executed. This program is an example of how you can test the product and verify that your test circuit is functioning at the same time.

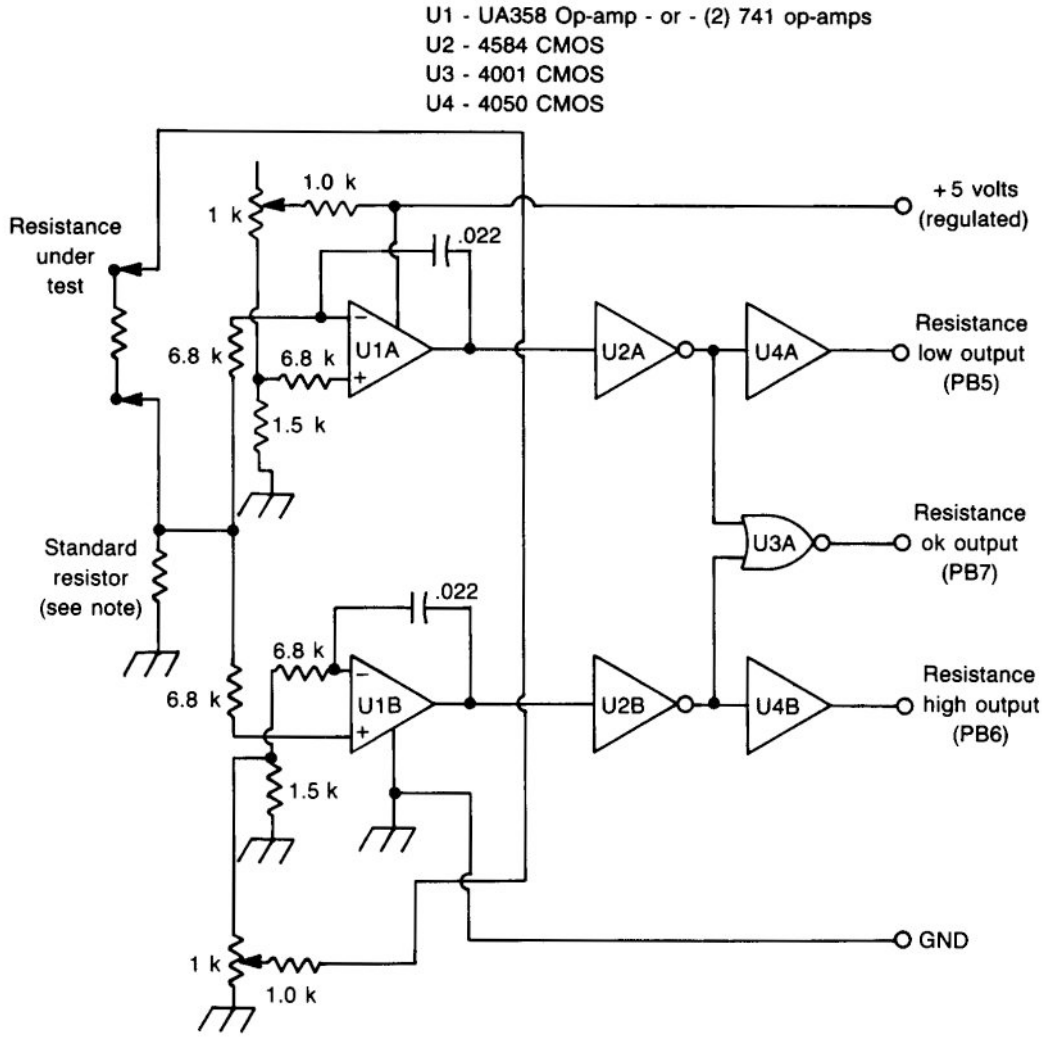
Using the Window Comparator to Check Resistance

The window comparator's main objective is to inform you when the input voltage is at a given voltage level. If you connect to resistors as a voltage divider on the input, as shown in Fig. 12-5 you can set up the window comparator circuit to inform you when both resistors have the same resistance value. A standard resistor or a decade resistor box can be connected between the comparator input and ground and when an unknown resistance is connected between the input and the positive supply voltage, which is equal to the standard resistance, the voltage at the input of the comparator will be one-half of the supply voltage. The window comparator circuit can be set up to detect that one-half supply voltage point plus whatever tolerance level you may wish to adjust into the test system. The test resistance can be about anything like a coil of wire, a heating element, or a resistor, because you are comparing the unknown test resistance with a known standard resistance in a voltage divider circuit that does not care about anything but dividing the voltage as per the values of resistance in the circuit and Ohm's law.

Using the resistance-checking window comparator with one of the four computers that we are using in this book is not too different from what we did with the voltage-comparator circuit. Just make sure that you observe which port bits are used for the high and low resistance connections. They are the opposite of the high and low voltage comparator connections that were used in Fig. 12-4. Program 12-5 is a resistance-checking program that is used with the circuit of Fig. 12-5.

```
1 REM : PROGRAM 12.4  FOR THE C-64
2 REM : VOLTAGE CHECKING PROGRAM FOR FIGURE 12.4.
3 REM : USE THE C-64 USER PORT AT ADDRESS 56577
5 PRINTCHR$(147)
10 PRINT"CHECKING VOLTAGE COMAPRATOR"
20 IF PEEK(56577)=32 THEN GOTO 100
30 IF PEEK(56577)=64 THEN GOTO 150
40 IF PEEK(56577)=128 THEN GOTO 200
50 PRINT"CHECK TEST CIRCUITS"
60 END
100 PRINT"VOLTAGE CHECK LOW":END
150 PRINT"VOLTAGE CHECK HIGH":END
200 PRINT"VOLTAGE CHECK OK":END
```

Program 12-4. **A voltage-checking program for** Fig. 12-4.



NOTE: The standard resistor must have a resistance value which is high enough to limit the current through it to a save value in case the resistance under test is shorted.

Fig. 12-5. The voltage comparator circuit designed for checking resistance.

INTERFACING TO A BRIDGE CIRCUIT

You can only do so much testing by measuring dc and ac voltages and currents. When it comes time to measure other electrical quantities, such as resistance, inductance, and capacitance, the test systems engineer will need to resort to more advanced testing methods that will characterize the component in question by securing the required technical data. The basic circuit that is used to secure this type of technical data is called the *bridge circuit*. The theory of the bridge circuit can vary from simple to very complicated. In this chapter we will assume that you have a basic idea of what a

```
1 REM : PROGRAM 12.5  FOR THE C-64
2 REM : RESISTANCE CHECKING PROGRAM FOR FIGURE 12.5.
3 REM : USE THE C-64 USER PORT AT ADDRESS 56577
5 PRINTCHR$(147)
10 PRINT"CHECKING RESISTANCE"
20 IF PEEK(56577)=32 THEN GOTO 100
30 IF PEEK(56577)=64 THEN GOTO 150
40 IF PEEK(56577)=128 THEN GOTO 200
50 PRINT"CHECK TEST CIRCUITS"
60 END
100 PRINT"RESISTANCE CHECK HIGH":END
150 PRINT"RESISTANCE CHECK LOW":END
200 PRINT"RESISTANCE CHECK OK":END
READY.
```

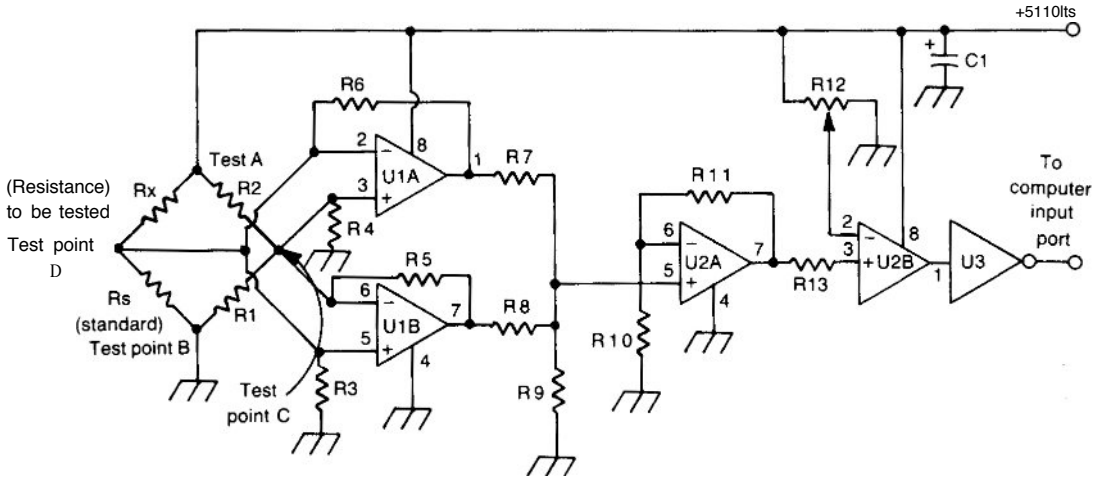
Program 12-5. A resistance-checking program for Fig. 12-5.

bridge is, how it works, and continue on from there.

Bridge circuits usually function with very low currents and voltages while the C-64 computer requires the use of logic ones and zeros at a level of five volts and zero volts (respectively) which does not make too good of a match up. So, we will have to build some type of amplifier circuit to increase the voltages and currents and then build a detector circuit to tell the voltages and currents and then build a detector circuit to tell the computer when the bridge circuit is in a *null* condition by generating a logic one or zero to indicate the given condition. All of the test circuits that will be described in this chapter will have one common function, which is to indicate when the bridge circuit is in the required null condition. The circuits will not tell you if you are above or below the null point, but only that you are in or out of the null condition of the bridge. You will be able to adjust the circuits to indicate different levels of the null point so you can detect a specific null point plus or minus a given percentage. When you use these bridge circuits in a high-speed test system with a C-64 computer, you will be able to secure highly accurate and expedient test measurements. The detection circuits are fast enough that, if you include a null meter in your system, the null meter's needle will not have time to indicate the null condition because the detection circuits will have already told the C-64 computer that the null condition is okay and the computer will have started the next test function. We will only discuss a few typical bridge circuit applications, but it will be very easy for you to expand these circuit's applications with a little technical understanding and ingenuity.

The bridge circuit that is shown in Fig. 12-6 is used to measure dc resistance values. The actual bridge circuit is made up of resistances Rl, R2, Rx' and Rs' which are connected in the form of a rectangle. Test point A is where the +V voltage is applied to the circuit, and test point B is where the bridge is connected to the ground or common circuit point. Resistors Rl and R2 are chosen so their resistances are equal or as close to equal as you can obtain. Resistor Rs is the *standard* resistor which is the resistance that you are comparing to Rx. Rx is the resistor under test, which is an unknown quantity. Since resistor Rl and R2 are equal, the voltage at test point C will be exactly one-half of the +V voltage. If the resistance of the unknown resistance Rx is equal to the resistance of the standard resistor Rs' the voltage at test point D will be equal to one-half of the +V voltage. Now if both voltages at test point C and D are equal, the outputs of both op amps (UIA and UIB) will be zero, which will indicate that the bridge circuit is in a null condition. The null condition will tell you that Rx and Rs are equal resistances.

When the Rx and Rs resistances are not equal, the voltage at test point D will no longer be one-



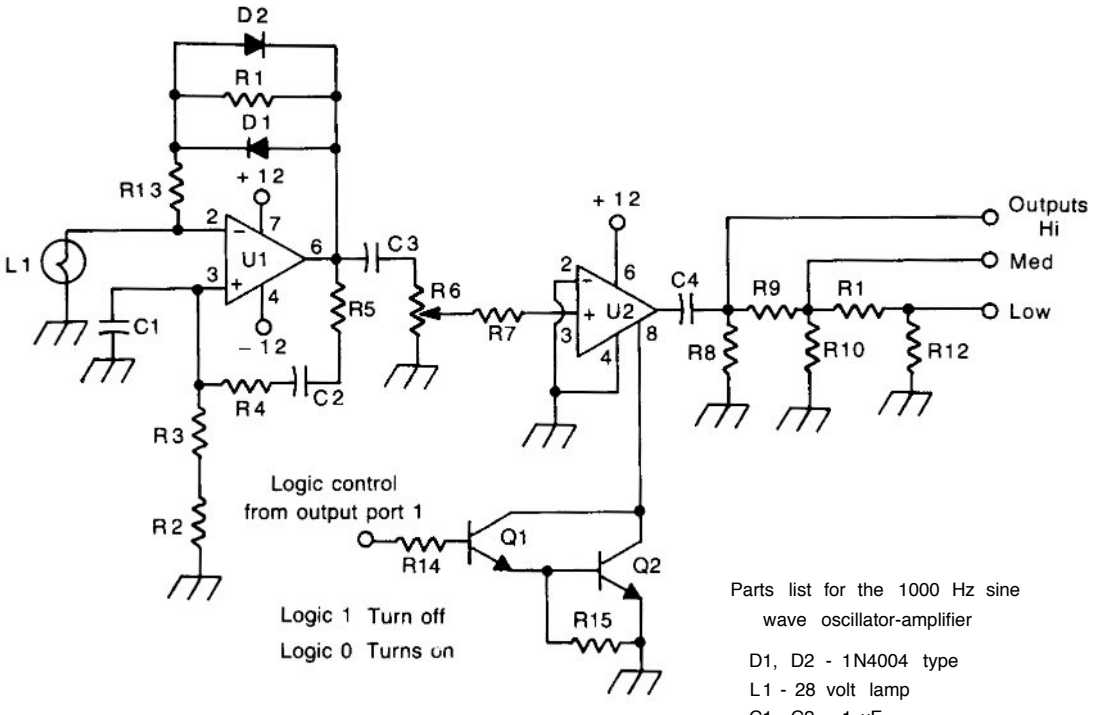
Parts List for the dc bridge amplifier and detector circuit

Rx • Resistance to be tested
Rs • Standard resistance value
R1, R2, R13 ·1 k resistors (matched as close as possible)
R3, R4, R5, R6. R11 . 100 k resistors
R7, R8 . 4.7 k resistor
R9 • 1.5 k reSistor
R10· 10 k resistor
R12 • 5 k trim pot
U1, U2 - LM 358 op-amp
U3 • 4584 CMOS
C1 ·470 μF

Fig. 12-6. The dc bridge-amplifier circuit.

half of the +V voltage because of the different voltage drops across the Rx and Rs resistances. The voltage difference that will exist between test points C and D will be detected by the op-amps and one of them will increase its output voltage. The increased output voltage will be amplified by op amp U2A and applied to the noninverting input of op amp U2B, which is a voltage detector circuit. If the voltage at pin 3 of op amp U2B is higher than the voltage at pin 2, the output of U2B will go high. If the voltage at pin 3 is lower than pin 2, the op amp's output will go low. U3 is a 4584 CMOS Schmitt trigger inverter, which will take out any noise or slow rise and fall times from the op-amp circuits and permit easy connection to the input port of the computer.

In a nutshell, the operation of the circuit can be described as generating a logic one if the bridge circuit is in a null condition or generating a logic zero if the bridge circuit is not in a null condition. Trim pot R12 is used to control the voltage-detector circuit's trip point. By setting the trip point low, you can detect a very sharp null point, and by setting the trip point high, you will detect a wide null point. The trip-point setting can be used to control a given amount of tolerance on each side of the standard resistance, such as testing for the standard resistance value plus or minus 10 percent.



Parts list for the 1000 Hz sine wave oscillator-amplifier

D1, D2 - 1N4004 type
L1 - 28 volt lamp
C1, C2 - 1 μF
C3 - .50 μF
C4 -220 μF
R1, R13, - 100 ohms
R2, R3, R4, R5 - 680 ohms
R6 - 10 k trim pot
R7, R15 - 1 k resistor
R8, R9, R10, R11, R12 - 47 ohms
R14 - 6.8 k resistor
U1 - μA 741 op amp
U2 - LM 388 audio amplifier
01, 02 - 2N2222

Fig. 12-7. The 1000 Hz sine-wave circuit.

AC BRIDGE CIRCUITS AND AC SINE-WAVE SOURCES

Ac bridge circuits are much more complicated than dc bridge circuits because you are dealing with many more variables than you are with dc bridge circuits. The variables that are in your ac bridge circuits do not really cause a lot of trouble until your ac frequencies go into the high audio frequency range and on into the rf frequency and on into the rf frequency range. The ac bridge circuits that are described in this chapter can be used at frequencies between 60 and 1500 Hz quite easily. However, if you try and measure high inductances or low capacitance values, the situation will become a bit tricky. The best instructor on bridge circuits is experience, so, get out the experimenter's boards and go to it.

In order to experiment with an ac bridge cir-

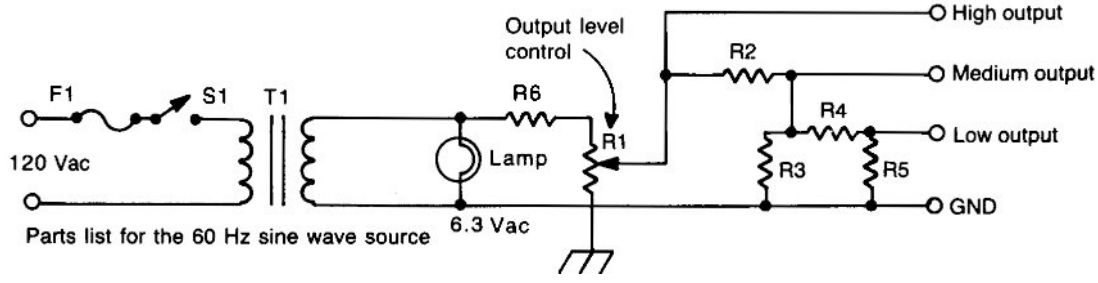
cuit, you will need a sine-wave generator. There are three ways to secure a sine-wave signal. Way number one is to go out and buy a sine-wave generator. Way number two is to build a sine-wave generator such as the one shown in Fig. 12-7.

Way number three is to simply use the 60 Hz sine wave frequency that is available from your friendly power company as shown in Fig. 12-8. If you are going to use an ac bridge circuit, you will need an ac signal from some source, so you might as well decide which method you intend to use.

Figure 12-8 is quite simple and does a fine job of giving you a 60-Hz sine wave which is variable between about 6.3 volts and zero volts depending upon the load that the bridge circuits place on it. Do not use this sine-wave signal source if the ac or dc resistance of the total bridge circuit is less than 50 ohms because the high output current requirements can damage the output control pot (Rl). This means that you had better keep a watchful eye on your circuit if you try and measure high capacitance values or low inductance values.

Figure 12-7 shows a 1,000-Hz sine-wave generator that can be built on a very limited budget. The nice point about this sine-wave generator is the fact that it can be turned off and on by the computer by sending a logic one or zero to transistor Q1. This type of off and on operation is advantageous in situations where you must do both ac and dc measurements on the same device. There is nothing critical about this circuit. You can change the operating frequency of the circuit by changing the values of capacitors Cl and C2 or resistors R2, R3, R4, and R5. If at any time the waveform becomes distorted, try adjusting the values of resistors Rl and R13 one way or another but not too much.

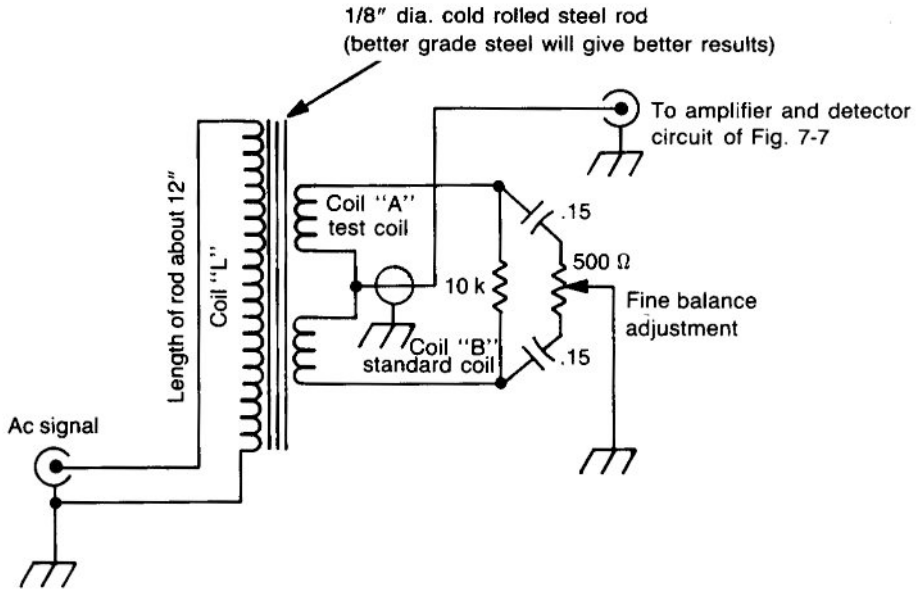
The bridge circuit that is presented in Fig. 12-9 can be used to compare inductor coils. Most of the time a coil will have to be manufactured to a certain inductance specification (measured in henrys). But, when you wind this coil, you will find that you can have a great difference in coil turns and still have an adequate amount of inductance. Naturally you will not want to wind any more copper wire on the coil than is needed in order to keep the price of the coil low. This fact means that you will need some method of testing the number of turns on the coil to insure that a given inductance is being produced with the least amount of copper in the coil. The bridge circuit in Fig. 12-9 will give you the ability to compare coil A with coil B and that the number of turns on coil A is within .005% of the number of turns on coil B. Of course, the tighter you try to measure the coil turns the more difficult it will become. You will just have to keep ex-



Parts list for the 60 Hz sine wave source

F1 - *1/2* amp fuse
S1 - 3 amp ac switch
T1 - Radio Shack 273-1384
Lamp - 6.3 volt lamp
R6 - 49 ohms, 1 watt resistor
R1 - 500 ohms, 2 watt control pot
R2, R3, R4, R5 - 68 ohm, *1/2* watt resistor

Fig. 12-8. The 60 Hz sine-wave source.



Note: Coils A and B must be placed on the transformer rod equal distance on each side of the rod center point. The best null will be secured when both coils are together and centered on the transformer rod.

Fig. 12-9. The inductor-comparison bridge circuit.

perimenting with different fixturing, probes, and bridge-circuit values until you find the combination that will work. You can start out by winding coil "L" over the entire length of the 12-inch steel rod. Place the rod with coil "L" through the coil A and B cores and place coils A and B together in the center of the rod. You should now be able to null out the bridge circuit by adjusting the 500 ohm fine-balance control.

The bridge circuit in Fig. 12-10 can be used to compare the capacitance value of one capacitor to a standard capacitor. The circuit is not as critical as the inductance comparison circuit, so, you should not have any trouble getting this circuit working. The bridge circuit is easier to experiment with because you can easily buy capacitors of the same value.

AC BRIDGE CIRCUIT AMPLIFIER AND LEVEL DETECTOR

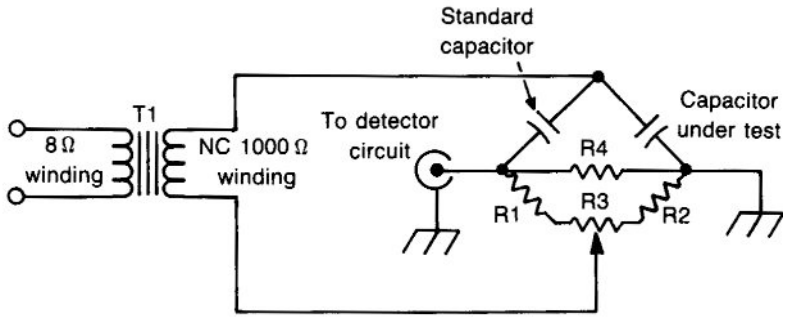
Once you have built the ac bridge circuit and have it working, you must find a method to detect the null point of the bridge circuit and then convert that null point into a logic signal that can be used by a computer. The ac bridge circuit interfacing system is not as simple as the dc bridge system because the ac circuits are not linear functions, and as the ac sine-wave driving signal increases in frequency, the stray capacitances and inductances of the circuit wiring can play tricks on you if you are not careful.

To start off, you will have to build an ac amplifier circuit to amplify the ac output signal from the bridge circuit so you can observe the null point and then you will need an ac detector circuit so you

can tell the computer when the null has been found. The ac output signal from the bridge circuit will be very low (less than 200 microvolts) at the null point, so the amplifiers will have to have a very low internal noise factor to accurately yield a good amplified nUll-point signal. A good low-noise op-amp circuit can be hard to build from low priced op amps. The solution is to build a low-noise ac amplifier from, dual gate MOSFETs.

A good low-noise amplifier is shown in Fig. 12-11. This amplifier has a complete set of control pots so you can adjust every circuit amplification factor required for your interface system. You can adjust this circuit to be a low-gain amplifier or you can adjust it to be a very high gain compression amplifier is the diode combination of D3 and D4. After you have gained a little ac bridge experience, you will find that you will need a very high gain amplifier if you are going to be attempting any high quality detection work around the bridge null point. For example, let's say that you are attempting to detect a .01 μF capacitor at a tolerance of plus or minus a few percent. You will observe that it is easy to find the .01 μF capacitor at 1% but the null points at the 2, 3, 4, and 5 percentage point are all scrunched together because of the high gain of the amplifier. The diodes D3 and D4 will give you a dead band in the area of the null point which will give you the ability to discriminate between the 2, 3,4 and 5 percentage tolerance points a little easier. This ability to discriminate between the 2, 3, 4 and 5 percentage points is really what makes this bridge-circuit interface system usable in the go/no-go type of test circuit work.

After the amplifier in Fig. 12-11 is working, you can build the detection circuit in Fig. 12-12. This circuit is a straightforward voltage detector circuit using op-amp UI as a noninverting buffer circuit, and op-amp U2 as a voltage detector circuit. The voltage doubler circuit of Dl and D2 develops a dc voltage level will be proportional to the ac signal at the input of the detection circuit. When the bridge circuit is in a null condition, there will be a very low ac input signal which will yield a low dc voltage level. By adjusting trimpot R2, you will be able to detect about any voltage level between 0.5 volts to 8 volts which will give you the ability to detect various levels of null points. Trimpot R3 is used to control the time constant of the rectifier filter circuit. The output of the voltage detector circuit goes to the three CMOS circuits and then on to your computer.



Parts list for the capacitor comparison bridge
T1 - Radio Shack 273-1380
R4 - 47 k resistor
R3 - 100 ohm ten turn trim pot
R1, R2 - 1 k resistors
Note: Control pot R3 should be adjusted for the bridge null condition with two matched capacitors of the same value.

Fig. 12-10. The capacitor-comparison bridge circuit.

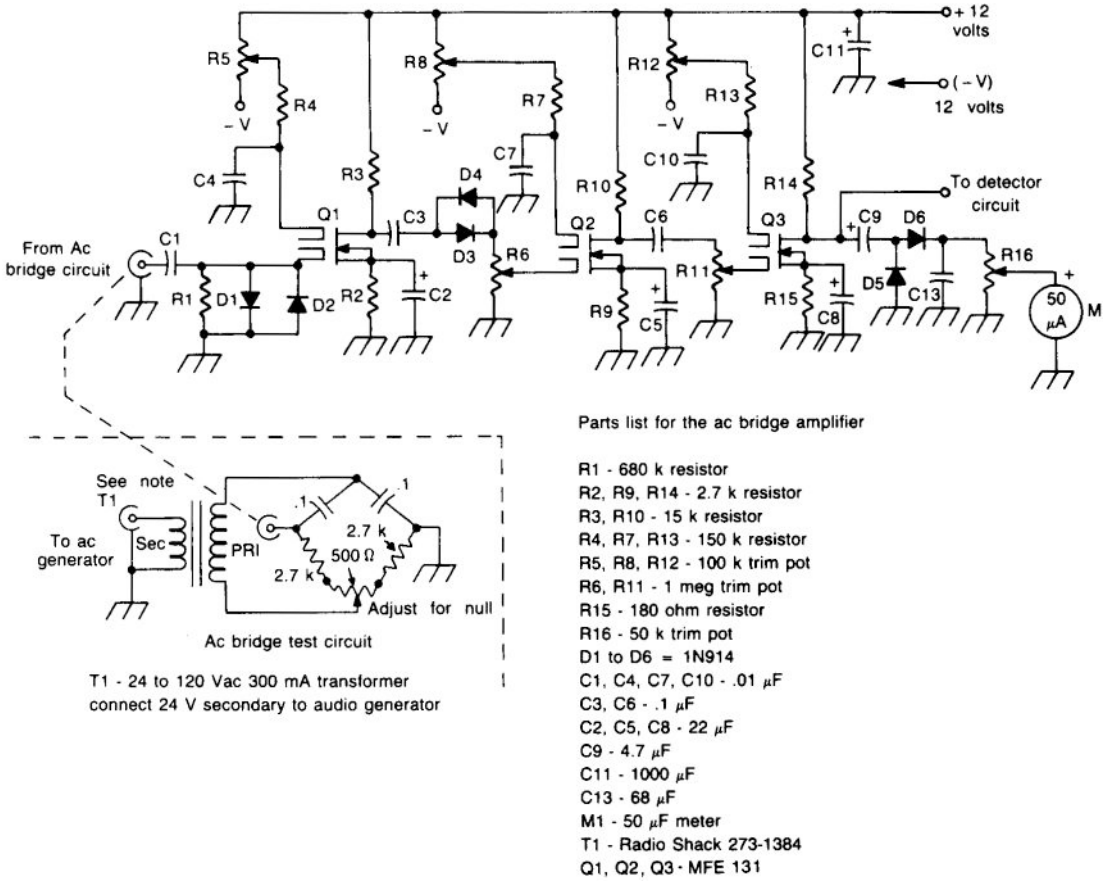
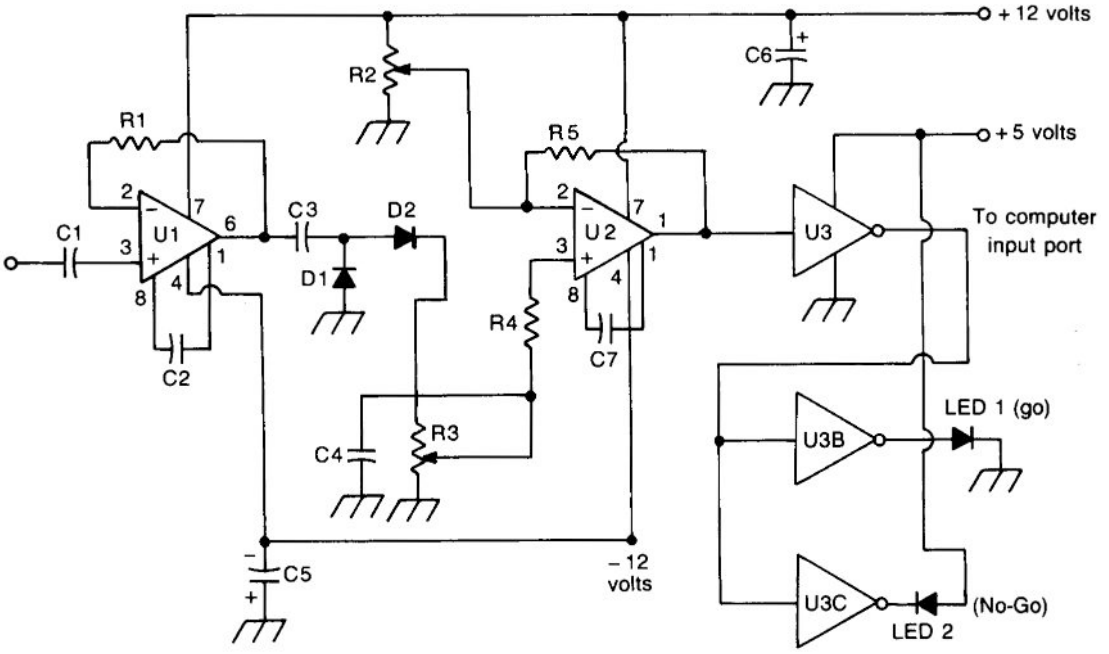


Fig. 12-11. **The** low-noise ac bridge-amplifier.

It will take you a little time to learn how to use the ac bridge interfacing circuits, but after you have solved the start-up problems, you will find that you can do some really outstanding high speed ac bridge testing using this method.

CONCLUSION

In this chapter, we have mainly worked with only simple bridge circuits, but even the more advanced circuits must be approached in a similar manner because only the complexity of the bridge circuits will differ. You will discover that when you try to interface to a commercial bridge system that you will only be looking for a logic one or zero, which will be brought out to one of the application terminals on the commercially manufactured bridge.



Parts list for the bridge amplifier level detector

R1, R4 - 1 k resistor
R2 - 5k trim pot
R3 - 50k trim pot
AS - 1 meg resistor
C1 - .05 μF
C2, C7 - 150 pF
C3 - 10 μF
C4 -.5 μF
C5, C6 - 220 μF
U1 - CA-3140 CMOS op amp
U2 - CA-3130 CMOS op amp
Note: If CMOS op amps can not be secured, try using two μA 741 op amps.
U3 - 4049 CMOS hex inverter
LED-1 green LED
LED-2 red LED
All capacitors are rated at 16 volts or higher.
All resistors are 1/4 watt or bigger.

Fig. 12-12. The ac bridge-amplifier level-detector circuit.

Chapter 13

The 6502 Instruction Set— An Alphabetical Presentation

This chapter presents a complete description of the 6502 instruction set from the *R6500 Microcomputer Systems Programming Manual*[1] by Rockwell International. The difference between this chapter and the programming manual is that, in this chapter, all of the instruction-set data and descriptions are presented in alphabetical order. This type of presentation will help the part-time programmer to use 6502 machine language. Table 13-1 presents the notations which are used in this chapter.

1. R 6500 Microcomputing Systems Programming Manual, Rockwell International, 1979. Data from this publication is reproduced by permission of Rockwell International.

Table 13-1. These Notations are used in the 6502 Op-Code Presentations in this Chapter.

The following notation applies to this chapter:

A	Accumulator	⊻	Logical Exclusive Or
X, Y	Index Registers	↑	Transfer from Stack
M	Memory	↓	Transfer to Stack
P	Processor Status Register	→	Transfer to
S	Stack Pointer	←	Transfer to
✓	Change	V	Logical OR
–	No Change	PC	Program Counter
+	Add	PCH	Program Counter High
∧	Logical AND	PCL	Program Counter Low
-	Subtract	OPER	Operand
		#	Immediate Addressing Mode

Table 13-2. **ADC.**

ADC *Add memory to accumulator with carry* **ADC**

Operation: A + M + C → A, C

N Z C I D V
✓ ✓ ✓ – – ✓

(Ref: 2.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ADC # Oper	69	2	2
Zero Page	ADC Oper	65	2	3
Zero Page, X	ADC Oper, X	75	2	4
Absolute	ADC Oper	6D	3	4
Absolute, X	ADC Oper, X	7D	3	4*
Absolute, Y	ADC Oper, Y	79	3	4*
(Indirect, X)	ADC (Oper, X)	61	2	6
(Indirect), Y	ADC (Oper), Y	71	2	5*

* Add 1 if page boundary is crossed.

This instruction adds the value of memory and carry from the previous operation to the value of the accumulator and stores the result in the accumulator (see Table 13-2).

This instruction affects the accumulator; sets the carry flag when the sum of a binary add exceeds 255 or when the sum of a decimal add exceeds 99, otherwise carry is reset. The overflow flag is set when the sign or bit 7 is changed due to the result exceeding +127 or - 128, otherwise overflow is reset. The negative flag is set if the accumulator result contains bit 7 on, otherwise the negative flag is reset. The zero flag is set if the accumulator result is 0, otherwise the zero flag is reset.

It is a "Group One" instruction and has the following addressing modes: Immediate; Absolute; Zero Page; Absolute, X; Absolute,Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-3. AND.

AND *"AND" memory with accumulator* **AND**

Logical AND to the accumulator

Operation: A ∧ M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	AND # Oper	29	2	2
Zero Page	AND Oper	25	2	3
Zero Page, X	AND Oper, X	35	2	4
Absolute	AND Oper	2D	3	4
Absolute, X	AND Oper, X	3D	3	4*
Absolute, Y	AND Oper, Y	39	3	4*
(Indirect, X)	AND (Oper, X)	21	2	6
(Indirect), Y	AND (Oper), Y	31	2	5*

* Add 1 if page boundary is crossed.

The AND instructions transfer the accumulator and memory to the adder which performs a bit-by-bit AND operation and stores the result back in the accumulator (see Table 13-3).

This instruction affects the accumulator; sets the zero flag if the result in the accumulator is 0, otherwise resets the zero flag; sets the negative flag if the result in the accumulator has bit 7 on, otherwise resets the negative flag.

AND is a "Group One" instruction having addressing modes of Immediate; Absolute; Zero Page; Absolute, X; Absolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-4. ASL.

ASL — ASL *Shift Left One Bit (Memory or Accumulator)* — **ASL**

Operation: C ← [7|6|5|4|3|2|1|0] ← 0

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 10.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ASL A	0A	1	2
Zero Page	ASL Oper	06	2	5
Zero Page, X	ASL Oper, X	16	2	6
Absolute	ASL Oper	0E	3	6
Absolute, X	ASL Oper, X	1E	3	7

The shift left instruction shifts either the accumulator or the address memory location 1 bit to the left, with the bit 0 always being set to 0 and the bit 7 output always being contained in the carry flag. ASL either shifts the accumulator left 1 bit or is a read/modify/write instruction that affects only memory (see Table 13-4).

The instruction does not affect the overflow bit, sets N equal to the result bit 7 (bit 6 in the input), sets Z flag if the result is equal to 0, otherwise resets Z and stores the input bit 7 in the carry flag.

ASL is a read/modify/write instruction and has the following addressing modes: Accumulator; Zero Page; Zero Page, X; Absolute; Absolute, X.

Table 13-5. BCC.

BCC — BCC *Branch on Carry Clear* — **BCC**

Operation: Branch on C = 0

N Z C I D V
– – – – – –

(Ref: 4.1.2.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCC Oper	90	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

This instruction tests the state of the carry bit and takes a conditional branch if the carry bit is reset (see Table 3-5).

It affects no flags or registers other than the program counter and then only if the C flag is not on.

The addressing mode is Relative.

Table 13-6. BCS.

BCS *BCS Branch on carry set* **BCS**

Operation: Branch on C = 1

N Z C I D V

\- - - - - -

(Ref: 4.1.2.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCS Oper	BØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

This instruction takes the conditIOnal branch if the carry flag is on (see Table 13-6).

BCS does not affect any of the flags or registers except for the program counter and only then if the carry flag is on.

The addressing mode is Relative.

Table 13-7. BEQ.

BEQ **BEQ** *BEQ Branch on result zero*

Operation: Branch on Z = 1

N Z C I D V

\- - - - - -

(Ref: 4.1.2.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BEQ Oper	FØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

This instruction could also be called "Branch on Equal." It takes a conditional branch whenever the Z flag is on or the previous result is equal to 0 (see Table 13-7).

BEQ does not affect any of the flags or registers other than the program counter and only then when the Z flag is set.

The addressing mode is Relative.

Table 13-8. **BIT.**

BIT **BIT** *Test bits in memory with accumulator* **BIT**

Operation: A ∧ M, $M_7 \rightarrow N$, $M_6 \rightarrow V$

Bit 6 and 7 are transferred to the status register. N Z C I D V

If the result of A∧M is zero then Z = 1, otherwise M_7 ✓ – – – M_6

Z = Ø

(Ref: 4.2.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

This instruction performs an AND between a memory location and the accumulator but does not store the result of the AND into the accumulator (see Table 13-8).

The symbolic notation is M ∧ A.

The bit instruction affects the N flag with N being set to the value of bit 7 of the memory being tested, the V flag with V being set equal to bit 6 of the memory being tested and Z being set by the result of the AND operation between the accumulator and the memory if the result is Zero, Z is reset otherwise. It does not affect the accumulator.

The addressing modes are Zero Page and Absolute.

The BIT instruction actually combines two instructions from the PDP-11 and MC6800, that of TST (Test Memory) and (BIT Test). This, like the compare test, allows the examination of an individual bit without disturbing the value in the accumulator.

Table 13-9. **BMI.**

BMI **BMI** *Branch on result minus* **BMI**

Operation: Branch on N = 1

N Z C I D V

– – – – – –

(Ref: 4.1.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BMI Oper	3Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

This instruction takes the conditional branch if the N bit is set (see Table 13-9).

BMI does not affect any of the flags or any other part of the machine other than the program counter and then only if the N bit is on.

The mode of addressing for BMI is Relative.

Table 13-10. BNE.

BNE BNE *Branch on result not zero* **BNE**

Operation: Branch on Z = 0

N Z C I D V
– – – – – –

(Ref: 4.1.2.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BNE Oper	DØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

This instruction could also be called "Branch on Not Equal." It tests the Z flag and takes the conditional branch if the Z flag is not on, indicating that the previous result was not zero (see Table 13-10).

BNE does not affect any of the flags or registers other than the program counter and only then if the Z flag is reset.

The addressing mode is Relative.

Table 13-11. BPL.

BPL BPL *Branch on result plus* **BPL**

Operation: Branch on N = Ø

N Z C I D V
– – – – – –

(Ref: 4.1.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BPL Oper	1Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

This instruction is the complementary branch to branch on result minus (see Table 13-11). It is a conditional branch which takes the branch when the N bit is reset (0). BPL is used to test if the previous result bit 7 was off (0) and branch on result minus is used to determine if the previous result was minus or bit 7 was on (1).

The instruction affects no flags or other registers other than the P counter and only affects the P counter when the N bit is reset.

The addressing mode is Relative.

Table 13-12. BRK.

BRK — **BRK** *Force Break* — **BRK**

Operation: Forced Interrupt PC + 2 ↓ P ↓

N Z C I D V
– – – 1 – –

(Ref: 9.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	BRK	00	1	7

1. A BRK command cannot be masked by setting I.

The break command causes the microprocessor to go through an interrupt sequence under program control (see Table 13-12). This means that the program counter of the second byte after the **BRK** is automatically stored of the stack along with the processor status at the beginning of the break instruction. The microprocessor then transfers control to the interrupt vector.

Other than changing the program counter, the break instruction changes no values in either the registers or the flags.

The **BRK** is a single-byte instruction and its addressing mode is Implied.

Table 13-13. BVe.

BVC — **BVC** *Branch on overflow clear* — **BVC**

Operation: Branch on V = 0

N Z C I D V
– – – – – –

(Ref: 4.1.2.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVC Oper	50	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

This instruction tests the status of the V flag and takes the conditional branch if the flag is not set (see Table 13-13).

BVC does not affect any of the flags and registers other than the program counter and only when the overflow flag is reset.

The addressing mode is Relative.

Table 13-14. BVS.

BVS — **BVS** *Branch on overflow set* — **BVS**

Operation: Branch on V = 1

N Z C I D V
– – – – – –

(Ref: 4.1.2.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVS Oper	7Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

This instruction tests the V flag and takes the conditional ranch if V is on (see Table 13-14).

BVS does not affect any flags or registers other than the program counter and only when the overflow flag is set.

The addressing mode is Relative.

Table 13-15. CLC.

CLC — **CLC** *Clear carry flag* — **CLC**

Operation: Ø → C

N Z C I D V
– – Ø – – –

(Ref: 3.0.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLC	18	1	2

This instruction initializes the carry flag to a 0. This operation should normally precede an ADC loop (see Table 13-15). It is also useful when used with a ROL instruction to clear a bit in memory.

This instruction affects no registers in the microprocessor and no flags other than the carry flag which is reset.

CLC is a single-byte instruction and its addressing mode is Implied.

Table 13-16. CLD.

CLD **CLI** *Clear interrupt disable bit* **CLD**

Operation: Ø → D

N Z C I D V
– – – – Ø –

(Ref: 3.3.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

This instruction sets the decimal mode flag to a O. This causes all subsequent ADC and SBC instructions to operate as simple binary operations (see Table 13-16).

CLD affects no registers in the microprocessor and no flags other than the decimal mode flag which is set to a 0.

Table 13-17. CLI.

CLI **CLI** *Clear interrupt disable bit* **CLI**

Operation: Ø → I

N Z C I D V
– – – Ø – –

(Ref: 3.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2

This instruction initializes the interrupt disable to a O. This allows the microprocessor to receive interrupts (see Table 13-17).

It affects no registers in the microprocessor and no flags other than the interrupt disable which is cleared.

CLI is a single-byte instruction and is addressing mode is Implied.

Table 13-18. CLV.

CLV — CLV *Clear overflow flag* — **CLV**

Operation: Ø → V

N Z C I D V
– – – – – Ø

(Ref: 3.6.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

This instruction clears the overflow flag to a 0 (see Table 13-18). This command is used in conjunction with the set overflow pin which can change the state of the overflow flag with an external signal.

CLV affects no registers in the microprocessor and no flags other than the overflow flag which is set to a 0.

Table 13-19. CMP.

CMP — CMP *Compare memory and accumulator* — **CMP**

Operation: A - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 4.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CMP #Oper	C9	2	2
Zero Page	CMP Oper	C5	2	3
Zero Page, X	CMP Oper, X	D5	2	4
Absolute	CMP Oper	CD	3	4
Absolute, X	CMP Oper, X	DD	3	4*
Absolute, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper), Y	D1	2	5*

* Add 1 if page boundary is crossed.

This instruction subtracts the contents of memory from the contents of the accumulator (see Table 13-19).

The use of a CMP affects the following flags: Z flag is set on an equal comparison, reset otherwise; the N flag is set or reset by the result bit 7, the carry flag is set when the value in memory is less than or equal to the accumulator, reset when it is greater than the accumulator. The accumulator is not affected.

It is a "Group One" instruction and therefore has as its addressing modes: Immediate; Zero Page; Zero Page, X; Absolute; Absolute, X; Absolute, Y; (Indirect, X); (Indirect), Y.

Table 13-20. CPX.

CPX — CPX *Compare Memory and Index X* — **CPX**

Operation: X - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 7.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPX #Oper	EØ	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

This instruction subtracts the value of the addressed memory location from the content of index register X using the adder but does not store the result; therefore, its only use is to set the N, Z and C flags to allow for comparison between the index register X and the value in memory (see Table 13-20).

The CPX instruction does not affect any register in the machine; it also does not affect the overflow flag. It causes the carry to be set on if the absolute value of the index register X is equal to or greater than the data from memory. If the value of the memory is greater than the content of the index register X, carry is reset. If the results of the subtraction contain a bit 7, then the N flag is set, if not, it is reset. If the value in memory is equal to the value in index register X, the Z flag is set, otherwise it is reset.

The addressing modes for CPX are Immediate. Absolute and Zero Page.

Table 13-21. CPY.

CPY — CPY *Compare memory and index Y* — **CPY**

Operation: Y - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 7.9)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPY #Oper	CØ	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

This instruction performs a two's complement subtraction between the index register Y and the specified memory location (see Table 13-21). The results of the subtraction are not stored anywhere. The instruction is strictly used to set the flags.

CPY affects no registers in the microprocessor and also does not affect the overflow flag. If the value in the index register is Y equal to or greater than the value in the memory, the carry flag will be set, otherwise it will be cleared. If the results of the subtraction contain bit 7 on the N bit will be set, otherwise it will be cleared. If the value in the index register Y and the value in the memory are equal, the zero flag will be set, otherwise it will be cleared.

The addressing modes for CPY are Immediate, Absolute and Zero Page.

Table 13-22. DEC.

DEC DEC *Decrement memory by one* **DEC**

Operation: M - 1 → M

N Z C I D V
✓ ✓ – – – –

(Ref: 10.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

This instruction subtracts **1,** in two's complement, from the contents of the addressed memory location (see Table 13-22).

The decrement instruction does not affect any internal register in the microprocessor. **It** does not affect the carry or overflow flags. **If** bit 7 is on as a result of the decrement, then the N flag is set, otherwise it is reset. **If** the result of the decrement is 0, the Z flag is set, otherwise it is reset.

Table 13-23. DEX.

DEX DEX *Decrement index X by one* **DEX**

Operation: X - 1 → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

This instruction subtracts one from the current value of theindex register X and stores the result in the index register X (see Table 13-23).

DEX does not affect the carry or overflow flag, it sets the N flag if it has bit 7 on as a result of the decrement, otherwise it resets the N flag; sets the Z flag if X is a 0 as a result of the decrement, otherwise it resets the Z flag.

DEX is a single-byte instruction, the addressing mode is Implied.

NOTE: Decrement of the index registers is the most convenient method of using the index registers as a counter, in that the decrement involves setting the value N on as a result of having passed through 0 and sets Z on when the results of the decrement are 0.

Table 13-24. DEY.

DEY — **DEY** *Decrement index Y by one* — **DEY**

Operation: Y - 1 → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

This instruction subtracts one from the current value in the index register Y and stores the result into the index register Y (see Table 13-24). The result does not affect or consider carry so that the value in the index register Y is decremented to 0 and then through 0 to FF.

Decrement Y does not affect the carry or overflow flags; if the Y register contains bit 7 on as a result of the decrement the N flag is set, otherwise the N flag is reset. If the Y register is 0 as a result of the decrement, the Z flag is set otherwise the Z flag is reset. This instruction only affects the index register Y.

DEY is a single-byte instruction and the addressing mode is Implied.

Table 13-25. EOR.

EOR — **EOR** *"Exclusive-Or" memory with accumulator* — **EOR**

Operation: A ⊻ M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	EOR #Oper	49	2	2
Zero Page	EOR Oper	45	2	3
Zero Page, X	EOR Oper, X	55	2	4
Absolute	EOR Oper	4D	3	4
Absolute, X	EOR Oper, X	5D	3	4*
Absolute, Y	EOR Oper, Y	59	3	4*
(Indirect, X)	EOR (Oper, X)	41	2	6
(Indirect),Y	EOR (Oper), Y	51	2	5*

* Add 1 if page boundary is crossed.

The EOR instruction transfers the memory and the accumulator to the adder which performs a binary "EXCLUSIVE OR" on a bit-by-bit basis and stores the result in the accumulator (Table 13-25).

This instruction affects the accumulator; sets the zero flag if the result in the accumulator is 0, otherwise resets the zero flag; sets the negative flag if the result in the accumulator has bit 7 on, otherwise resets the negative flag.

EOR is a "Group One" instruction having addressing modes of Immediate; Absolute; Zero Page; Absolute, X; Absolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-26. INC.

INC — INC *Increment memory by one* — **INC**

Operation: M + 1 → M

N Z C I D V
✓ ✓ – – – –

(Ref: 10.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	INC Oper	E6	2	5
Zero Page, X	INC Oper, X	F6	2	6
Absolute	INC Oper	EE	3	6
Absolute, X	INC Oper, X	FE	3	7

This instruction adds 1 to the contents of the addressed memory location (see Table 13-26).

The increment memory instruction does not affect any internal registers and does not affect the carry or overflow flags. If bit 7 is on as the result of the increment, N is set, otherwise it is reset; if the increment causes the result to become 0, the Z flag is set on, otherwise it is reset.

The addressing modes for increment are: Zero Page; Zero Page, X; Absolute; Absolute, X.

Table 13-27. INX.

INX — INX *Increment Index X by one* — **INX**

Operation: X + 1 → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

Increment X adds 1 to the current value of the X register (see Table 13-27). This is an 8-bit increment which does not affect the carry operation, therefore, if the value of X before the increment was FF, the resulting value is 00. NX does not affect the carry or overflowing flags; it sets the N flag if the flag that is the result of the increment has one in bit 7, otherwise resets N; sets the Z flag if the result of the increment is 0, otherwise it resets the Z flag. INX does not affect any other register other than the X register. INX is a single byte instruction and the only addressing mode is Implied.

Table 13-28. INY.

INY — INY *Increment Index Y by one* — **INY**

Operation: Y + 1 → Y

(Ref: 7.5)

N	Z	C	I	D	V
✓	✓	–	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INY	C8	1	2

INX is a single-byte instruction and the only addressing mode is Implied.

Increment Y increments or adds one to the current value in the Y register, storing the result in the Y register (see Table 13-28). As in the case of INX the primary application is to step through a set of values using the Y register. The INY does not affect the carry or overflow flags, sets the N flag if the result of the increment has a one in bit 7, otherwise resets N, sets Z if as a result of the increment the Y register is zero otherwise resets the Z flag. Increment Y is a single byte instruction and the only addressing mode is Implied.

Table 13-29. JMP.

JMP — JMP *Jump to new location* — **JMP**

Operation: (PC + 1) → PCL
(PC + 2) → PCH

(Ref: 4.0.2)
(Ref: 9.8.1)

N	Z	C	I	D	V
–	–	–	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

In this instruction,the data from the memory location located in the program sequence after the OP CODE is loaded into the low order byte of the program counter (PCL) and the data from the next memory location after that is loaded into the high order byte of the program counter (PCH) (see Table 13-29).

As stated earlier, the "()" means "contents of" a memory location. PC indicates the contents of the program counter at the time the OP CODE is fetched. Therefore (PC + 2) PCH reads, "the contents of the program counter two locations beyond the OP CODE fetch location are transferred to the new PC high order byte."

The addressing modes are Absolute and Absolute Indirect.

The *IMP* instruction affects no flags and only PCL and PCH.

JMP Indirect-for indirect jump.

This instruction establishes a new value for the program counter.

It affects only the program counter in the microprocessor and affects no flags in the status register.

IMP Indirect is a three-byte instruction.

In the *IMP* Indirect instruction, the second and third bytes of the instruction represent the indirect low and high bytes respectively of the memory location containing ADL. Once ADL is fetched, the program counter is incremented with the next memory location containing ADH.

Table 13-30. JSR.

JSR — JSR *Jump to new location saving return address* — **JSR**

Operation: PC + 2 ↓, (PC + 1) → PCL, (PC + 2) → PCH

N Z C I D V
– – – – – –

(Ref: 8.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JSR Oper	20	3	6

This instruction transfers control of the program counter to a subroutine location but leaves a return pointer on the stack to allow the user to return to perform the next instruction in the main program after the subroutine is complete (see Table 13-30). To accomplish this, JSR instruction stores the program counter address which points to the last byte of the jump instruction onto the stack using the stack pointer. The stack byte contains the program count high first, followed by program count low. The JSR then transfers the addresses following the jump instruction to the program counter low and the program counter high, thereby directing the program to begin at that new address.

The JSR instruction affects no flags, causes the stack pointer to be decremented by 2 and substitutes new values into the program counter low and the program counter high. The addressing mode for the JSR is always Absolute.

Table 13-31. LDA.

LDA — LDA *Load accumulator with memory* — **LDA**

Operation: M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.1.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDA #Oper	A9	2	2
Zero Page	LDA Oper	A5	2	3
Zero Page, X	LDA Oper, X	B5	2	4
Absolute	LDA Oper	AD	3	4
Absolute, X	LDA Oper, X	BD	3	4*
Absolute, Y	LDA Oper, Y	B9	3	4*
(Indirect, X)	LDA (Oper, X)	A1	2	6
(Indirect), Y	LDA (Oper), Y	B1	2	5*

* Add 1 if page boundary is crossed.

When instruction LDA is executed by the microprocessor, data is transferred from memory to the accumulator and stored in the accumulator (see Table 13-31).

Rather than continuing to give a word picture of the operation, introduced will be the symbolic representation M- A, where the arrow means "transfer to." Therefore the LDA instruction symbolic representation is read, "memory transferred to the accumulator."

LDA affects the contents of the accumulator, does not affect the carry or overflow flags; sets the zero flag if the accumulator is zero as a result of the LDA, otherwise resets the zero flag; sets the negative flag if bit 7 of the accumulator is a 1, otherwise resets the negative flag.

The addressing modes include Immediate; Absolute; Zero Page; Absolute, X; ABsolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-32. LDX.

LDX — **LDX** *Load index X with memory* — **LDX**

Operation: M → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDX # Oper	A2	2	2
Zero Page	LDX Oper	A6	2	3
Zero Page, Y	LDX Oper, Y	B6	2	4
Absolute	LDX Oper	AE	3	4
Absolute, Y	LDX Oper, Y	BE	3	4*

* Add 1 when page boundary is crossed.

Load the index register X from memory (see Table 13-32).

LDX does not affect the C or V flags; sets Z if the value loaded was zero, otherwise resets it; sets N if the value loaded in bit 7 is a 1; otherwise N is reset, and affects only the X register. The addressing modes for LDX are Immediate; Absolute; Zero Page; Absolute Indexed by Y; and Zero Page Indexed by Y.

Table 13-33. LDY.

LDY — **LDY** *Load index Y with memory* — **LDY**

Operation: M → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDY #Oper	A0	2	2
Zero Page	LDY Oper	A4	2	3
Zero Page, X	LDY Oper, X	B4	2	4
Absolute	LDY Oper	AC	3	4
Absolute, X	LDY Oper, X	BC	3	4*

* Add 1 when page boundary is crossed.

Load the index register Y from memory (see Table 13-33).

LDY does not affect the Cor V flags, sets the N flag if the value loaded in bit 7 is a 1, otherwise resets N, sets Z flag if the loaded value is zero otherwise resets Z and only affects the Y register. The addressing modes for load Yare Immediate; Absolute; Zero page; Zero Indexed by X, Absolute Indexed by X.

Table 13-34. LSR.

LSR — LSR *Shift right one bit (memory or accumulator)* — **LSR**

Operation: Ø → [7|6|5|4|3|2|1|0] → C

N	Z	C	I	D	V
Ø	✓	✓	–	–	–

(Ref: 10.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

This instruction shifts either the accumulator or a specified memory location 1 bit to the right, with the higher bit of the result always being set to 0, and the low bit which is shifted out of the field being stored in the carry flag (see Table 13-34).

The shift right instruction either affects the accumulator by shifting it right 1 or is a ready/modify/write instruction which changes a specified memory location but does not affect any internal registers. The shift right does not affect the overflow flag. The N flag is always reset. The Z flag is set if the result of the shift is 0 and reset otherwise. The carry is set equal to bit 0 of the input.

LSR is a read/write/modify instruction and has the following addressing modes: Accumulator; Zero Page; Zero Page, X; Absolute; Absolute, X.

Table 13-35. NOP.

NOP — NOP *No operation* — **NOP**

Operation: No Operation (2 cycles)

N	Z	C	I	D	V
–	–	–	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

Table 13-36. **ORA.**

ORA — **ORA** *"OR" memory with accumulator* — **ORA**

Operation: A V M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ORA #Oper	Ø9	2	2
Zero Page	ORA Oper	Ø5	2	3
Zero Page, X	ORA Oper, X	15	2	4
Absolute	ORA Oper	ØD	3	4
Absolute, X	ORA Oper, X	1D	3	4*
Absolute, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	Ø1	2	6
(Indirect), Y	ORA (Oper), Y	11	2	5*

* Add 1 on page crossing

The ORA instruction transfer the memory and the accumulator to the adder which performs a binary "OR" on a bit-by-bit basis and stores the result in the accumulator (see Table 13-36).

This instruction affects the accumulator; sets the zero flag if the result in the accumulator is 0, otherwise resets the zero flag; sets the negative flag if the result in the accumulator has bit 7 on, otherwise resets the negative flag. ORA is a "Group One" instruction. It has the addressing modes Immediate; Absolute; Zero Page; Absolute, X; Absolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-37. **PHA.**

PHA — **PHA** *Push accumulator on stack* — **PHA**

Operation: A ↓

N Z C I D V
– – – – – –

(Ref: 8.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

This instruction transfers the current value of the accumulator to the next location on the stack, automatically decrementing the stack to point to the next empty location (see Table 13-37).
Noted should be that the notation means push to the stack. > means pull from the stack.

The Push A instruction only affects the stack pointer register which is decremented by 1 as a result of the operation. It affects no flags.

PHA is a single-byte instruction and its addressing mode is Implied.

Table 13-38. PHP.

PHP **PHP** *Push processor status on stack* **PHP**

Operation: P↓

N Z C I D V
– – – – – –

(Ref: 8.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHP	Ø8	1	3

The following example shows the operations which occur during Push A instruction.

This instruction transfers the contents of the processor status register stack, as governed by the stack pointer (see Table 13-38).

The PHP instruction affects no registers or flags in the microprocessor.

PHP is a single-byte instruction and the addressing mode is Implied.

Table 13-39. PLA.

PLA **PLA** *Pull accumulator from stack* **PLA**

Operation: A ↑

N Z C I D V
✓ ✓ – – – –

(Ref: 8.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

This instruction adds 1 to the current value of the stack pointer and uses it to address the stack and loads the contents of the stack into the A register (see Table 13-39).

The PLA instruction does not affect the carry or overflow flags. It sets N if the bit 7 is on in accumulator A as a result of instructions, otherwise it is reset. If accumulator A is zero as a result of the PLA, then the Z flag is set, otherwise it is reset. The PLA instruction changes content of the accumulator A to the contents of the memory location at stack register plus 1 and also increments the stack register.

The PLA instruction is a single-byte instruction and the addressing mode is Implied.

Table 13-40. PLP.

PLP **PLP** *Pull processor status from stack* **PLP**

Operation: P ↑ N Z C I D V

From Stack

(Ref: 8.12)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

This instruction transfers the next value on the stack to the Processor Status register, thereby changing all of the flags and settings the mode switches to the values from the stack (see Table 13-40).

The PLP instruction affects no registers in the processor other than the status register. This instruction could affect all flags in the status register.

PLP is a single-byte instruction and the addressing mode is Implied.

Table 13-41. ROL.

ROL **ROL** *Rotate one bit left (memory or accumulator)* **ROL**

Operation: ↑ M or A [7 6 5 4 3 2 1 Ø] ← [C] ← N Z C I D V

✓ ✓ ✓ – – –

(Ref: 10.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

The rotate left instruction shifts either the accumulator or addressed memory left 1 bit, with the input carry being stored in bit 0 and with the input bit 7 being stored in the carry flags (see Table 13-41).

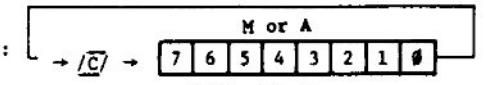
The ROL instruction either shifts the accumulator left 1 bit and stores the carry in accumulator bit 0 or does not affect the internal registers at all. The ROL instruction sets carry equal to the input bit 7, sets N equal to the input bit 6, sets the Z flag if the result of the rotate is 0, otherwise it resets Z and does not affect the overflow flag at all.

ROL is a read/modify/write instruction and it has the following addressing modes: Accumulator; Zero Page; Zero Page, X; Absolute; Absolute, X.

Table 13-42. ROR.

ROR **ROR** *Rotate one bit right (memory or accumulator)* **ROR**

Operation:



(Ref: 10.4)

N Z C I D V
✓ ✓ ✓ - - -

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROR A	6A	1	2
Zero Page	ROR Oper	66	2	5
Zero Page,X	ROR Oper,X	76	2	6
Absolute	ROR Oper	6E	3	6
Absolute,X	ROR Oper,X	7E	3	7

The rotate right instruction shifts either the accumulator or addressed memory right 1 bit with bit 0 shifted into the carry and carry shifted into bit 7 (see Table 13-42).

The ROR instruction either shifts the accumulator right 1 bit and stores the carry in accumulator bit 7 or does not affect the internal registers at all. The ROR instruction sets carry equal to input bit 0, sets N equal to the input carry and sets the Z flag if the result of the rotate is 0; otherwise it resets Z and does not affect the overflow flag at all.

ROR is a read/modify/write instruction and it has the following addressing modes: Accumulator; Zero Page; Absolute; Zero Page, X; Absolute, X.

Table 13-43. RTI.

RTI **RTI** *Return from interrupt* **RTI**

Operation: P↑ PC↑

N Z C I D V
From Stack

(Ref: 9.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTI	40	1	6

This instruction transfers from the stack into the microprocessor for the processor status and the program counter location for the instruction which was interrupted (see Table 13-43). By virtue of the interrupt having stored this data before executing the instruction and the fact that the RTI reinitializes the microprocessor to the same state as when it was interrupted, the combination of interrupt plus RTI allows truly reentrant coding.

The RTI instruction reinitializes all flags to the position to the point they were at the time the interrupt was taken and sets the program in the microprocessor.

RTI is a single-byte instruction and its addressing mode is Implied.

RTS — **RTS** *Return from subroutine* — **RTS**

Operation: PC↑, PC + 1 → PC

N Z C I D V
– – – – – –

(Ref: 8.2)

Table 13-44. RTS.

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTS	60	1	6

This instruction loads the program count low and program count high from the stack into the program counter and increments the program counter so that it points to the instruction following the JSR (see Table 13-44). The stack pointer is adjusted by incrementing it twice.

The RTS instruction does not affect any flags and affects only PCL and PCH. RTS is a single-byte instruction and its addressing mode is Implied.

Table 13-45. SBC.

SBC — **SBC** *Subtract memory from accumulator with borrow* — **SBC**

Operation: A – M – $\overline{C}$ → A

N Z C I D V
✓ ✓ ✓ – – ✓

Note: $\overline{C}$ = Borrow (Ref: 2.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	SBC #Oper	E9	2	2
Zero Page	SBC Oper	E5	2	3
Zero Page, X	SBC Oper, X	F5	2	4
Absolute	SBC Oper	ED	3	4
Absolute, X	SBC Oper, X	FD	3	4*
Absolute, Y	SBC Oper, Y	F9	3	4*
(Indirect, X)	SBC (Oper, X)	E1	2	6
(Indirect), Y	SBC (Oper), Y	F1	2	5*

* Add 1 when page boundary is crossed.

This instruction subtracts the value of memory and borrow from the value of the accumulator, using two's complement arithmetic, and stores the result in the accumulator (see Table 13-45). Borrow is defined as the carry flag complemented; therefore, a resultant carry flag indicates that a borrow has not occurred.

This instruction affects the accumulator. The carry flag is set if the result is greater than or equal to 0. The carry flag is reset when the result is less than 0, indicating a borrow. The overflow flag is set when the result exceeds +127 or -127, otherwise it is reset. The negative flag is set if the result in the accumulator has bit 7 on, otherwise it is reset. The Z flag is set if the result in the accumulator is 0, otherwise it is reset.

It is a "Group One" instruction. It has addressing modes Immediate; Absolute; Zero Page; Absolute, X; Absolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-46. SEC.

SEC — SEC *Set carry flag* — **SEC**

Operation: 1 → C

N Z C I D V
– – 1 – – –

(Ref: 3.0.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

This instruction initializes the carry flag to a 1 (see Table 13-46). This operation should normally procede a SBC loop. It is also useful when used with ROL instruction to initialized a bit in memory to a 1.

This instruction affects no registers in the microprocessor and no flags other than the carry flag which is set.

SEC is a single-byte instruction and its addressing mode is Implied.

Table 13-47. SED.

SED — SED *Set decimal mode* — **SED**

Operation: 1 → D

N Z C I D V
– – – – 1 –

(Ref: 3.3.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

This instruction sets the decimal mode flag D to a 1 (see Table 13-47), This makes all subsequent ADC and SBC instructions operate as a decimal arithmetic operation.

SED affects no registers in the microprocessor and no flags other than the decimal mode which is set to a 1.

Table 13-48. SEI.

SEI — **SEI** *Set interrupt disable status* — **SEI**

Operation: 1 → I

N Z C I D V
– – – 1 – –

(Ref: 3.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

This instruction initializes the interrupt disable to a 1 (see Table 13-48). It is used to mask interrupt requests during system reset operations and during interrupt command.

It affects no registers in the microprocessor and no flags other than the interrupt disable which is set.

SEI is a single-byte instruction and its addressing mode is Implied.

Table 13-49. STA.

STA — **STA** *Store accumulator in memory* — **STA**

Operation: A → M

N Z C I D V
– – – – – –

(Ref: 2.1.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STA Oper	85	2	3
Zero Page, X	STA Oper, X	95	2	4
Absolute	STA Oper	8D	3	4
Absolute., X	STA Oper. X	9D	3	5
Absolute, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper), Y	91	2	6

This instruction transfers the contents of the accumulator to memory (see Table 13-49).

This instruction affects none of the flags in the processor status register and does not affect the accumulator,

It is a "Group One" instruction and has the following addressing modes available to it: Absolute; Zero Page; Absolute, X; Absolute, Y; Zero Page, X; Indexed Indirect; and Indirect Indexed.

Table 13-50. STX.

STX STX *Store index X in memory* **STX**

Operation: X → M

N Z C I D V
- - - - - -

(Ref: 7.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STX Oper	86	2	3
Zero Page, Y	STX Oper, Y	96	2	4
Absolute	STX Oper	8E	3	4

Transfers value of X register to addressed memory location (see Table 13-50).

No flags or registers in the microprocessor are affected by the store operation. the addressing modes for STX are Absolute, Zero Page, and Zero Page Indexed by Y.

Table 13-51. STY.

STY STY Store index Y in memory **STY**

Operation: Y → M

N Z C I D V
- - - - - -

(Ref: 7.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STY Oper	84	2	3
Zero Page, X	STY Oper, X	94	2	4
Absolute	STY Oper	8C	3	4

Transfer the value of the Y register to the addressed memory location (see Table 13-51). STY does not affect any flags or registers in the microprocessor. The addressing modes for STY are Absolute; Zero Page; and Zero Page Indexed by X.

Table 13-52. TAX.

TAX — TAX *Transfer accumulator to index X* — **TAX**

Operation: A → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

This instruction takes the value from accumulator A and transfers or loads it into the index register X without disturbing the content of the accumulator A (see Table 13-52).

TAX only affects the index register X, does not affect the carry or overflow flags. The N flag is set if the resultant value in the index register X has bit 7 on, otherwise N is reset. The Z bit is set if the content of the register X is 0 as a result of the operation, otherwise it is reset. TAXis a single-byte instruction and its addressing mode is Implied.

Table 13-53. TAY.

TAY — TAY *Transfer accumulator to index Y* — **TAY**

Operation: A → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.13)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

This instruction moves the value of the accumulator into index register Y without affecting the accumulator (see Table 13-53).

TAY instruction only affects the Y register and does not affect either the carry or overflow flags. If the index register Y has bit 7 on, then N is set, otherwise it is reset. If the content of the index register Y equals 0 as a result of the operation, Z is set on, otherwise it is reset.

TAY is a single-byte instruction and the addressing mode is Implied.

Table 13-54. TYA.

TYA TYA *Transfer index Y to accumulator* **TYA**

Operation: Y → A

N Z C I D V
✓ ✓ – – – –

(Ref: 7.14)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

This instruction moves the value that is in the index register Y to accumulator A without disturbing the content of the register Y (see Table 13-54).

TYA does not affect any other register other than the accumulator and does not affect the carry or overflow flag. If the result in the accumulator A has bit 7 on, the N flag is set, otherwise it is reset. If the resultant value in the accumulator A is 0, then the Z flag is set, otherwise it is reset.

The addressing mode is Implied and it is a single-byte instruction.

Table 13-55. TSX.

TSX TSX *Transfer stack pointer to index X* **TSX**

Operation: S → X

N Z C I D V
✓ ✓ – – – –

(Ref: 8.9)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

TXS is a single-byte instruction and its addressing mode is Implied.

Another application for TXS is the concept of passing parameters to the subroutine by storing them immediately after the jump to subroutine instruction.

This instruction transfers the value in the index register X to the stack pointer (see Table 13-55).

TXS changes only the stack pointer, making it equal to the content of the index register X. It does not affect any of the flags.

Table 13-56. TXA.

TXA TXA *Transfer index X to accumulator* **TXA**

Operation: X → A

N Z C I D V
✓ ✓ – – – –

(Ref: 7.12)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

This instruction moves the value that is in the index register X to the accumulator A without disturbing the content of the index register X (see Table 13-56).

TXA does not affect any register other than the accumulator and does not affect the carry or overflow flag. If the result in A has bit 7 on, then the N flag is set, otherwise it is reset. If the resultant value in the accumulator is 0, then the Z flag is set, otherwise it is reset.

The addressing mode is Implied, it is a single-byte instruction.

Table 13-57. TXS.

TXS TXS *Transfer index X to stack pointer* **TXS**

Operation: X → S

N Z C I D V
– – – – – –

(Ref: 8.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

This instruction transfers the value in the stack pointer to the index register X (see Table 13-57).

TSX does not affect the carry or overflow flags. It sets N if bit 7 is on in index X as a result of the instruction, otherwise it is reset. If index X is zero as a result of the TSX, the Z flag is set, otherwise it is reset. TSX changes the value of the index X, making it equal to the content of the stack pointer.

TSX is a single-byte instruction and the addressing mode is Implied.

Chapter 14

Analog Control Projects

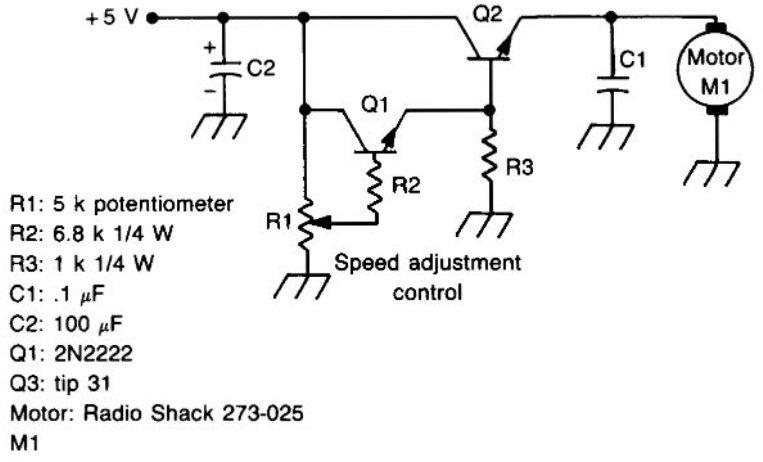
THE PROJECTS IN THIS CHAPTER ARE DESIGNED to demonstrate the basic capability of analog voltage control. Analog voltage control is used in process control systems, servomechanisms, and other electronic circuits such as the AVC circuit *(automatic volume control)* in a radio. Analog control lends itself nicely for control purposes because of the ease with which you can use a simple potentiometer as a voltage developing sensor for data pick-up and a simple transistor as a voltage or current driver circuit. The basic concept of analog control is quite simple, but in practice it can become technically very complex. The projects that are presented in this chapter have been kept simple, but they still present a good basic demonstration of how analog control works.

Control systems come in two forms, which are open-loop and closed-loop systems. The open-loop system is one where the input control signal is independent of the system's output operation. The closed-loop control system uses an input driving signal that is dependent upon a feedback signal from the output circuit. Two open-loop motor-control systems are shown in Figs. 14-1 and 14-2. A closed loop control system is shown in Fig. 14-3.

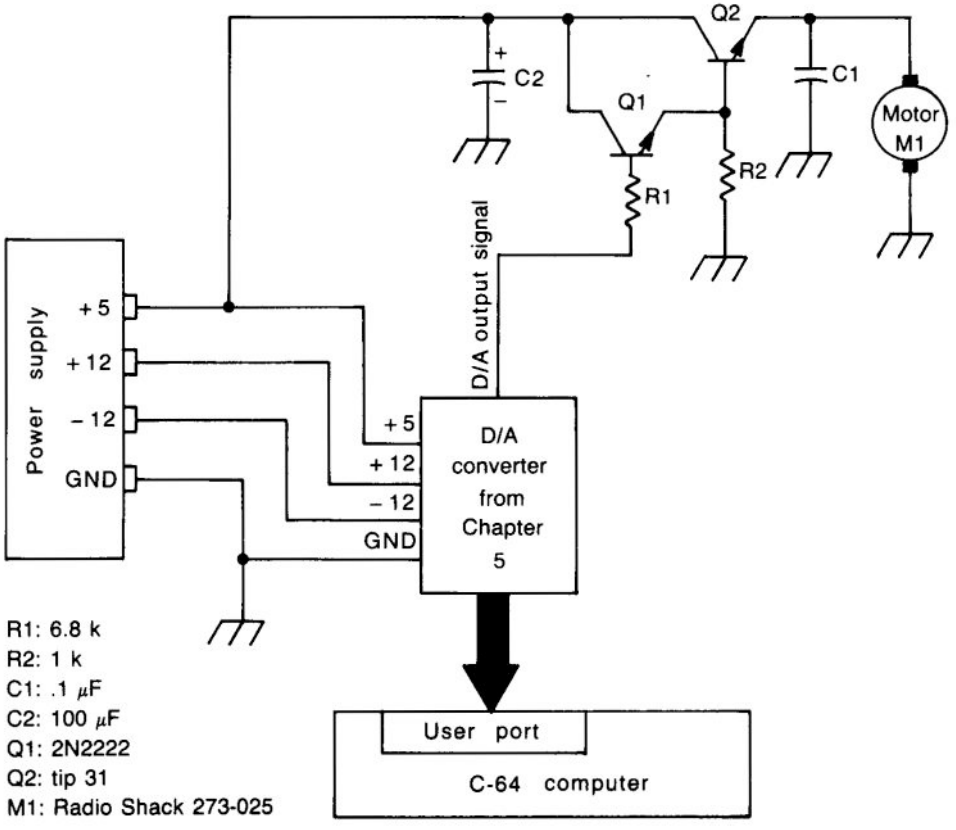
OPEN·LOOP MOTOR·SPEED CONTROL

Figure 14-1 shows the basic dc motor-control circuit. The circuit is a simple series-regulator circuit that controls the dc current to the motor. Transistor Q2 operates as a series-pass transistor in an emitter-follower dc amplifier circuit. In this circuit, the controlling base current at transistor Q1 can be very small because of the current amplification of the circuit. Because only a small base current is required to drive this amplifier circuit, the driving current can be supplied by a potentiometer as shown in Fig. 14-1. A computer-controlled digital-to-analog converter circuit can also drive the motor speed-control circuit as shown in Fig. 14-2. In the circuit of Fig. 14-1, the motor speed is dependent on the setting of potentiometer R1, and in the circuit of Fig. 14-2, the motor speed is dependent on the analog output from the digital-to-analog converter.

Programs 14-1 and 14-2 are two open-loopcon-



Fig, 14-1. An open-loop dc motor speed-control circuit.



Fig, 14-2. An open-loop computer-controlled dc motor speed-control circuit.

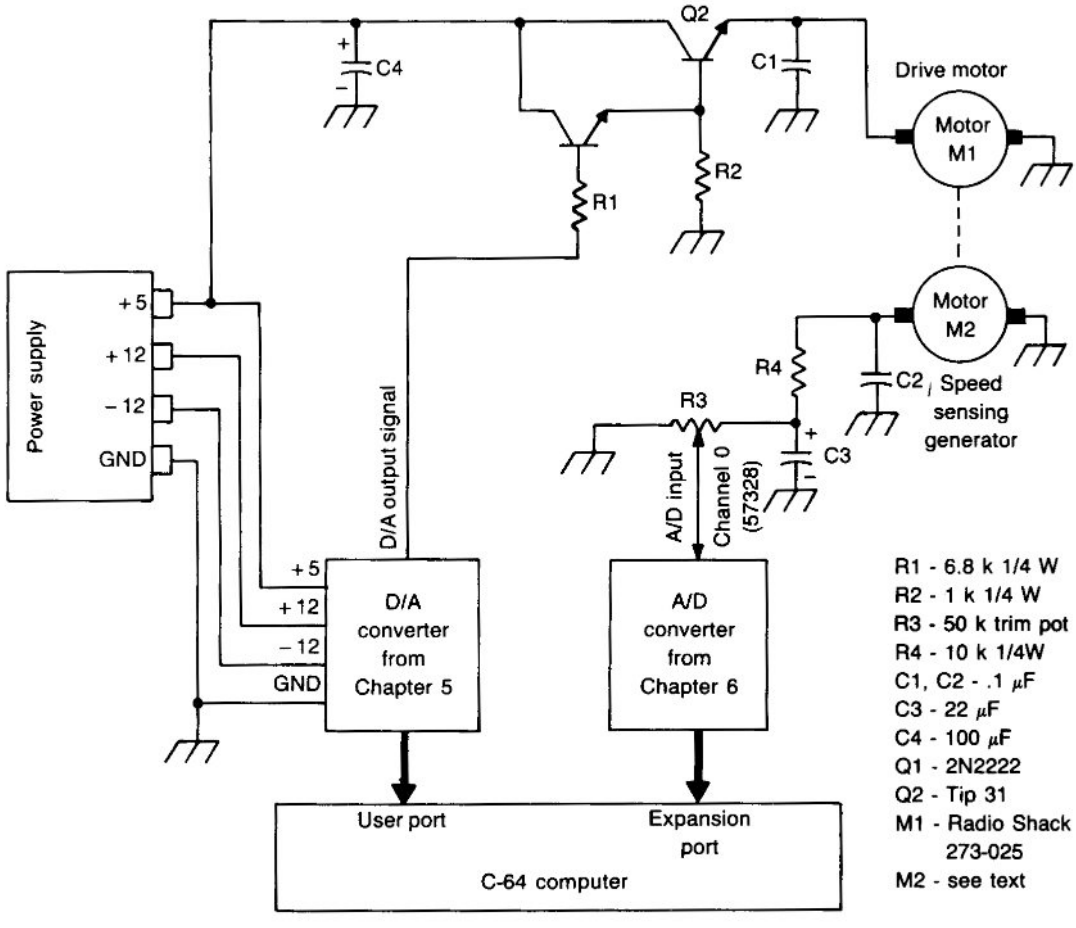


Fig. 14-3. A closed-loop computer-controlled dc motor speed-control circuit.

```
1 REM : PROGRAM 14.1 FOR THE C-64
2 REM : BASIC OPEN LOOP MOTOR SPEED CONTROL DEMONSTRATION ( SAW TOOTH RAMP )
3 REM : FOR USE WITH FIGURE 14.2
10 POKE56579,255
20 A=0
30 POKE 56577,A
40 A=A+1
50 IF A=255 THEN A=0
60 GOTO30
```

Program 14-1. An open-loop dc motor control demonstration program for Fig. 14-2.

```
1 REM : PROGRAM 14.2 FOR THE C-64
2 REM : OPEN LOOP MOTOR SPEED CONTROL PROGRAM
3 REM : FOR USE WITH FIGURE 14.2
5 PRINTCHR$(147):PRINT" "
7 PRINT"PRESS <I> TO INCREASE MOTOR SPEED"
8 PRINT"PRESS <S> TO SLOW  MOTOR SPEED"
10 POKE56579,255
20 A=0
30 POKE 56577,A:PRINT CHR$(19);:PRINTA
40 GET A$
50 IF A$ = "I" THEN GOTO 80
60 IF A$ = "S" THEN GOTO 90
70 GOTO 40
80 A=A-1: IF A=256 THEN A=255
85 GOTO 30
90 A=A-1: IF A=-1 THEN A=1
95 GOTO 30
100 PRINT"A"
```

Program 14-2. An open-loop dc motor speed-control program for Fig. 14-2.

trol programs for the C-64 computer that demonstrate the principles of open-loop motor-speed control. The two programs are designed to use the D/A converter from Chapter 5. The D/A converter should be adjusted to generate an output voltage from zero to five volts. Program 14-1 turns the D/A converter into a sawtooth signal generator, which causes the motor to start out slow and speed up to a maximum rpm speed. The motor will then stop and start over again. Program 14-2 is an open-loop program that gives you the ability to control the motor speed from the C-64 keyboard by entering control numbers between 0 and 255.

CLOSED·LOOP MOTOR·SPEED CONTROL

Using the open-loop control Program 14-2 along with the circuit of Fig. 14-2, you can easily control the motor rpm speed. About 10 to 15 percent of the actual motor speed is really controlled by the load that is placed on the motor. You can adjust the motor speed to 1000 rpm, but if the load on the motor increases or decreases, the motor rpm speed will change accordingly. If you are going to keep the motor speed at 1000 rpm, you will need a feedback signal from a rpm detecting sensor to generate a rpm speed adjustment. When you add a feedback control circuit, you have created a closed-loop control system. A closed-loop motor-control circuit is presented in Fig. 14-3. Program 14-3 is the supporting computer control program for Fig. 14-3.

The heart of this closed-loop system is the motor rpm detection sensor. The sensor used in Fig. 14-3 is a permanent magnet dc motor (M2) that is used as a dc voltage generator. The motor was taken out of a used tape recorder. A tape recorder motor is a good quality motor that will generate a dc voltage between zero and eight volts if you open up the motor and disable the rpm speed governor. Most motors use a centrifuge type governor that can be disabled by simply soldering the governor contacts together. The shaft of the drive motor (M1) is connected to the shaft of the rpm sensor-generator (M2) The output voltage of the rpm sensor-generator is directly proportional to the speed of the drive motor. The output voltage of the sensor-generator circuit is connected to channel zero of the *AID* converter, which was presented in Chapter 6.

When you have the motor speed-control circuit of Fig. 14-3 working as an open-loop controller, add the speed-sensing circuit elements, which are M2, R3, R4, C2, and C3. Adjust R3 so the voltage at the *AID* converter input is 5 volts at maximum motor drive speed. When the speed-sensing circuit

is functional, you are ready to load the closed-loop control Program 14-3 into the C-64. Program 14-3 is an elementary closed-loop speed control program. The program's speed control function is slow to react to a motor load change, but it will give you a good demonstration of a closed-loop feedback control system. When the program is run, it will ask you to input the motor running speed data. If you are using a Radio Shack drive motor, the input data number will need to be greater than about 100 to start the motor. The program is written to let the motor speed reach its selected operation speed before the speed control function starts working. Figure 14-4 shows the data display that is displayed on the video monitor when this program is running. The SPEED INPUT DATA is the entered data that controls the running motor speed. The SPEED SENSING LEVEL is the output data from the *AID* converter after the motor has reached its running speed. The SENSOR DATA is the current *AID* converter output data, which is a relative indication of the actual motor speed. The sensor data is

```
1 REM : PROGRAM 14.3 FOR THE C-64
2 REM : CLOSED LOOP MOTOR SPEED CONTROL DEMONSTRATION PROGRAM
3 REM : FOR FIGURE 14.3
10 PRINTCHR$(147):POKE56579,255:POKE56577,0
20 PRINT"MOTOR SPEED CONTROL PROGRAM"
30 PRINT" ":PRINT" SPEED INPUT DATA - SPEED SENSING LEVEL"
40 PRINT" "
50 PRINTCHR$(19): PRINTTAB(200):INPUT" INPUT SPEED DATA (1-255) - ";A
60 PRINTCHR$(19):PRINTTAB(120);:PRINT"                                    "
70 FOR I = 1 TO 10:PRINT" ":NEXT
80 PRINT"SENSOR DATA - ERROR DATA - CONTROL DATA"
100 POKE 56577,A
110 POKE57328,00 : C=PEEK(57328)
120 FOR I=1 TO 50:NEXT
130 POKE57328,00 : D=PEEK(57328)
140 PRINTCHR$(19):PRINTTAB(120);:
145 PRINT"          ";:PRINTA;:PRINT"            ";:PRINTC
150 IF ABS(D-C)<2 THEN GOTO180
160 GOTO 110
180 PRINTCHR$(19):PRINTTAB(120);:PRINT"                                    "
190 PRINTCHR$(19):PRINTTAB(120);:
195 PRINT"          ";:PRINTA;:PRINT"            ";:PRINTC
200 PRINTTAB(200):PRINT"PRESS<C> TO CHANGE SPEED":Q=A
210 GET A$: IF A$="C" THEN GOTO50
220 POKE 57328,00: X=PEEK(57328)
222 PRINTCHR$(19):FOR I= 1 TO 8:PRINTTAB(080):NEXT:
224 PRINT"                                                                "
225 PRINTCHR$(19):FOR I= 1 TO 8:PRINTTAB(080):NEXT:
228 PRINT"    ";:PRINTX;:PRINT"          ";:PRINTY;:PRINT"         ";:PRINTQ
230 Y=C-X
240 IF Y>4 THEN GOTO 260
250 IF Y<4 THEN GOTO 270
255 GOTO210
260 Q=Q+ABS(INT(Y/05)):GOTO 280
270 Q=Q-ABS(INT(Y/05)) :GOTO 280
280 IF Q>255 THEN Q=255
285 IF Q<0  THEN Q=0
290 FOR I=1TO150:NEXT:POKE56577,Q
300 GOTO210
```

Program 14-3. A closed-loop dc motor speed-control program.

```
Motor speed control program

Speed input data - speed sensing level
        130H              75

INPUT SPEED DATA (1-255) - ? 130

Press <C> to change speed

Sensor data - error data - control data
    74            1            128
```

Fig. 14-4. Video data display that is generated by the closed-loop speed-control Program 14-3.

updated several times each second. The ERROR DATA is the difference between the speed sensing data and the sensor data. The error data is used to compute the CONTROL DATA number that is POKED into the USER PORT address location to correct the motor speed. The program's closed-loop function will attempt to keep the speed sensing level number and the sensor data number the same number. You can observe this control action by holding the motor drive shaft slightly to slow down the motor speed.

You might consider the closed-loop control system very elementary. Even with its simplicity, it will give you a good demonstration of a closed-loop motor speed control system.

CLOSED·LOOP SERVO·CONTROL SYSTEM

The preceding projects were about controlling the rotational speed of motors. Another control function that is used in robotics, automation, and hobby control projects is the control of a shaft's rotational position. This type of shaft rotational position control is usually referred to as SERVO control. One hobby area in which servos are used is radio-controlled models. The object of this servo-control project is to control an electronic servo circuit with a computer.

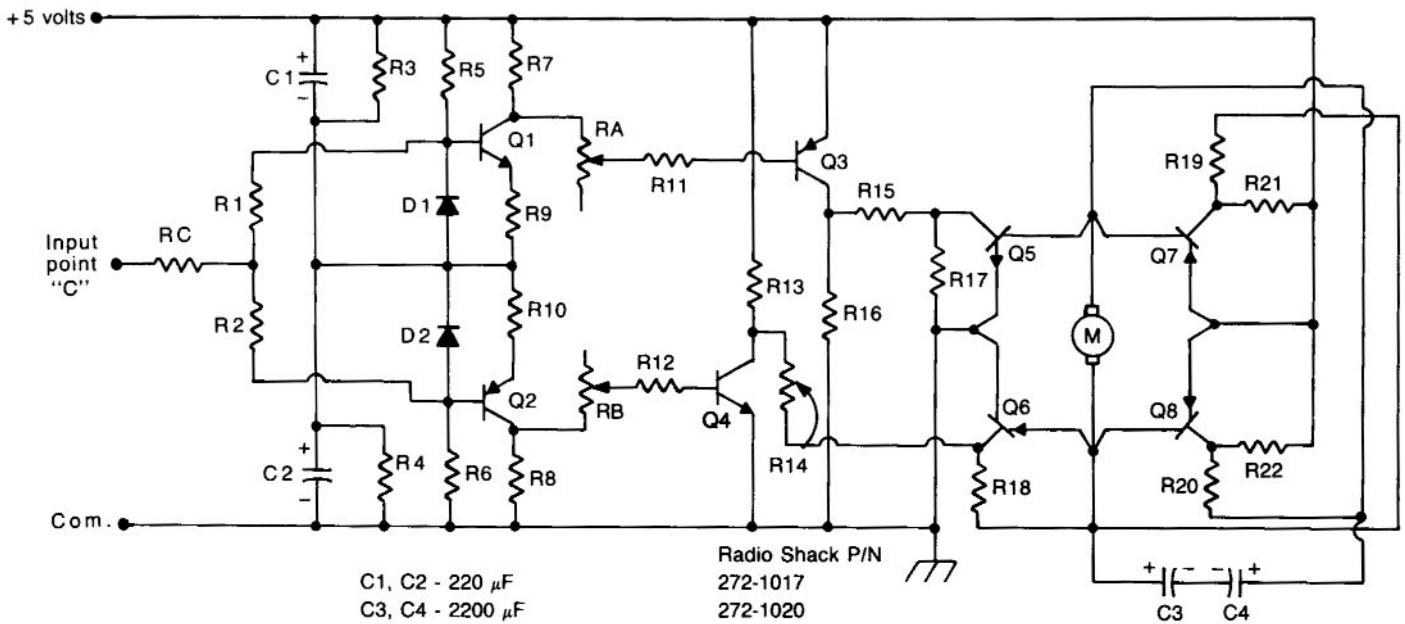
The servo control system is a closed-loop control system that uses two basic electronic circuits, which are a motor direction and speed control circuit and a shaft position pick-up sensor circuit. The basic operation of the servo system is to generate an electronic signal from an input data command to move the servo control shaft to a given rotational position. A generated electronic signal turns on the servo drive motor to move the control shaft in the proper direction, and when the shaft position sensor indicates that the shaft is in the correct position, another signal is generated to stop the motor. The basic operation sounds quite simple, but the required operational technology can become very complex depending on the required positional accuracy and servo operating speed. The project that is described here does not require a high degree of accuracy or speed because it is only a demonstration project. This project does have the capability to perform the servo functions that are required for hobby-level robotics.

The complete servo control system is built around the circuit in Fig. 14-5, which is a bridge-style servo amplifier that will control both the motor speed and direction of rotation. The servo circuit is a redesigned radio-control circuit that will work from a single five-volt power supply.1 The single five-volt supply voltage will make the servo amplifier compatible with the D/A and *AID* converters that have been presented in this book. The servo circuit requires an input signal voltage of 2.5 volts to stop the motor and keep it at rest. An input voltage above 2.5 volts will cause motor to run in one direction and an input voltage below 2.5 volts will cause the motor to run in the other direction. The motor speed is controlled by how much the input voltage is moved from the 2.5 volts resting point. The servo's motor-control characteristics are not linear because of the dc motor's operational characteristics and the circuit's transistor amplifier current gains.

When you have the servo amplifier completed, you can connect it to the D/A converter and the C-64 for an open-loop control test. Figure 14-6

---

(1)Howard G. McEntee, *Radio Control Handbook,* TAB Books, Inc. 1971, pg. 147.



	Radio Shack P/N
C1, C2 - 220 μF	272-1017
C3, C4 - 2200 μF	272-1020
RA, RB - 25 k trimpot	
RC - 10 k	
R1, R2 - 2.7 k	
R3, R4, R19, R20 - 47 ohm	
R5, R6 - 27 k	
R7, R8 - 1.5 k	
R9, R10, R13, R16 - 100 ohm	
R11, R12 - 4.7 k	
R14, R15 - 10 ohm	
R17, R18, R21, R22 - 1 k	
D1, D2 - 1N914	276-1122
Q1, Q4 - NPN	276-2009
Q2, Q3 - PNP	276-2023
Q5 - NPN	276-2020
Q6, Q7, Q8 - PNP	276-2027
M - Motor	273-025

Fig. 14-5. A 5-volt single-supply voltage servo motor-control circuit.

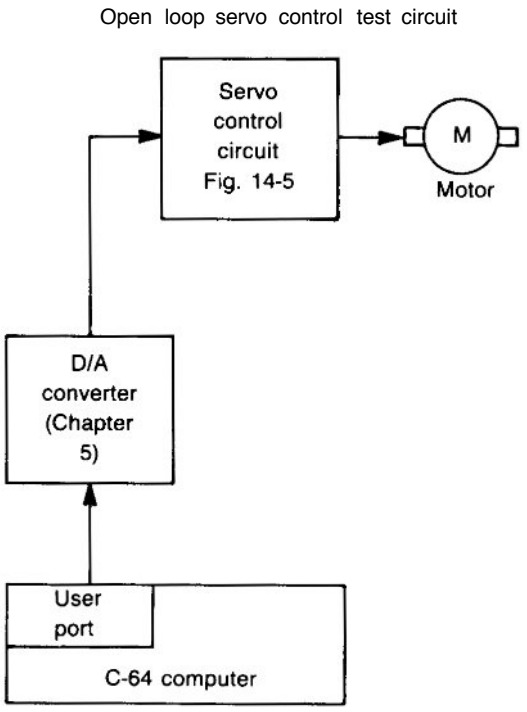


Fig. 14-6. Block diagram of the open-loop servo test circuit.

shows a block diagram of the open-loop test circuit. This test circuit can be used with Program 14-4 to test the servo amplifier. Open-loop circuit testing should give you some test results that are similar to the data that is presented in Fig. 14-7. The actual control characteristics will vary with different motors and transistor gains, but the overall operation should be close. Circuit trimpots RA and RB along with resistor RC can be varied to control the maximum motor speed and the speed ramp-up function.

A complete closed-loop servo control system is presented in Fig. 14-8 along with its supporting servo control Program 14-5. The servo system uses a C-64 as the system computer, the D/A converter from Chapter 5 to drive the servo amp, and the *AID* converter from Chapter 6 to detect the shaft rotational position. A standard 1K potentiometer is connected to the main control shaft and used as the shaft position sensor. The position sensor will develop a zero to five volts signal that is dependent on the shaft's rotational position. The control shaft can not rotate a complete 360 degrees because of the potentiometer end stops.

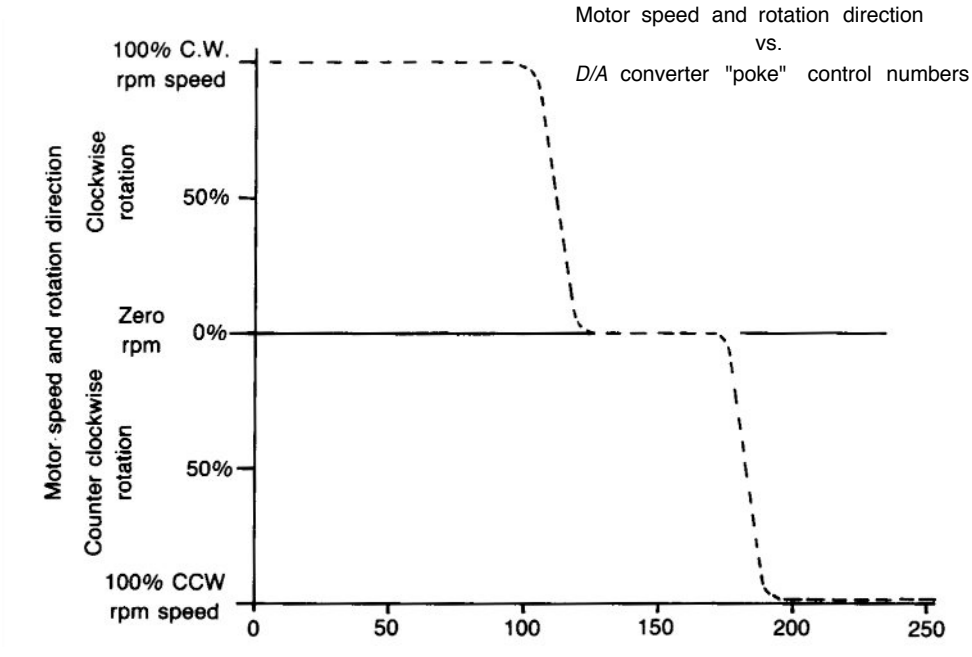
The drive motor is connected to the control shaft through a series of pulleys to permit the motor to run at a high speed and the control shaft to run at a slow speed. The speed reduction also gives the control shaft more driving power. The pulleys in this project were taken from a used tape recorder, but pulleys and drive belts can be purchased at hobby shops.

To really put on the best demonstration, you should label the main control shaft pulley with the numbers 0, 127, and 255 which indicates the output number from the *AID* converter for that shaft position. When you load in the servo program, it will ask you to INPUT a data number from 0 to 255. When a number is entered, the servo system will move the control shaft so that number is next to external pointer.

The positional accuracy and speed of operational of this system is limited only by the ability of the programmer to write the controlling servo

```
1 REM : PROGRAM 14.4
2 REM : COMPUTER CONTROLLED SERVO DRIVE TESTING PROGRAM FOR THE C-64
3 REM : USE WITH FIGURES 14.5 AND 14.6
10 POKE56579,255:POKE56577,140:PRINTCHR$(147)
15 INPUT "INPUT MOTOR DIRECTION AND SPEED DATA      (0-255) - ";A
20 POKE 56577,A:GOTO15
```

Program 14-4. An open-loop servo test program for Fig. 14-6.



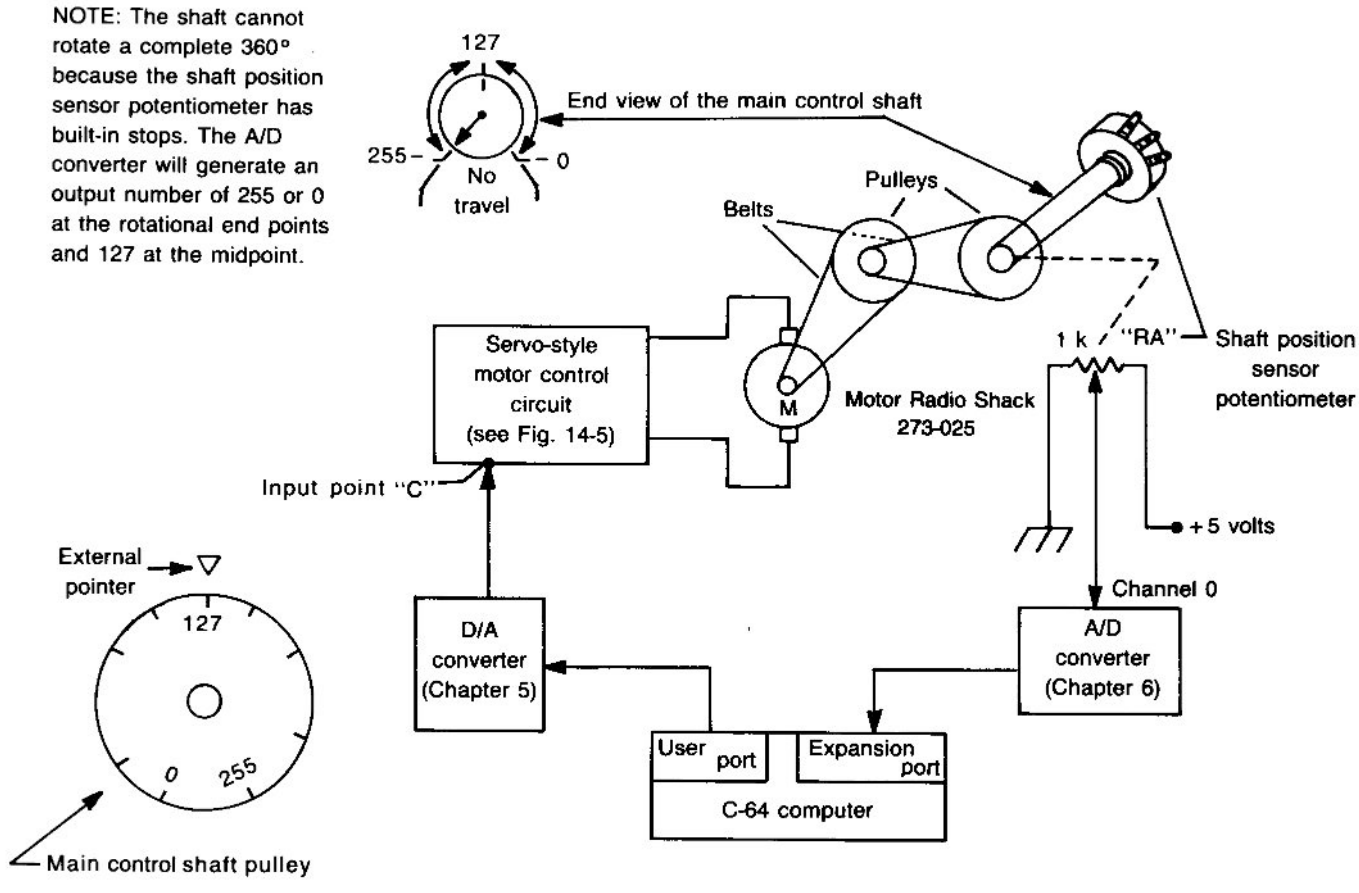
Notes:

1. Poke control numbers less than 119 causes a clockwise rotation.
2. Poke control numbers greater than 178 causes a counter clockwise rotation.

Fig. 14-7. Graphical data display shows the relationship between the servo motor speed and direction and the *D/A* converter POKE control numbers.

```
1 REM : PROGRAM 14.5
2 REM : COMPUTER CONTROLLED SERVO DEMONSTRAITION PROGRAM FOR THE C-64.
3 REM : USE WITH FIGURES 14.5 AND 14.8
10 POKE56579,255:POKE56577,140
15 A=127: GOTO 30
20 PRINTCHR$(147):PRINT"SHAFT POSITION IS NOW AT ";:PRINTA:PRINT" "
25 INPUT "INPUT NEW POSITION DATA (0-255) -";A
30 POKE 57328,00:B=PEEK(57328)
40 C =A-B
50 IF ABS(C))03 THEN GOTO70
60 POKE56577, X :GOTO20
70 IF C<0 THEN GOTO 100
80 IF C>0 THEN GOTO 200
100 POKE56577,200:GOTO30
200 POKE56577,20:GOTO30
```

Program 14-5. A closed-loop servo control program for Fig.14-B.



A closed-loop servo-style computer control system

Fig. 14-8. The closed-loop computer-controlled servo control system circuit.

program. The presented program is very simple, but it will perform a nice demonstration for you.

CONCLUSION

This chapter has presented a series of open- and closed-loop control projects. After you have completed these control projects, you will be able to comprehend the electronic control functions and problems that are experienced in and connected with the field of robotics.

Chapter 15

Useful Hobby and Control Circuits

THIS CHAPTER PRESENTS VARIOUS ELECtronic circuits that are useful in the area of electronic hobby projects_ These circuits have been used by me for various electronic projects, and they can be used for many other projects with slight modifications to fit your needs.

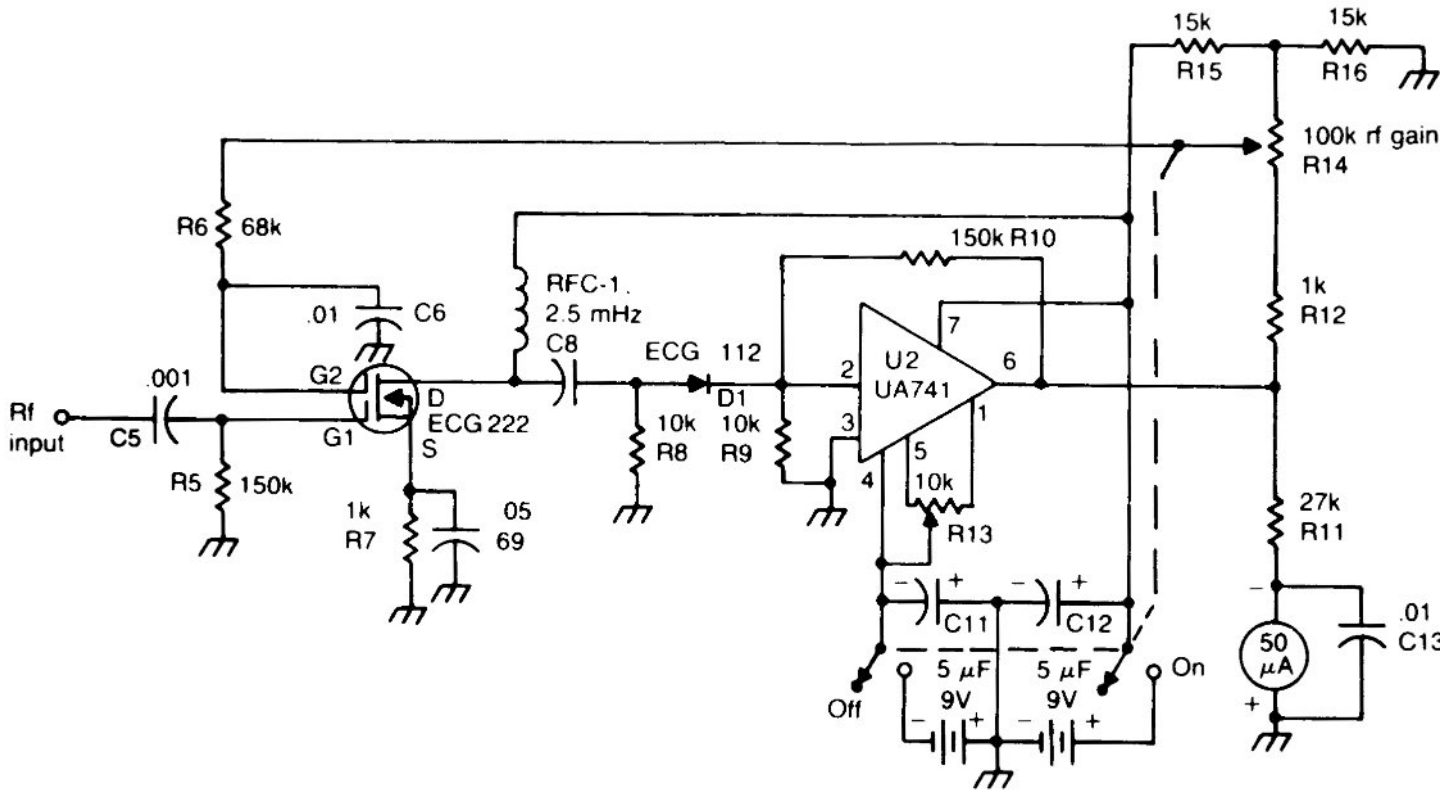


Fig. 15-1. A wide-band rf detector circuit. The dc signal indication on the meter is a relative rf field-strength reading.

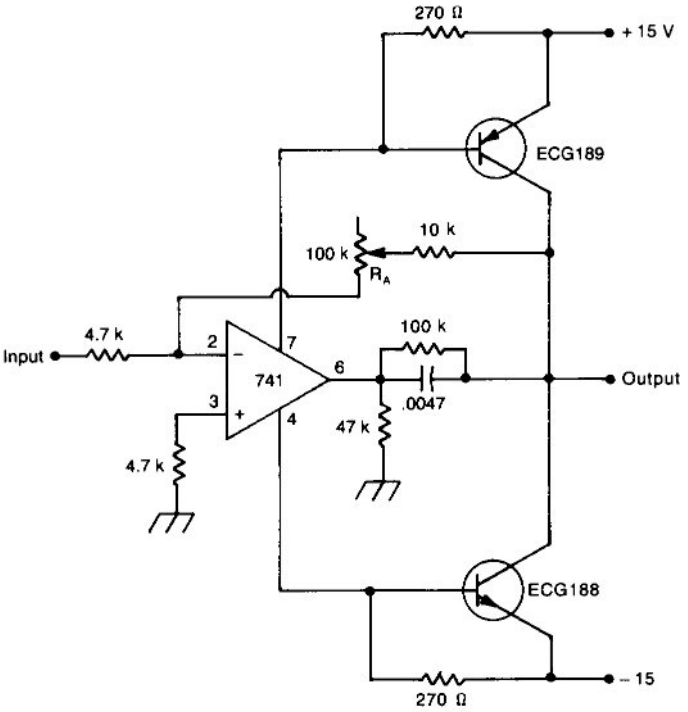
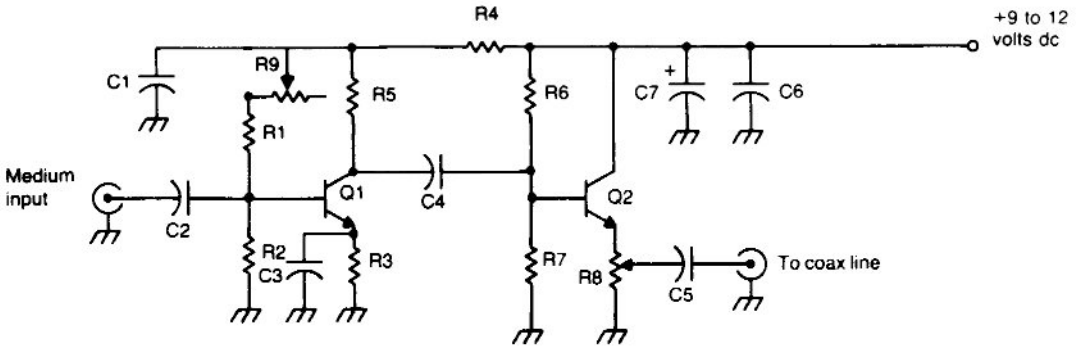


Fig. 15-2. An op-amp power booster circuit.



Parts List For Rf Amplifier-Buffer Circuit

R1, R2 - 10k resistor
R3 - 1.0k resistor
R4 - 270 ohm resistor
R5 - 1.5k resistor
R6, R7 - 6.8k resistor
R8 - 500 ohm trim pot (carbon or cermet)
R9 - 50k trim pot
C1 to C6 - .1 μF Mylar capacitor
C7 - 22 μF, 16 volts capacitor
FB-1 Ferrite Bead, Radio Shack 273-1571
Q1, Q2 - 2N2222 transistor

Fig. 15-3. A buffer amplifier circuit. This amplifier can be used to amplify an audio level signal from a data sensor and transmit the signal on a coax cable for a long distance.

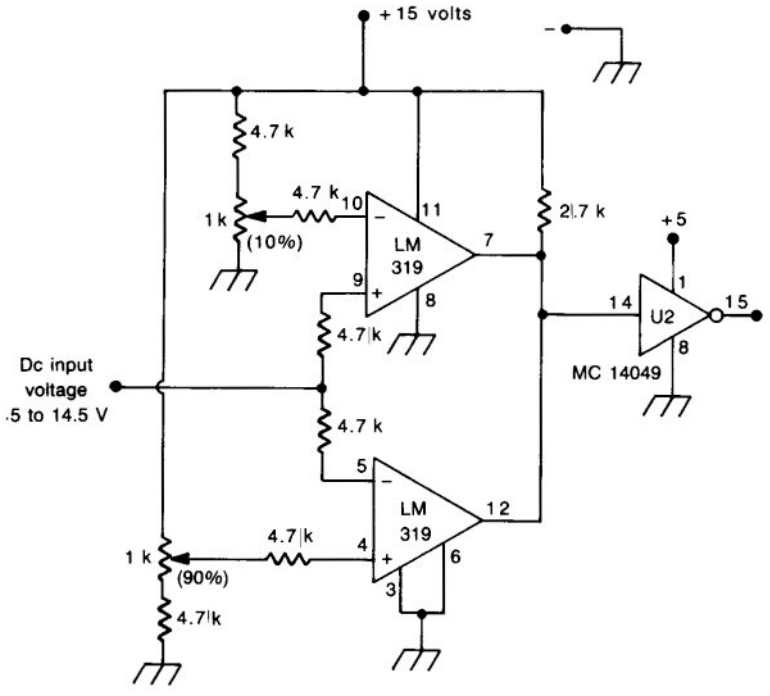


Fig. 15-4. A window comparator circuit. The circuit will work with supply voltages between 5 and 15 volts. In its present configuration, it is a rise and fall detector circuit. The circuit is set-up to generate a logic pulse at the output of U2 whose pulse length is equal to the rise or fall time of the input signal. If the rise or fall signal is longer than about 20 microseconds, a computer can be used to measure the rise and fall time by measuring the pulse width of the output signal.

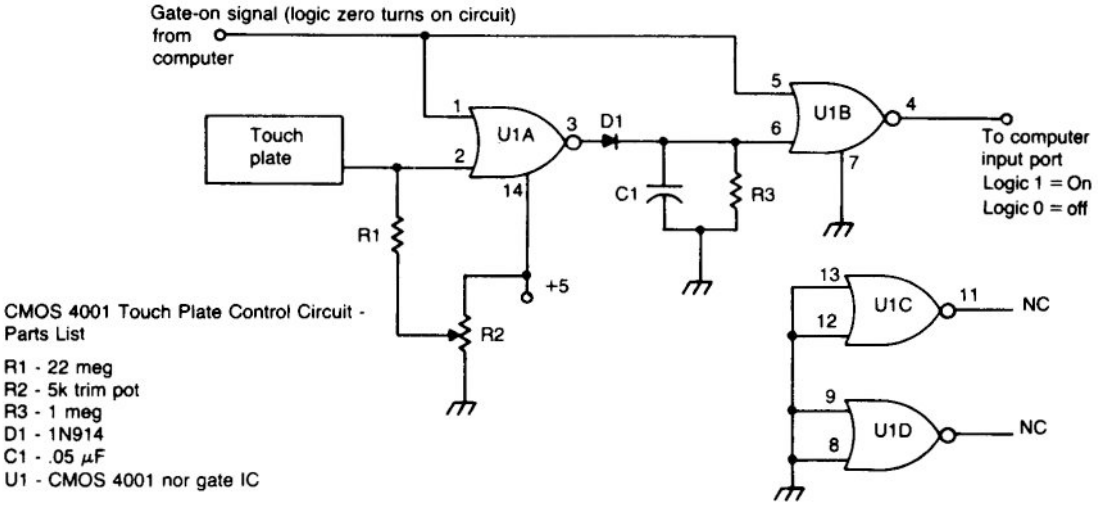
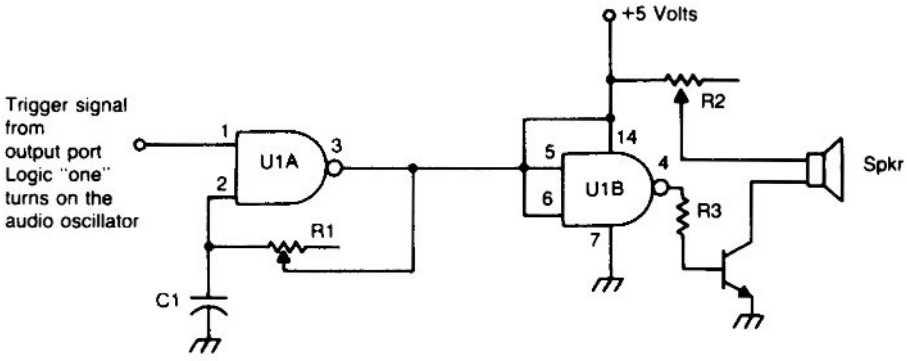


Fig. 15-5. A computer-controlled touch-plate circuit. The circuit is designed to generate a logic ONE pulse when you touch the touch plate. The circuit can also be turned off and on by the computer.



Computer Controlled Audio Oscillator - Parts List

R1 - 50k trim pot	C1 - .22 μF capacitor
R2 - 500 ohm trim pot	U1A and U1B - CMOS 4093
R3 - 6.8k resistor	Spkr - Radio Shack 40-245

Fig. 15-6. A computer-controlled audio oscillator.

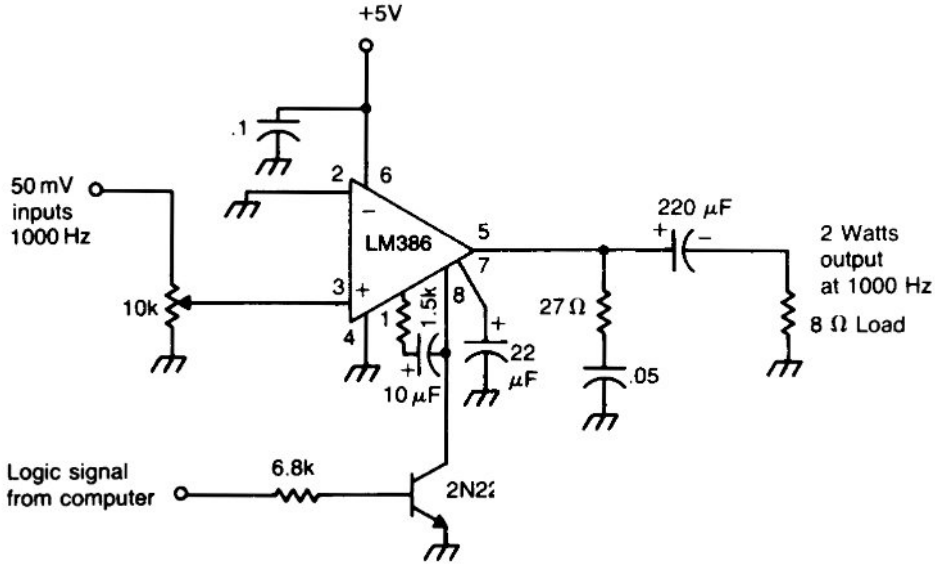
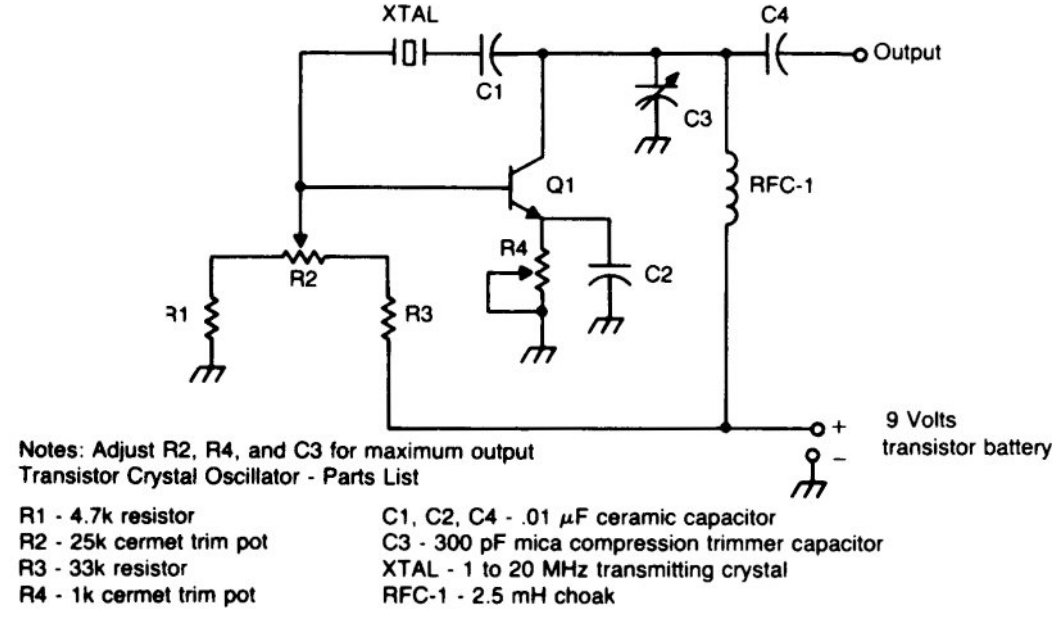


Fig. 15-7. A computer-controlled audio amplifier.

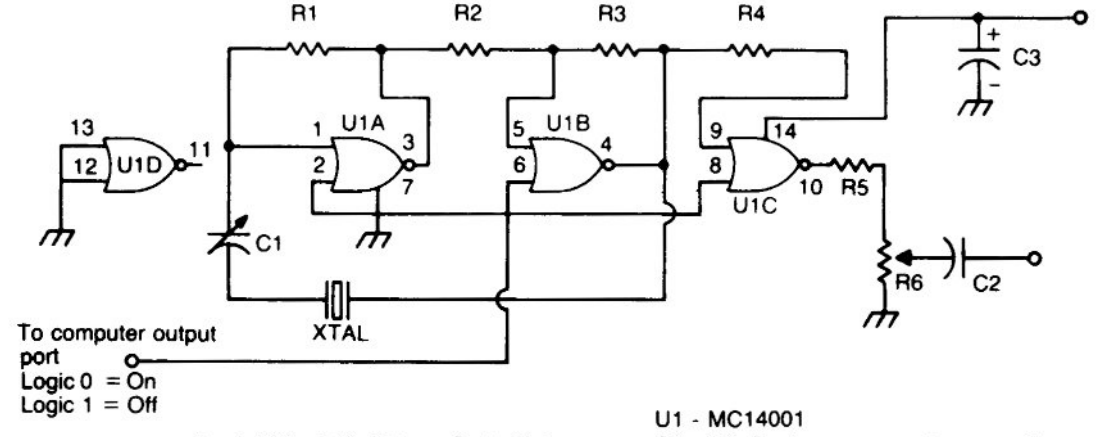


Notes: Adjust R2, R4, and C3 for maximum output
Transistor Crystal Oscillator - Parts List

R1 - 4.7k resistor
R2 - 25k cermet trim pot
R3 - 33k resistor
R4 - 1k cermet trim pot

C1, C2, C4 - .01 μF ceramic capacitor
C3 - 300 pF mica compression trimmer capacitor
XTAL - 1 to 20 MHz transmitting crystal
RFC-1 - 2.5 mH choak

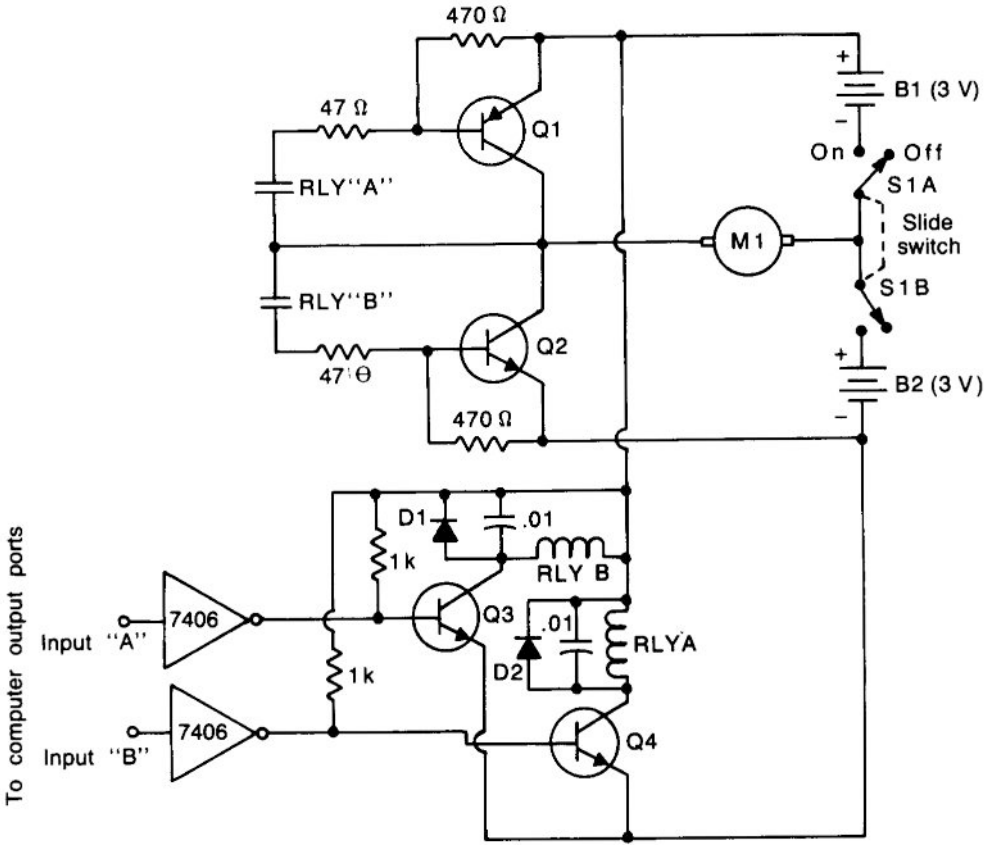
Fig. 15-8. A transistor crystal-controlled oscillator circuit.



Crystal Clock Oscillator - Parts List
R1, R3, R4 - 10k resistor
R2, R5 - 2.7k resistor
R6 - 5k cermet trim pot

U1 - MC14001
C1 - 60 pF mica compression capacitor
C2 - .001 μF ceramic capacitor
C3 - 1000 μF 16 volt capacitor
XTAL - 1 to 5 MHz clock crystal

Fig. 15-9. A **CMOS** crystal-controlled oscillator.



Radio Shack PIN

Q1 - 276-2027
Q2 - 276-2020
Q3, Q4 - 276-2009
M1 - 273-025
S1 - OPOT slide switch
B1, B2, - 2 "AA" battery cells
01,02 - 1N914
RLVA, RLVB - small 5 V Radio Shack relay

Note:
1: Both inputs must be logic "ONE" to turn off the motor

2: Make sure that both inputs never go to a logic "ZERO" at the same time because this would turn on both Q1 and Q2 which would short out the batteries.

Fig. 15-10. A small dc motor control circuit that is designed so you can turn the motor on and off and control the direction of rotation with a computer.

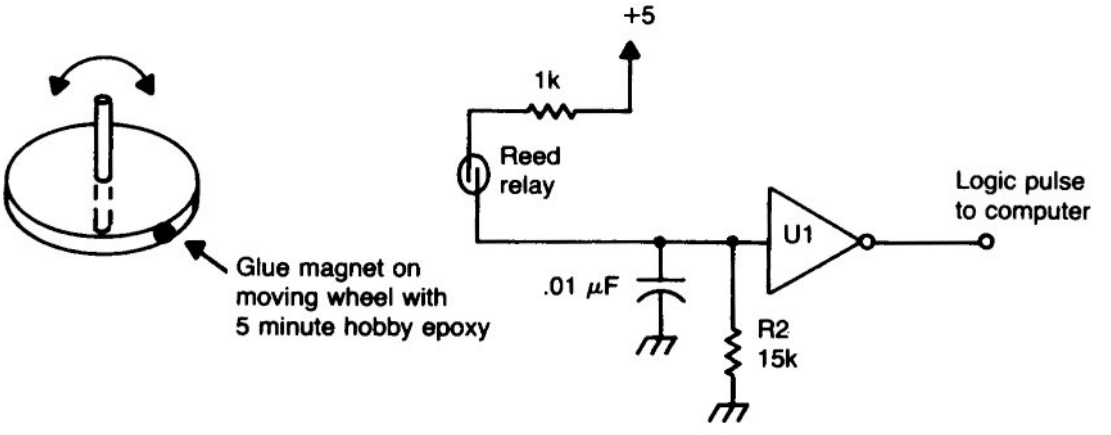


Fig. 15-11. How to use a reed relay and a magnet as a rpm sensing circuit or a rotating position sensor.

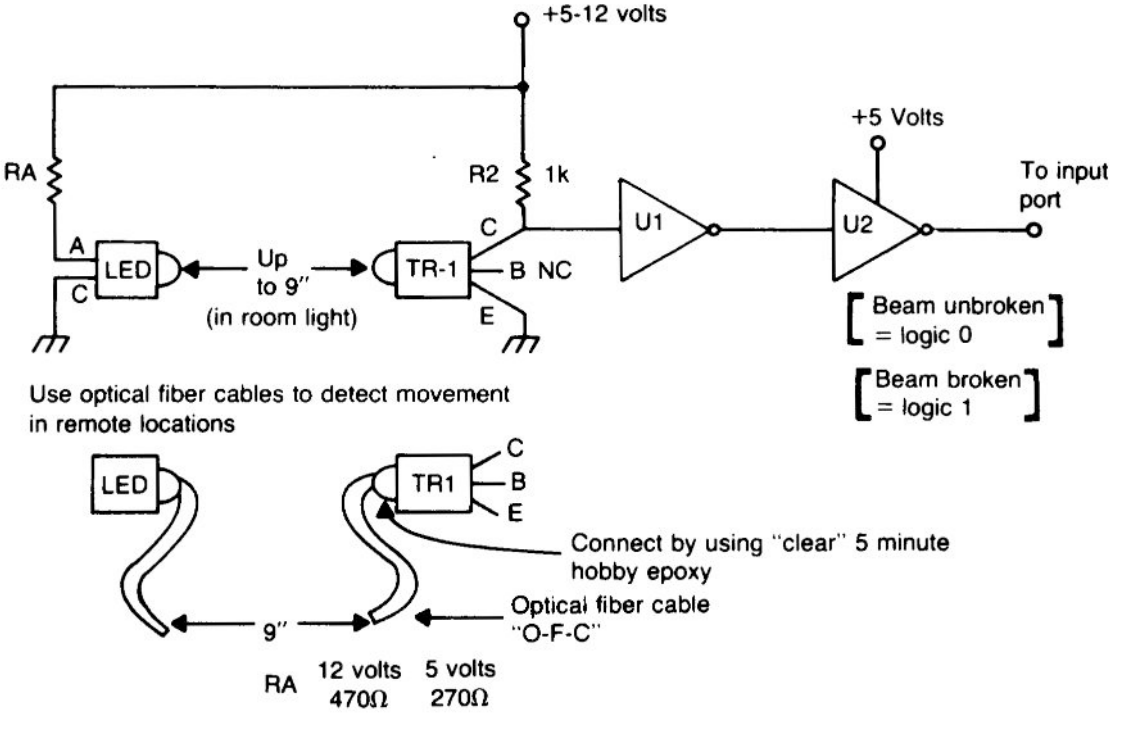


Fig. 15-12. An optical sensing circuit. It will input data into the computer anytime the light beam is broken.

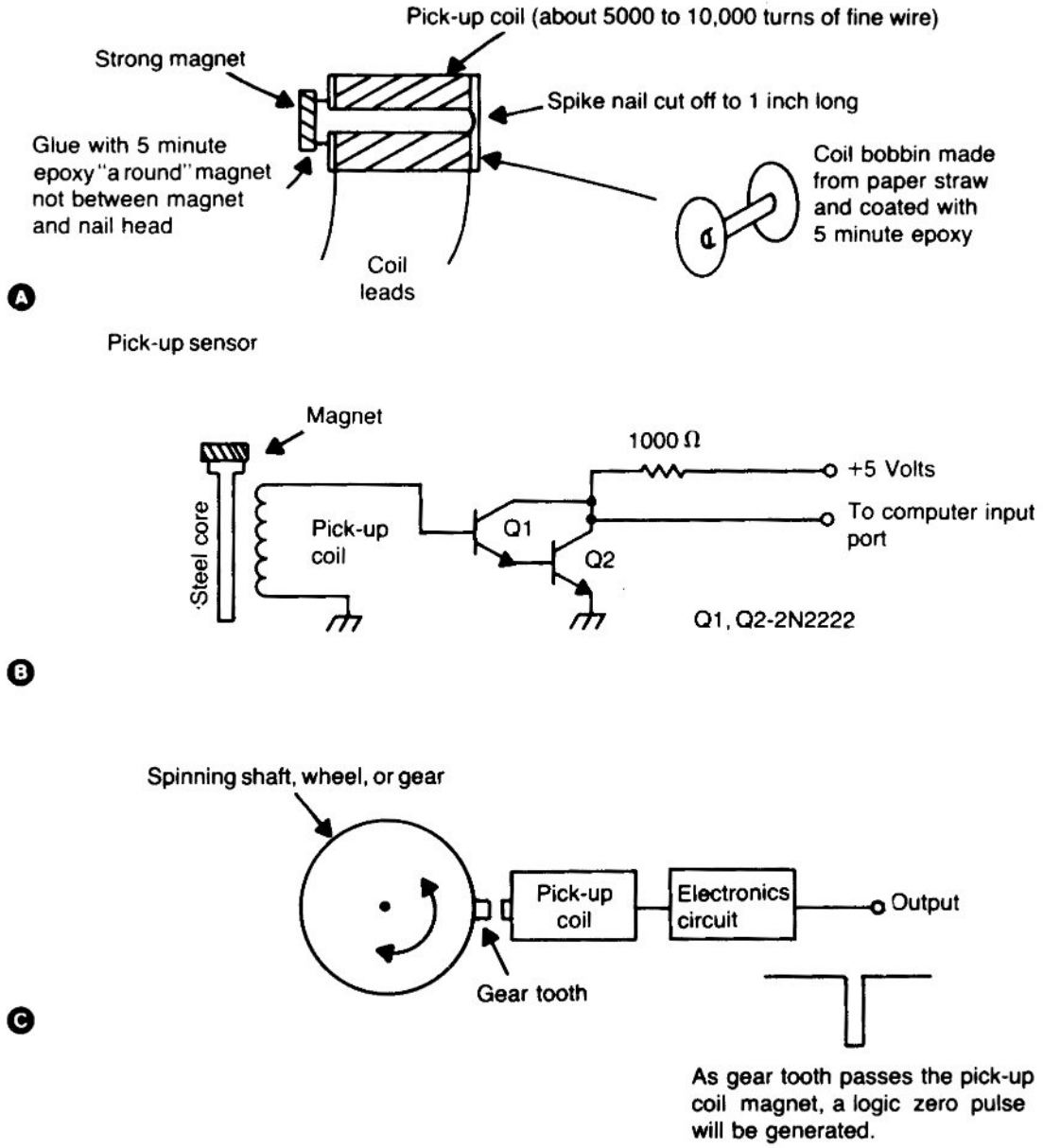


Fig. 15-13. A magnetic pick-up coil gear tooth sensor. When a gear tooth passes by the magnet, a change in the magnetic field flux around the magnet causes a voltage to be developed in the pick-up coil. The pick-up coil voltage is then used to turn-on the transistor circuit and generate a logic pulse every time a gear tooth passes by the magnet. A shows the physical construction of the sensor assembly, B shows the electrical circuit, and C shows the practical application method. This sensor requires a very strong magnet to generate logic pulses at slow gear speeds.

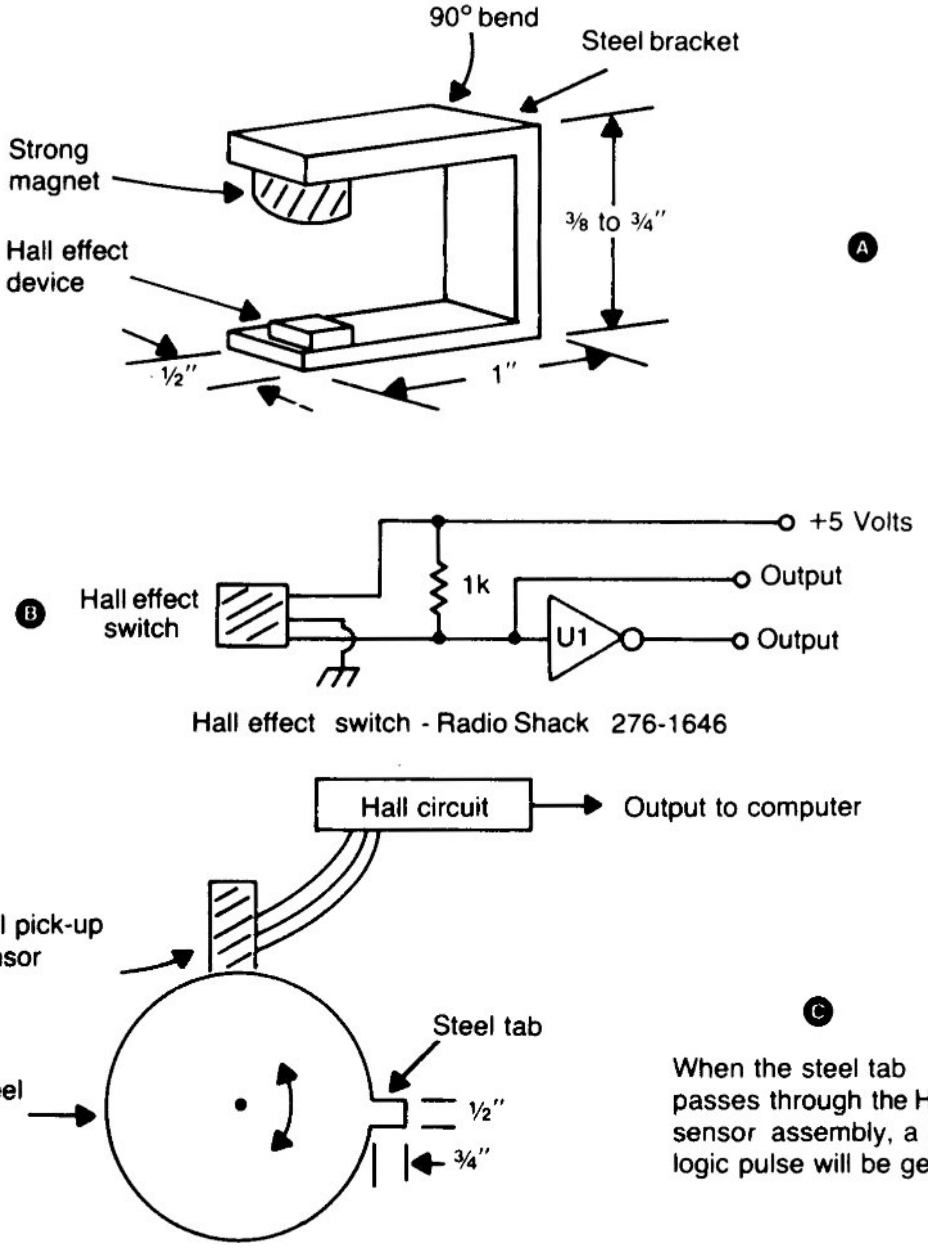


Fig. 15-14. Using a Hall-effect switch to input rotational position information into a computer. When the steel tab interrupts the magnetlc circuit between the magnet and the Hall device, the Hall-effect switch will change logic states. Some Hall devices will have switching times of less than one microsecond which will supply very accurate data to the computer.

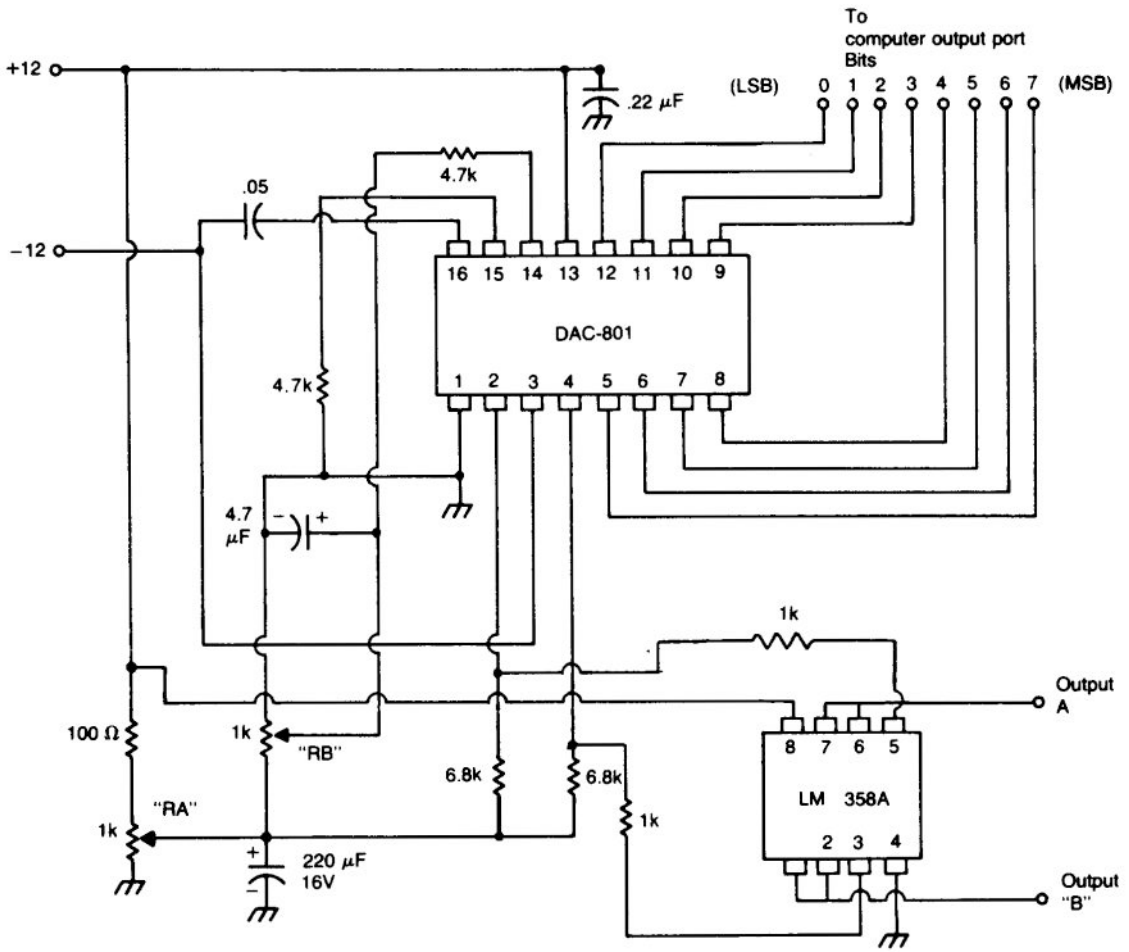


Fig. 15-15. This is another digital-to-analog circuit that can be built from easy to find parts.

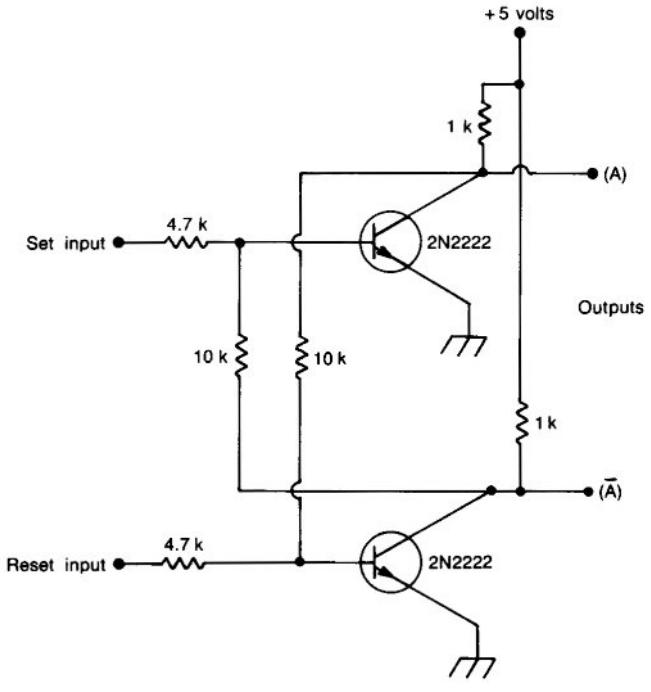


Fig. 15-16. This circuit is a simple transistor memory circuit. The "A" outputs can be toggled off and on by the set input signal and reset input signals.

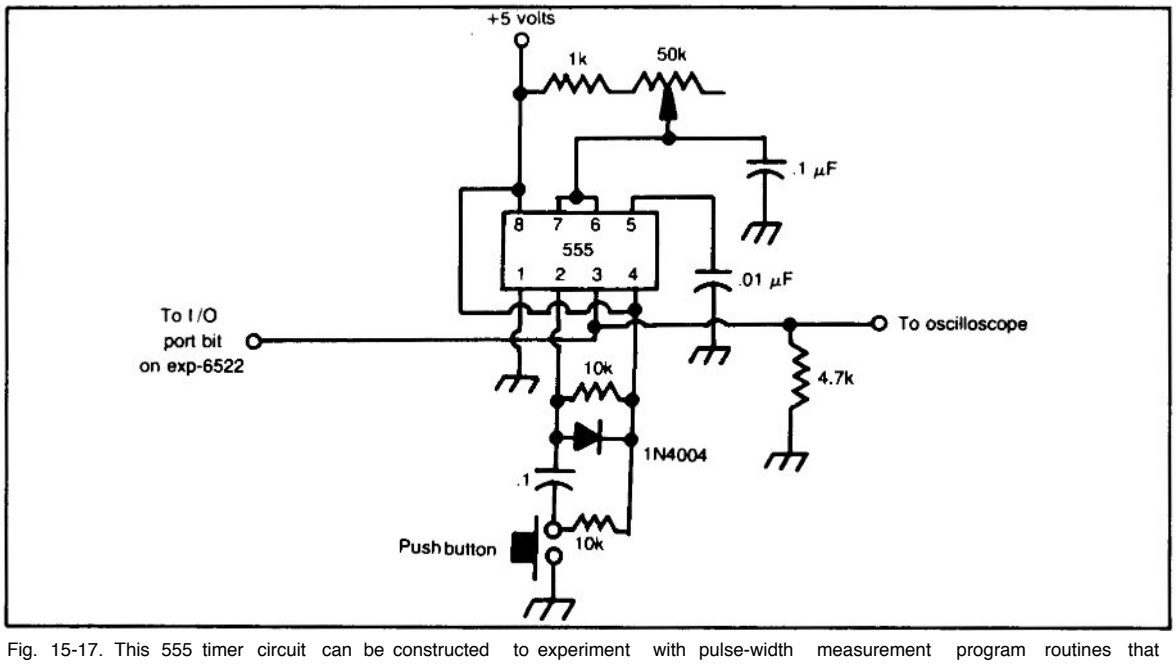


Fig. 15-17. This 555 timer circuit can be constructed to experiment with pulse-width measurement program routines that would be required for Fig. 15-4.

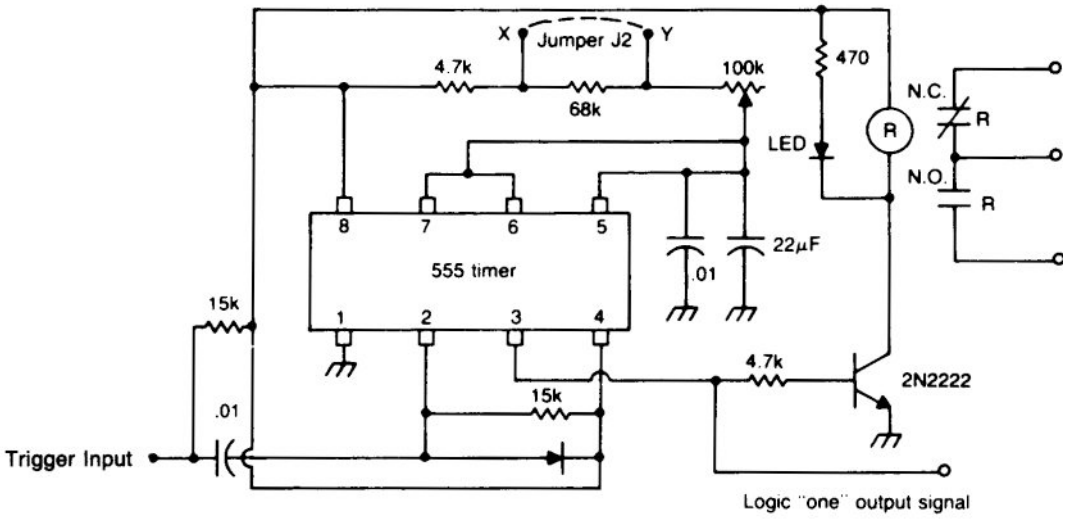
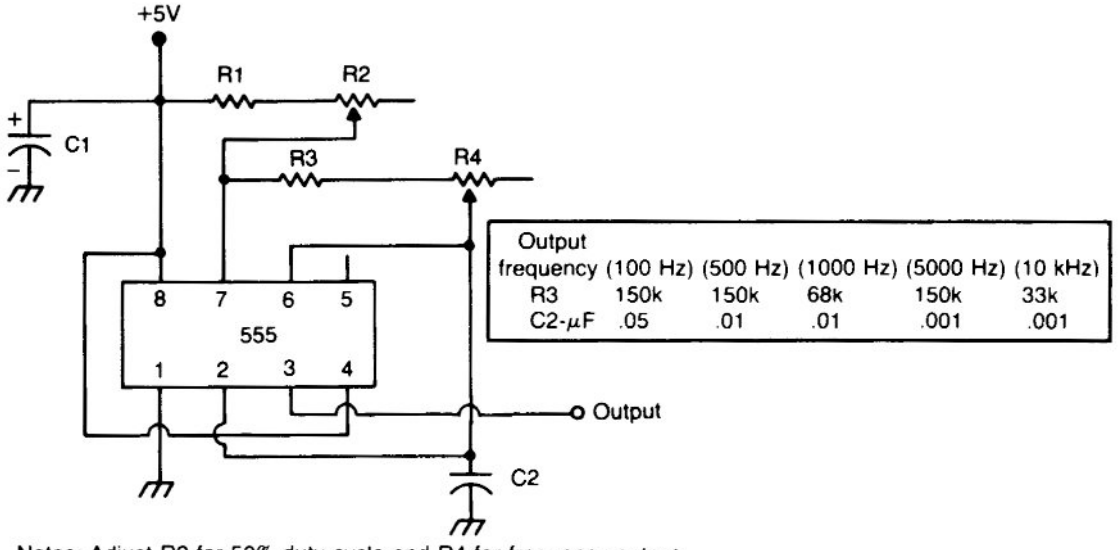


Fig. 15-18. This circuit is a 555 timer relay control circuit. Jumper J2 can be used to shorten the time delay.



Output frequency	(100 Hz)	(500 Hz)	(1000 Hz)	(5000 Hz)	(10 kHz)
R3	150k	150k	68k	150k	33k
C2-μF	.05	.01	.01	.001	.001

Notes: Adjust R2 for 50% duty cycle and R4 for frequency output.

555 Timer - Free Running Oscillator Circuit - Parts List

C1 - 1000 μF at 16 volts R2 - 500 ohms
C2 - See chart R3 - See chart
R1 - 150 ohms R4 - 50k trim pot

Fig. 15-19. This circuit is a 555 timer free-running oscillator circuit with part values for oscillation frequencies between 100 Hz to 10 kHz. One could use this circuit as an external CNT clock signal for the C-64.

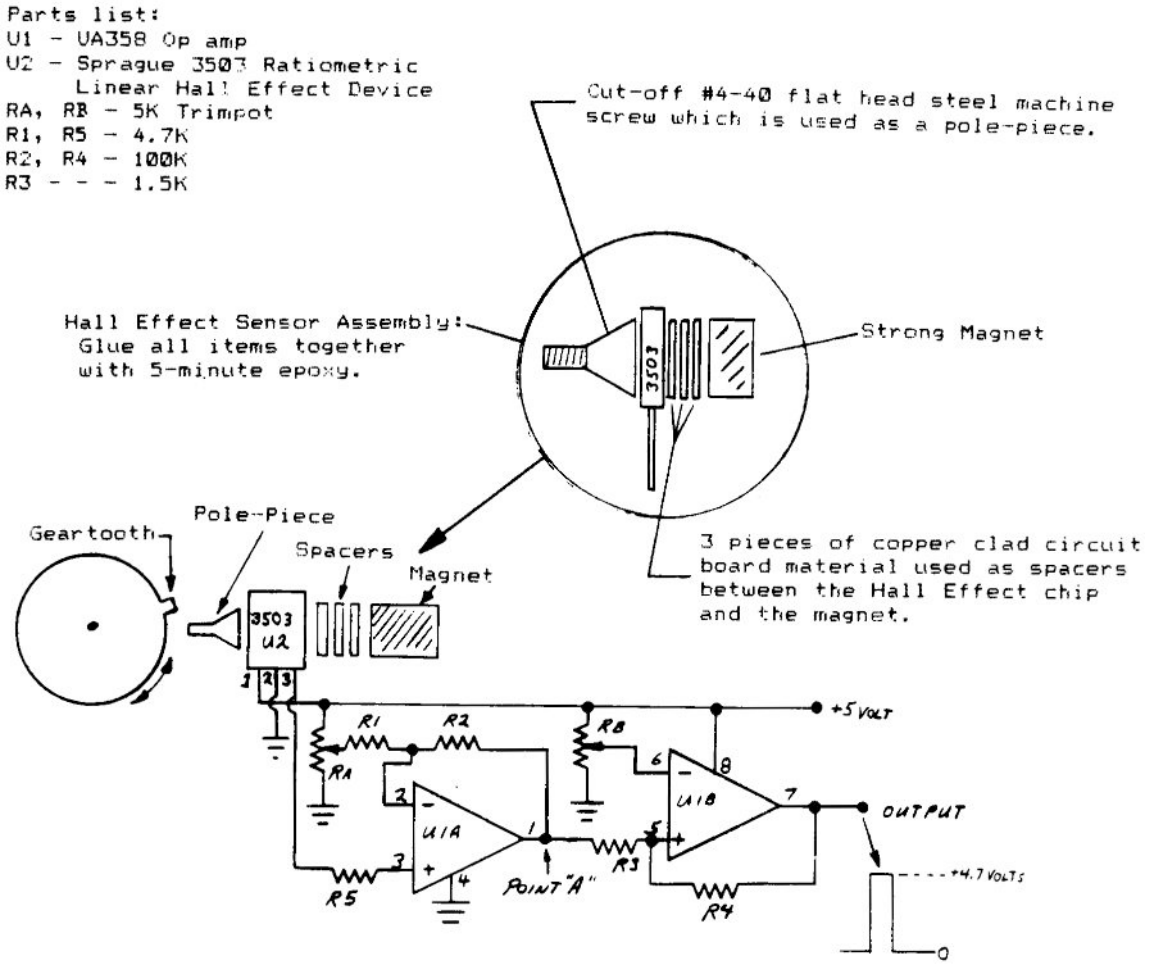
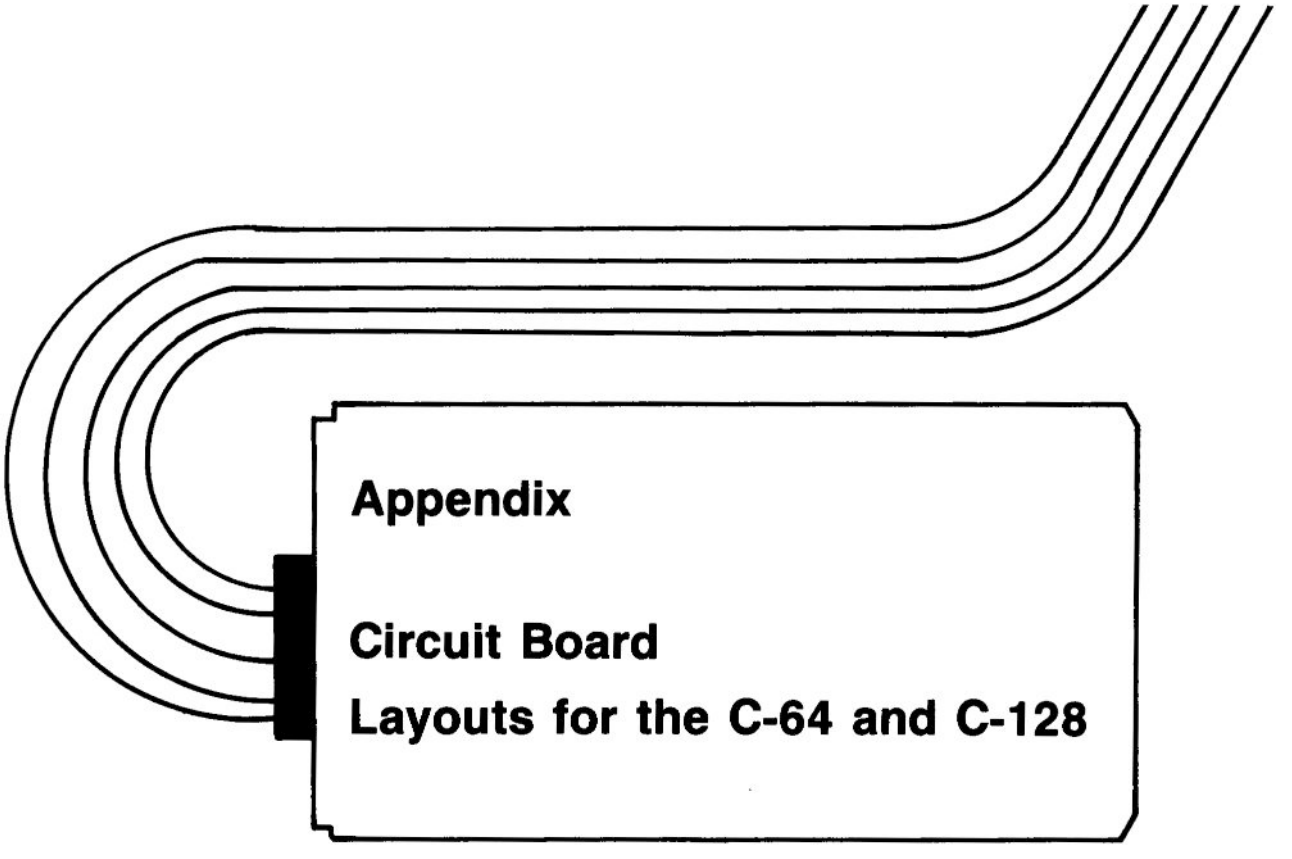


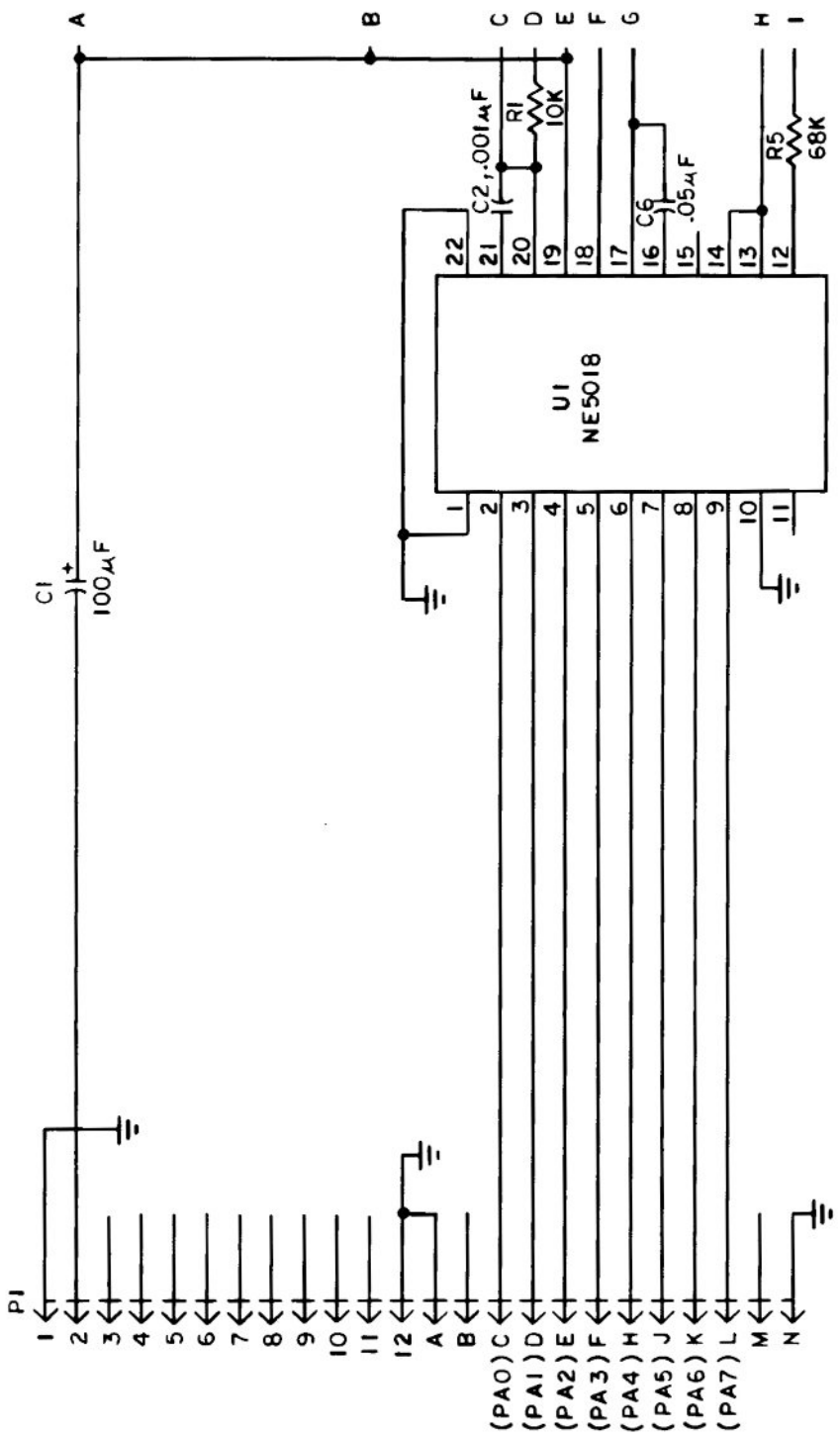
Fig. 15-20. This is a positional magnetic proximity detector circuit. The circuit generates a logic ONE signal when a steel object is held close to the sensor assembly pole-piece. The sensitivity of the circuit is controlled by the size of the pole-piece, the space between the Hall-effect chip and magnet, and the adjustment settings of RA and RB.



Appendix

Circuit Board Layouts for the C-64 and C-128

THE CIRCUIT BOARDS THAT ARE PRESENTED in this Appendix will work with both the Commodore 64 and the Commodore 128.


A
B
C
D
E
F
G
H
I
C1
100μF
C2, .001μF
R1
10K
C6
.05μF
R5
68K
U1
NE5018
P1
(PA0) C
(PA1) D
(PA2) E
(PA3) F
(PA4) H
(PA5) J
(PA6) K
(PA7) L

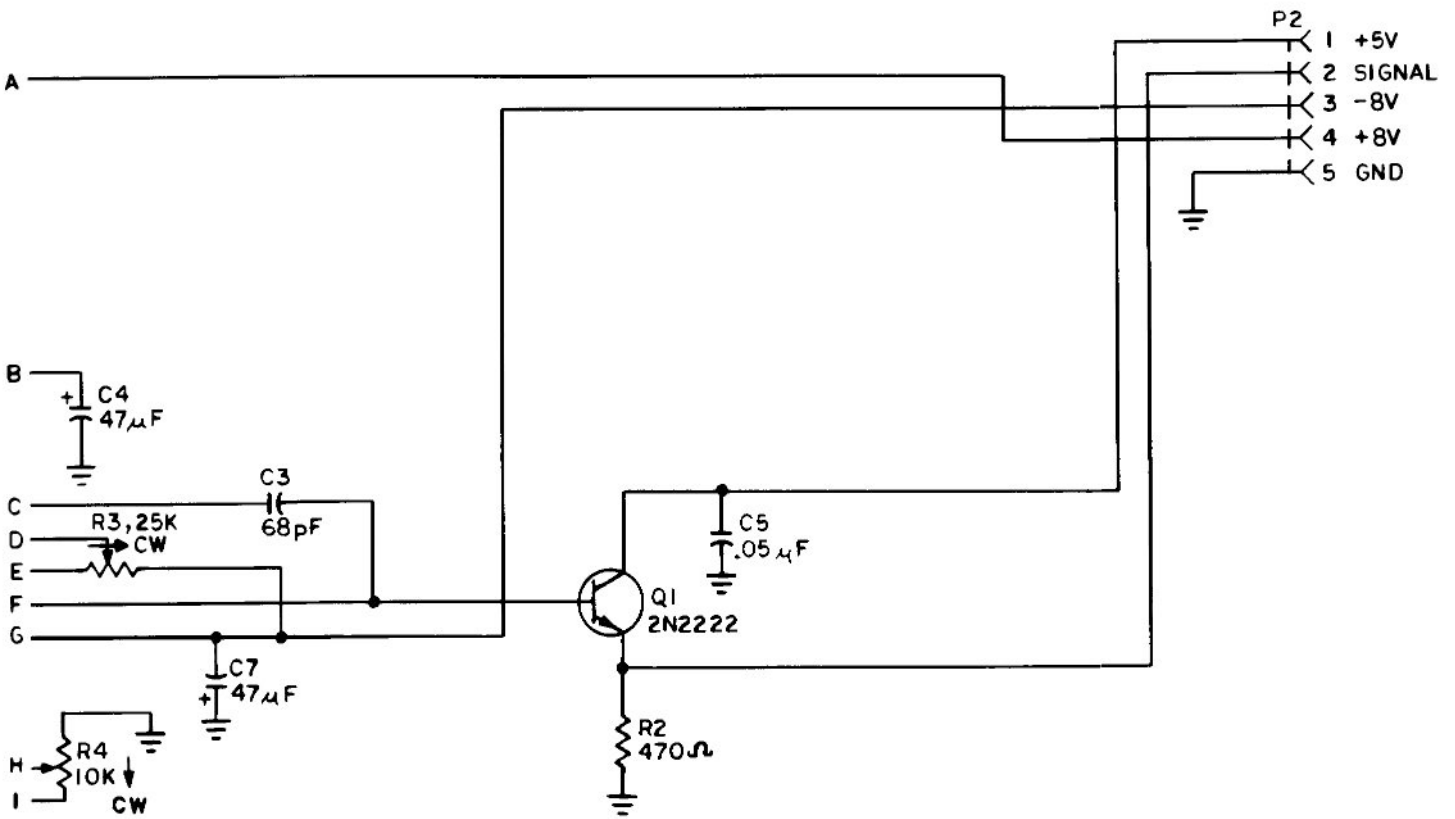


Fig. A-1. This is a redrawn schematic for the D/A converter that was presented in Chapter 5. The part identifications on this schematic match the circuit board layouts that are shown in Figs. A-2 and A-3.

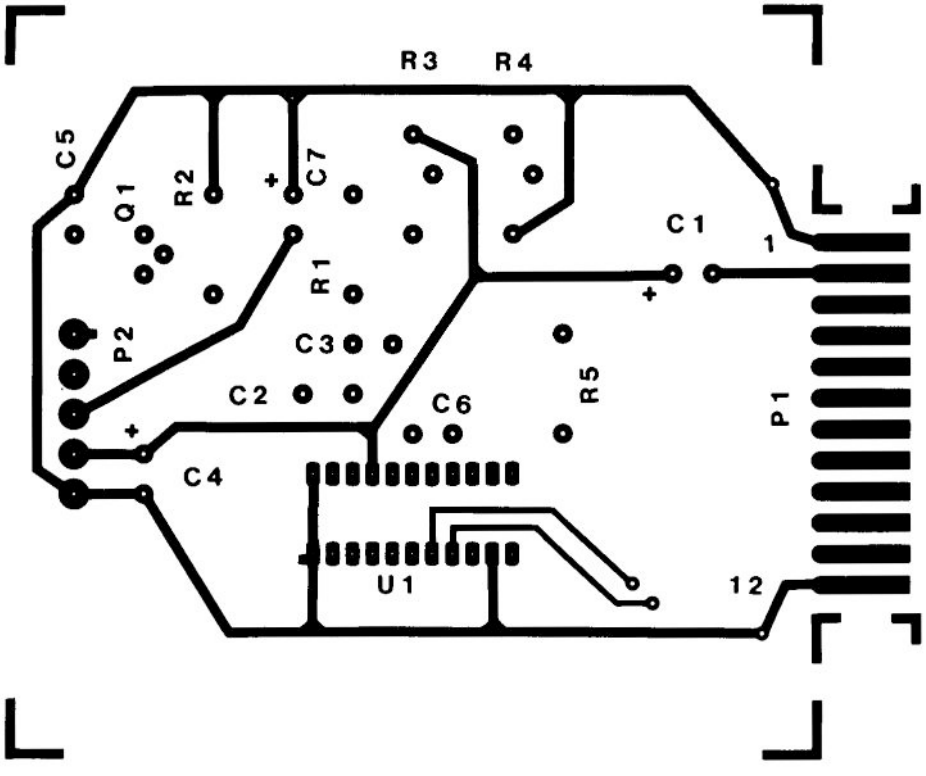


Fig. A-2. This is the circuit board layout for the top-side of the D/A converter board.

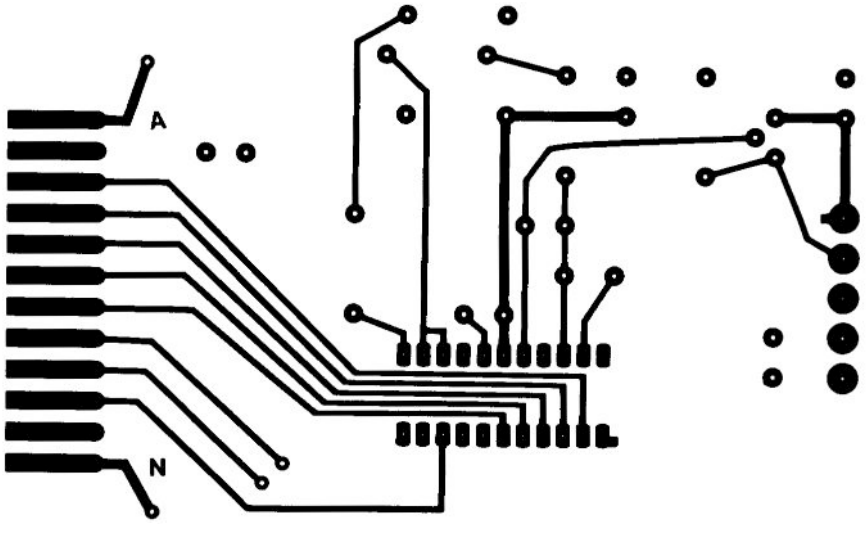


Fig. A-3. This is the circuit board layout for the bottom-side of the D/A converter board.

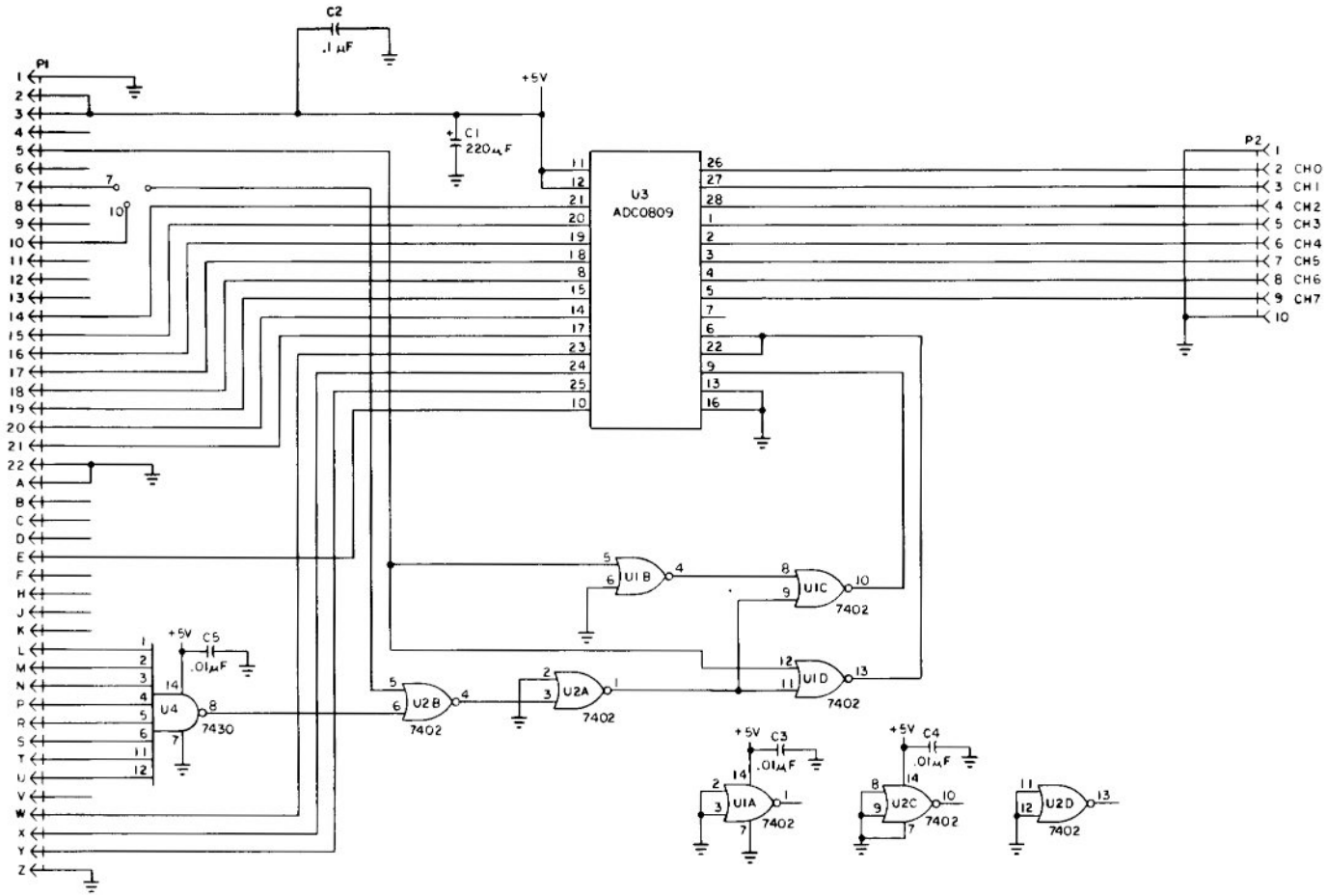


Fig. A-4. This is a redrawn schematic for the A/D converter in Chapter 6. The part identifications on this schematic match the circuit board layouts of Figs. A-5 and A-6. The jumper wire at pins 7 and 10 is used to select the memory address location for this board.

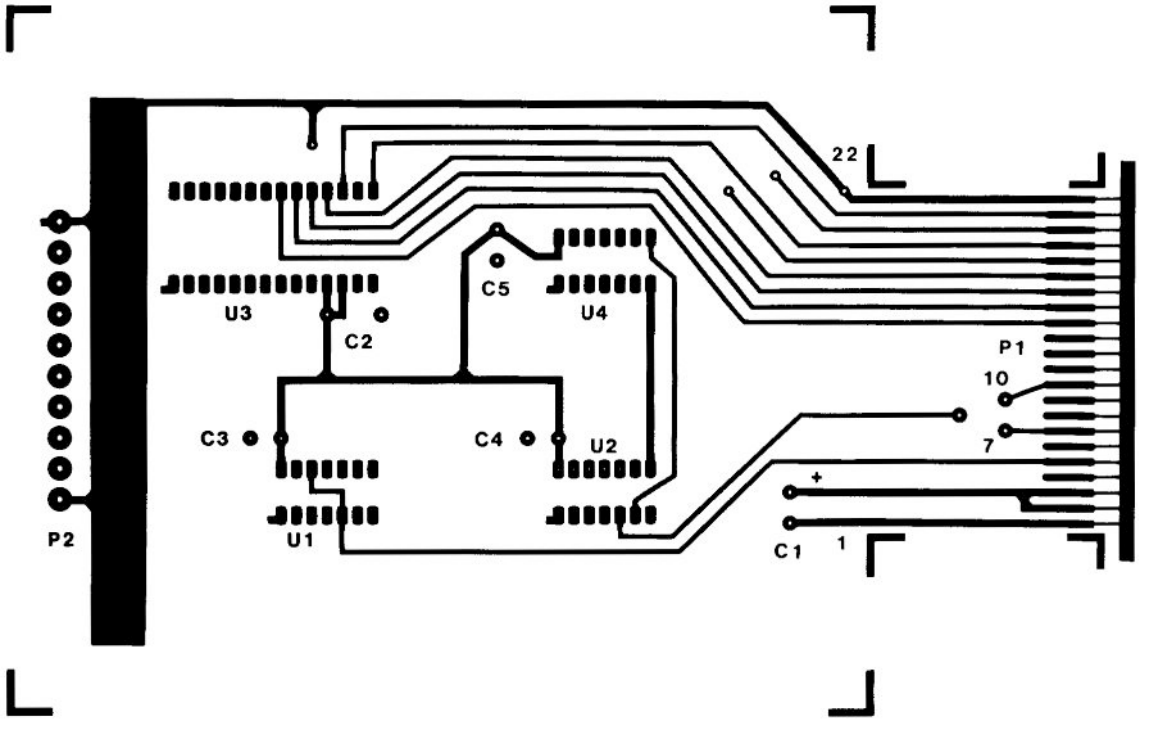


Fig. A-5. This is the circuit board layout for the top-side of the AID converter board (not full size).

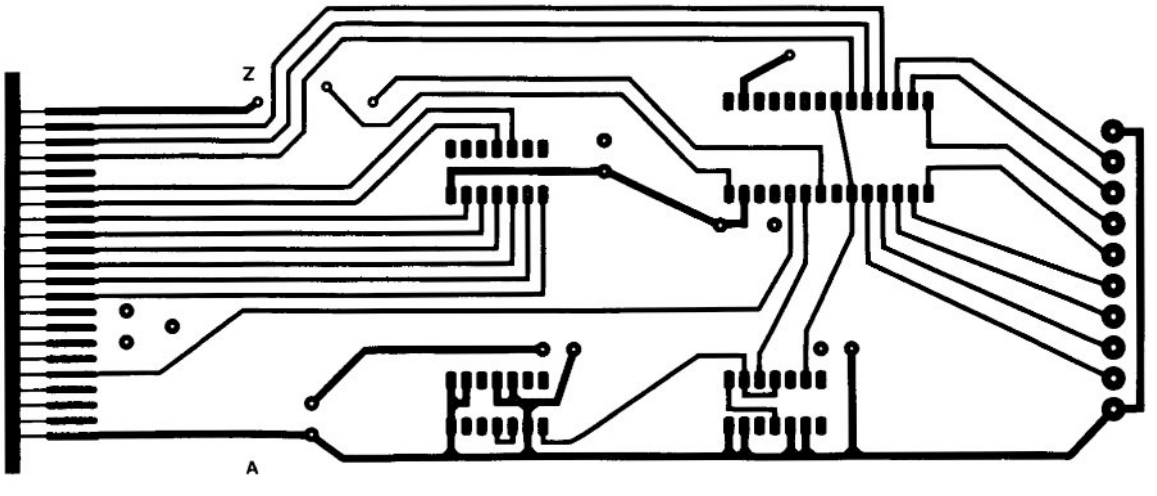


Fig. A-6. This is the circuit board layout for the bottom-side of the AID converter board (not full size).

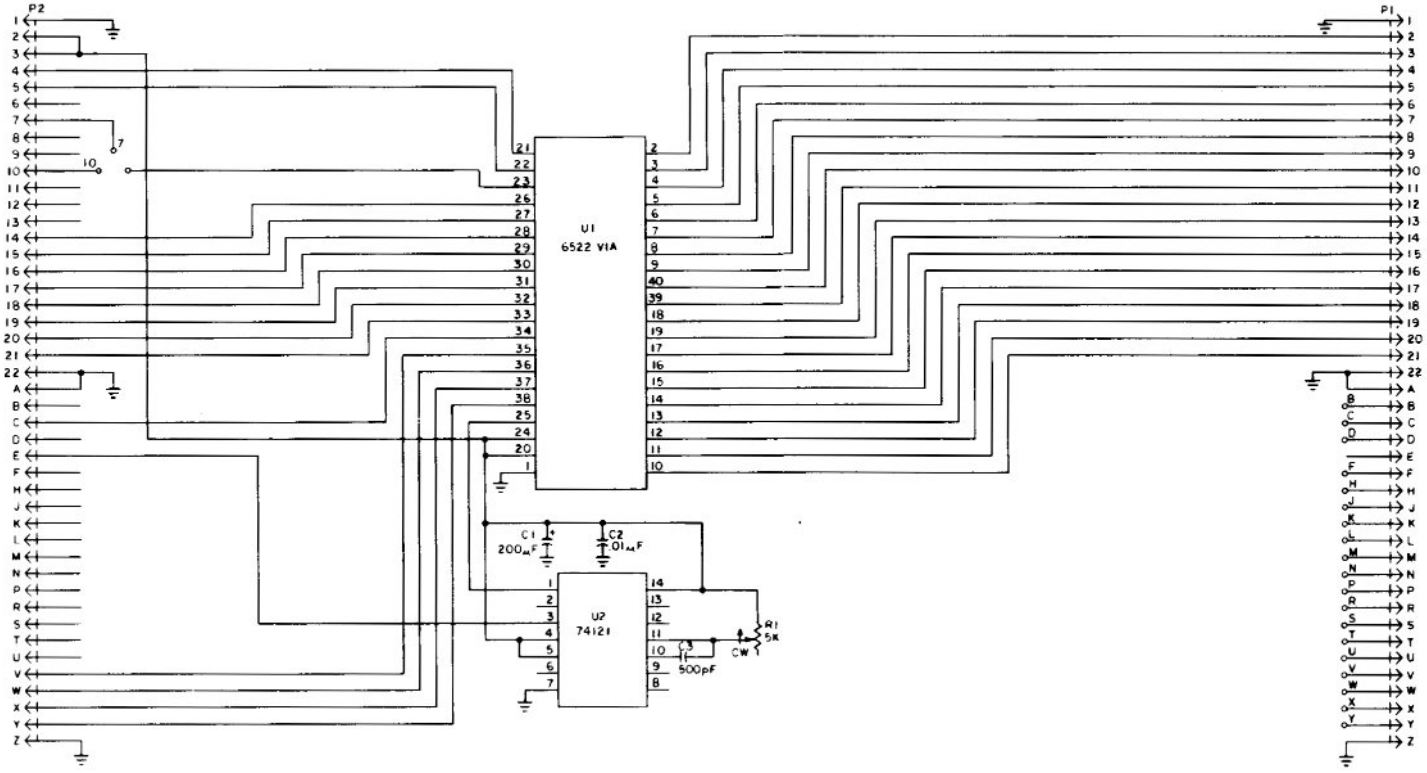


Fig. A-7. This is a redrawn schematic for the 6522 VIA I/O board in Chapter 6. The part identifications on this schematic match the circuit board layouts of Figs. A-8 and A-9. The jumper wire at pins 7 and 10 is used to select the memory address location for this board.

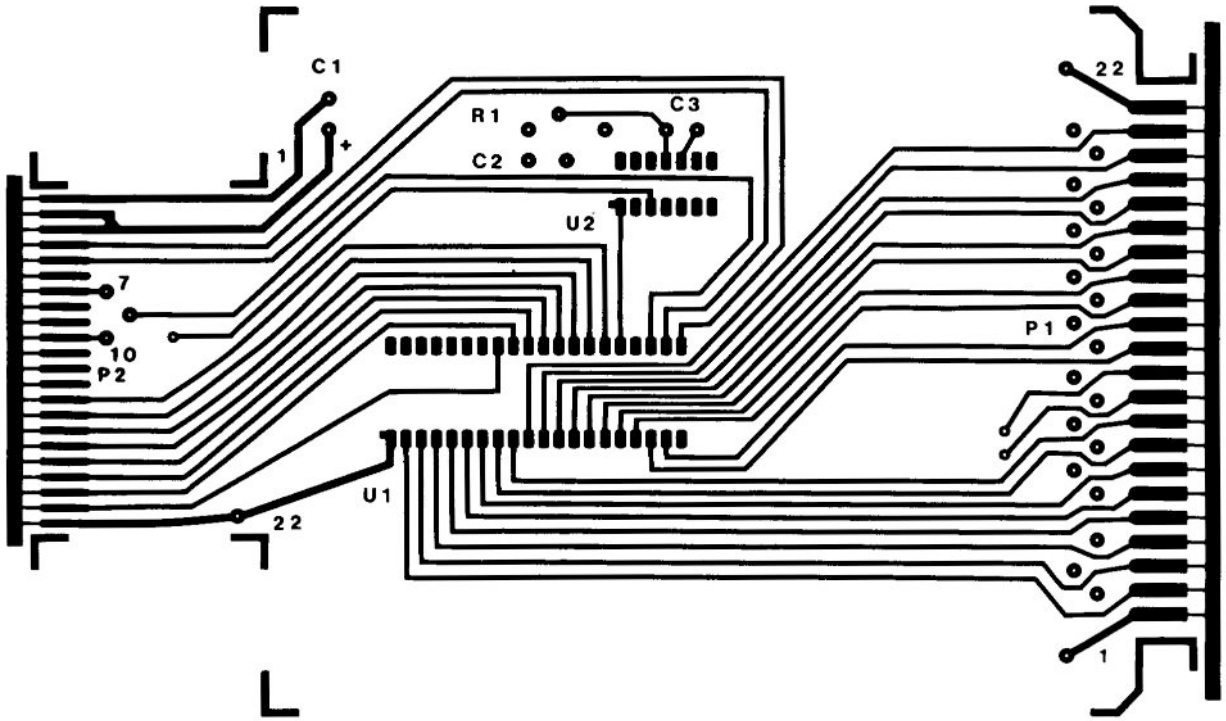


Fig. A-8. This is the circuit board layout for the top-side of the 6522 VIA I/O board (not full size).

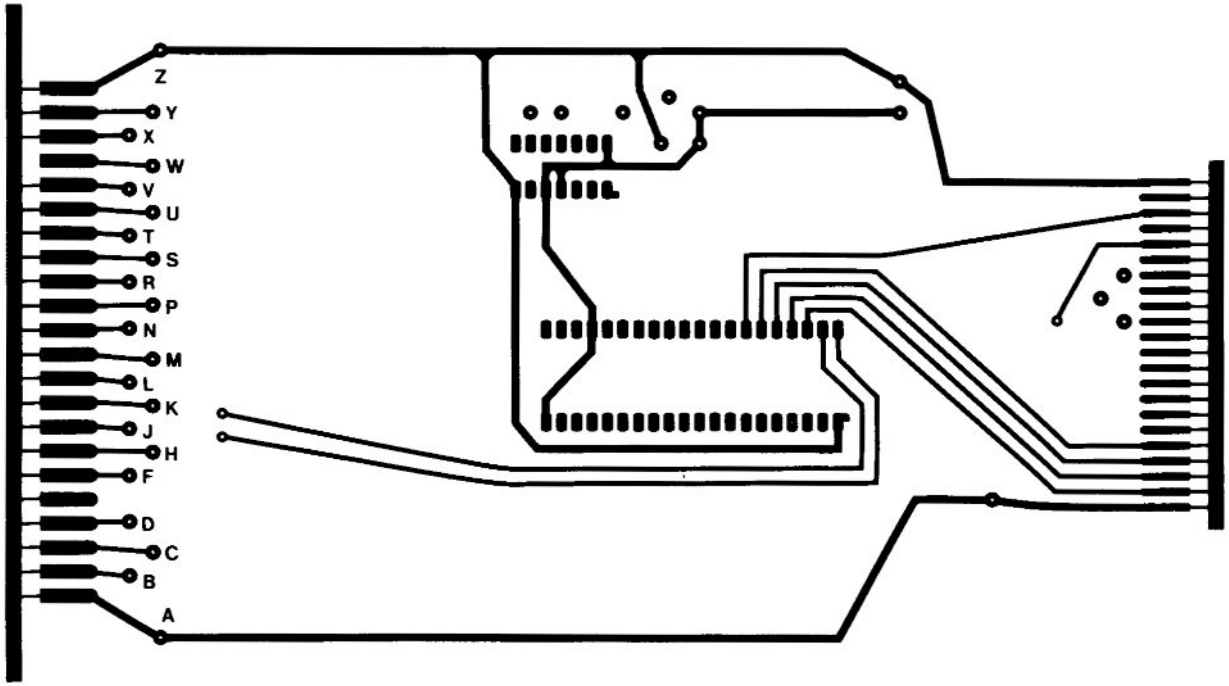
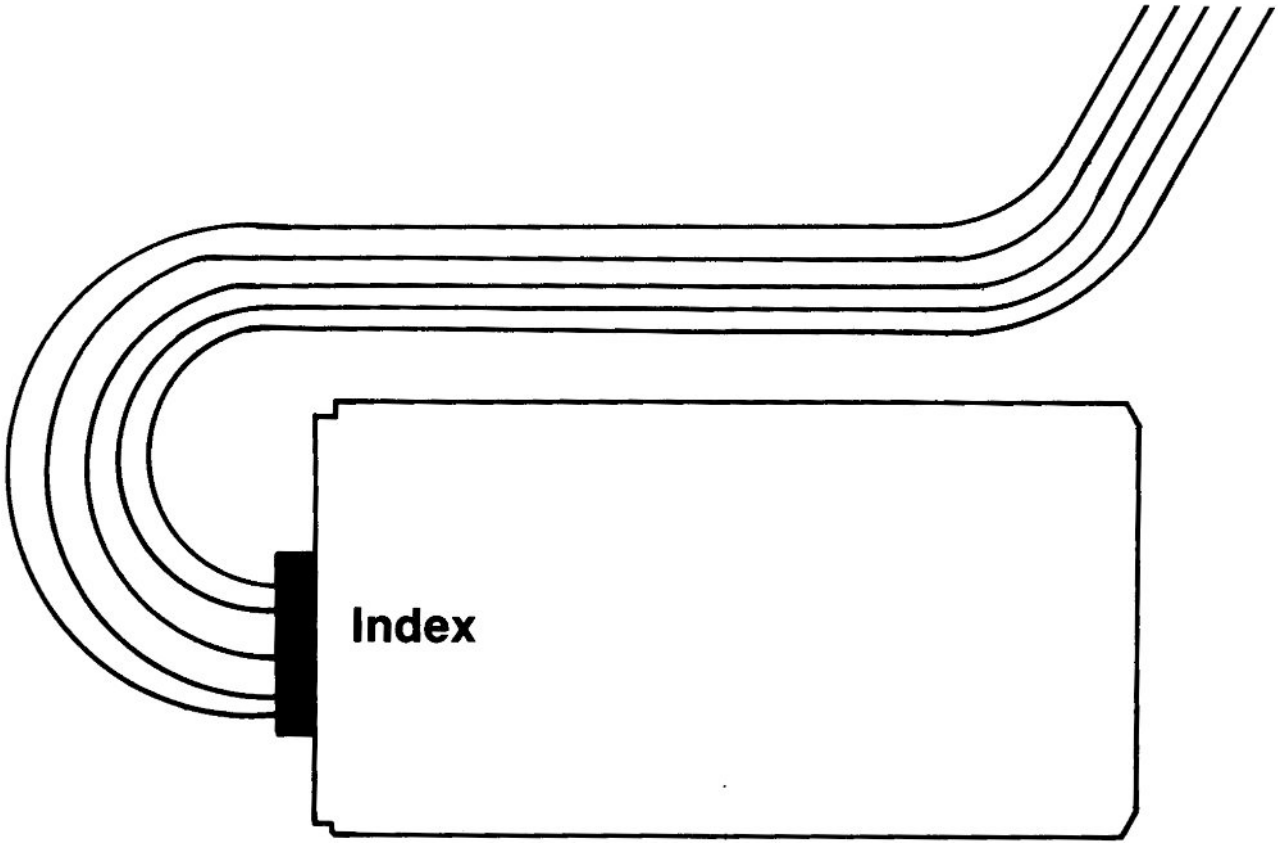


Fig. A-9. This is the circuit board layout for the bottom-side of the 6522 VIA I/O board (not full size).

Index



Index

A

ac bridge circuit, 173
ADC, 63
AID conversion, 62
A/O converter board, 64, 65
AID converter board, C-64, 67, 108
A/O converter board, VIC-20, 66
A/O converter circuit, 94, 117
AID converter circuit building instructions, 69, 70
AID converter circuit diagram, 107
AID converter circuit for the VIC-20, 96
AID test program, 127
address decoder board, 111
address decoder board schematic, 115
adjustable output voltage, 156
adjustable voltage range, 158
aliasing, graphical presentation, 138
analog control projects, 206
analog waveform recorder, 77
audio amplifier, 221
audio-oscillator Circuit, 221
automatic volume control, 206
AVC circuit, 206

B

BASIC language, 41
BASIC memory-expansion modules, 91
BASIC timing programs, 58
BASIC voltmeter programs, 72
BASIC waveform recording control program, 134-136
BASIC waveform recording program, 79-81
bridge-amplifier Circuit, dc, 170
bridge Circuit, capacitor-comparison, 174
bridge circuit, interfacing to a, 168
buffer amplifier circuit, 219

C

card-cage AID converter schematic, 124, 125
card-cage system, 110, 112
circuit board layouts, 231-239
closed-loop motor-speed control, 209
closed-loop servo-control system, 211
CMOS, 16
CMOS inverter buffer, 19
CMOS noninverting buffer, 20
CMOS operating characteristics, 17
Commodore 16, 1
Commodore 64, 1
Commodore 128, viii, 1
computer address locations, 43
continuity checks, 161
continuity test, eight-circuit, 164
control Circuits, 217
C-64 motor-control program, 29
C-64 relay program, 28
C-64 user port program, 26
curve-fitting program, 149

D

DIA Circuit, 227
DIA converter Circuit, 101
O/A converter circuit board, 102, 103
DIA converter circuit for the C-64, 100
O/A converter circuit for the VIC-20, 100
Darlington lamp-driver circuit, 15
Darlington transistor Circuit, 14
data direction register, 4
data register, 4
dc motor control circuit, 223
decoder board test program, 120
digital input data programs, 34

E

electronic measurements, computer-controlled, 161

expansion-port pin assignments for the C-16 and PLUSI4, 111
expansion-port plug-in connector board, 118, 119

F
floating output, 156
Fourier series program, 143
Fourier transform program, discrete, 144
Fourier transform program, fast 144

G
gear tooth sensor, 225
graph-plotting routines, 128

H
Hall-effect sensor, 230
Hall-effect switch, using a, 226
high-resolution graphing program, 130
high-resolution waveform-recording program, 85
hobby circuits, 217

I
IC input chip 74LS244 operation, 123
IC output chip 74LS373 operation, 123
input voltage, 157
instruction set, 6502, 177-205
integrated circuits, I/O interfacing with, 15
interface circuits, discrete transistor, 11
interface Circuits, VIC-20, 91
interfacing circuits, 10
I/O circuits for the C-64, 104
I/O port board, 116
I/O projects, VIC-20, 91
I/O system for the C-16 and PLUSI4, 110

L
LED Circuits, 32
LED display Circuit, 35
LED display programs, 37
LEDs, 17
light-level meter, 39
light-level meter circuit, 38
light-level meter program, 38
line regulation, 157
load regulation, 157

M
machine language, 41
machine language, 6502, 41
machine-language instructions, 48
machine-language monitor, 42
machine-language nomenclature, 51
machine-language time-delay subroutines, 57
machine-language timing subroutines, 59
memory map, 47
metal-film resistors, 157
motor-control circuit, 30
motor-speed control, closed-loop, 209
motor-speed control, open-loop, 206
motor-speed control circuit, 207

N
null condition, 169
number tables, 44-46

O
op-amp circuit board, 74, 75
op-amp circuit building instructions, 71
op-amp power booster circuit, 219
opcodes,41
optical sensing circuit, 224
oscillator circuit, crystal-controlled, 222
oscillator, CMOS crystal-controlled, 222

P
PEEK, 5, 6
pendulum, view of the, 139
pendulum positional data, 141
plotting, high-resolution, 129
plotting routines, low-resolution, 128
PLUSI4, 1
PLUSI4 motor-control program, 29
PLUSI4 relay program, 28
PLUSI4 user port program, 27
POKE routines, 49
position sensor Circuit, 224
power supplies, 154
power supply construction projects, 159
program experiments, 25
projects, special, 58
push-to-test routine, 163

R
RAM,93
relay-control circuit, 30
resistance-checking program, 169
response time, 158
rf detector circuit, 218
ripple, 157

S
Schmitt trigger, 18
servo control program, closed-loop, 214
servo motor-control circuit, 212
servo test program, open-loop, 213
short-circuit protection, 157
signal analysis, 143
Simon's high-resolution graphics, 78
sine-wave circuit, 60 Hz, 172
sine-wave Circuit, 1000 Hz, 171
6502 instruction set, 177-205
6522 VIA circuit board schematic, 105
switch circuits, 33

T
temperature coefficient, 158
temperature-compensated circuitry, 155
temperature rating, 158
time-delay subroutine, 54
time-interval programs, 60
timer circuit, 555, 228, 229
timing programs, 58
touch-plate circuit, 220
transistor specifications, 12, 13
TTL,11
TTL circuits, 13
TTL inverting buffer, 21, 22
TTL operating characteristics, 16

U
UEB, 7
UEB-1, bottom view of, 9
UEB-1, top view of, 8
UEB-2, top view of, 10
universal-input power supply, 155
user port, 1, 7, 31
user port experimenter's boards, 7

V
VIA circuit, 6522, 104
VIA I/O chip, 91
VIA I/O chip, adding an extra, 94
VIC-20, 1
VIC-20 expansion port data, 92
VIC-20 motor-control program, 29
VIC-20 relay program, 28
VIC-20 user port program, 27

W
waveform recorder printout, 3, 142
waveform-recording printouts, 87
waveform recording program, 133
waveform recording system, 134
window comparators, 164
window comparator, using the, 167
window comparator circuit, 220
wire-wound pots, 157

Edited by Roland S. Phelps

Other Bestsellers from TAB

O LISP-THE LANGUAGE OF ARTIFICIAL INTELLIGENCE-Holtz

Now here's your opportunity to learn and use LISP ... to enter the realm of artificial intelligence with confidence. Holtz explains LISP vocabulary and how artificial intelligence programming concepts differ. You'll get a look at how LISP handles mathematical operations ... be introduced to logical operators ... and see how close BASIC input/output is to that of Common LISP. 272 pp., 7" x 10".

Paper $16.95 Book No. 2620
Hard $25.95

O 469 PASCAL PROBLEMS WITH DETAILED SOLUTIONS-Veklerov

Now this unique self-teaching guide makes it amazingly easy even for a novice programmers to master Pascal. With a total of 469 problems ranging from the most basic to advanced applications, the guide provides a unique learning opportunity for anyone who wants hands-on understanding of the Pascal language and its programming capabilities. 224 pp., 23 illus. 7" x 10".

Paper $14.95 Book No. 1997
Hard $21.95

O TRUE BASIC® -A COMPLETE MANUAL

Written by microcomputer programmer and consultant Henry Simpson, this invaluable guide covers all the main features of True BASIC including commands, statements, and functions, program control, input/output, file-handling, and even graphics. Simpson even supplies you with example programs that demonstrate all the programming functions that can be performed by True BASIC as opposed to Microsoft BASIC. 208 pp., 53 illus. 7" x 10".

Paper $14.95 Book No. 1970
Hard $22.95

O COMMODORE 64™ EXPANSION GUIDE-Phillips

Far more than just a product listing or a rehash of manufacturers' sales brochures, these are the best of the hundredS of hardware accessories currently on the market . . . each one chosen for value and performance after exhaustive testing and examination. You'll find in-depth background on each type of device-printers, disk drives, modems, monitors, and photographic details. 288 pp., 31 illus. 7" x 10".

Paper $16.95 Book No. 1961
Hard $22.95

O THE COMPUTER SECURITY HANDBOOK-Baker

Electronic breaking and entering into computers systems used by business, industry and personal computers has reached epidemic proportions. That's why this up-to-date sourcebook is so important. It provides a realistic examination of today's computer security problems, shows you how to analyze your home and business security needs, and gives you guidance in planning your own computer security system. 288 pp., 61 illus. 7" x 10".

Hard $25.00 Book No. 2608

O TRUE BASIC® PROGRAMS AND SUBROUTINES

Explore the powerful, built-in features of True BASIC-a new language that Is destined to standardize microcomputer programming. Now professional programmer and consultant John Clark Craig shows you hands-on how True BASIC can make your programming easier and less time-consuming than traditional languages. You'll discover the features that make True BASIC unmatched: coherent syntax, compiled operating speed, greatly improved graphics capabilities, structured language features, and portability. 224 pp., 50 illus. 7" x 10".

Paper $16.95 Book No. 1990
Hard $24.95

O MAKING MONEY WITH YOUR MICROCOMPUTER-2nd Edition

Let your PC pay for itself by putting it to work in your own profitable part-time business. This *newly* revised, expanded, and updated idea book is overflowing with practical, proven business suggestions for getting started. Plus you'll find sources for software needed to get started. From setting up your office to locating tile best market, all the factors that equal success are provided. 208 pp., 78 illus.

Paper $10.95 Book No. 1969
Hard $16.95

O 101 PROGRAMMING SURPRISES AND TRICKS FOR YOUR COMMODORE 64 COMPUTER

This exciting new collection of games, novelties, and programming marvels is fresh, literate, and packed with all kinds of downright amazing ways to have fun with your C-64. And unlike other programming books, it makes no attempt to instruct you-instead, the object is to entertain and be entertained. 224 pp., 12 illus. 7" x 10".

Paper $11.95 Book No. 1951
Hard $18.95

Other Bestsellers From TAB

O ONLINE RESEARCH AND RETRIEVAL WITH MICROCOMPUTERS-Goldmann

This time-saving guide shows you how to turn a micro into an invaluable research "tool" for digging up information from databases across the country. Using Nahum Goldmann's "Subject Expert Searching Technique," businessmen, engineers, physicians, lawyers, professors, and students can be quickly and easily retrieve information in the comfort of their work station. 208 pp., 119 illus. 7" x 10".

Hard $25.00 Book No. 1947

O COMMODORE 64™ ASSEMBLY LANGUAGE ARCADE GAME PROGRAMMING-Bress

With the help of this outstanding guide, you'll be able to create, plan, code, and test virtually any fast-paced action game you can think of using assembly language techniques-and you'll be able to really take advantage of the power and amazing potential of your C-64. Plus, you'll be a better programmer in all areas! 272 pp., 148 illus. 7" x 10".

Paper $14.95 Book No. 1919

O COMPUTER USER'S GUIDE TO ELECTRONICS -Margolls

Assembly language will be easy to learn you'll be able to do your own interfacing with peripherals and most importantly, you can perform simple repair procedures yourself. In fact, the savings realized by changing one bad chip yourself will more than pay for this invaluable handbook! 320 pp., 250 illus. 7" x 10".

Paper $15.95 Book No. 1899
Hard $24.95

O INTERFACING YOUR MICROCOMPUTER TO VIRTUALLY ANYTHING-Carr

Here, at last, is a sourcebook that's literally packed with practical interfacing techniques, plus a wealth of useful projects. Vou'll find projects such as a single-ended amplifier, a differential amplifier, a universal rear-end, and a preciSion 10·volt reference power source. All of these building block circuits can be used in a variety of applications. 336 pp., 212 illus.

Paper $13.95 Book No. 1890
Hard $21.95

O COMMODORE 64™ ADVANCED GAME DESIGN-Schwenk

Professional game designers George and Nancy Schwenk reveal their winning formula for creating stimulating, professional-quality microcomputer games for family fun and even prOfit! Using three fully-developed C-64 games to illustrate game design, this innovative tutorial provides an informative and practical look at the conceptual and implementation techniques involved. 144 pp., 14illus. 7" x 10".

Paper $10.95 Book No. 1923
Hard $15.95

O COMPUTER COMPANION FOR THE COMMODORE 64™-Haviland

An essential programming tool for every C-64 user! Arranged alphabetically, it provides instant access to BASIC keyword names, tokens used for internal storage, the class of Instruction, the required form, and conditions and results of use. In addition, the author has included examples, error messages, and helpful cautions and warnings on their use. 160 pp., Heavy Card Stock with Comb Binding.

Paper $11.95 Book No. 1913

O PERFECT PASCAL PROGRAMS

Much more than just another collection of programs, this is a compilation of articles by Pascal experts who are members of Washington Apple Pi-the nation's second largest Apple® user's group. Covering applications from simple utility routines to advanced programming techniques, it's the perfect tool for Apple users *and* for Pascal programmers using almost any micro (Pascal features easy transportability). 288 pp., 13 illus. 7" x 10".

Paper $16.95 Book No. 1894
Hard $22.95

O TROUBLESHOOTING & REPAIRING YOUR COMMODORE 64™-Margolls

Written by computer expert Art Margolis, this exceptionally well-illustrated manual covers both simple chip-changing techniques (the cause of about 50% of micro breakdowns) and in-depth servicing *data-includlng a detailed Master Schematic of your machine that contains all the parts numbers.* So whatever the problem, you will be able to troubleshoot and repair it yourself. 368 pp., 291 illus. 7" x 10".

Paper $14.95 Book No. 1889

Other Bestsellers From TAB

O ARTIFICIAL INTELLIGENCE PROJECTS FOR THE COMMODORE 64™

This uniquely-exciting guide includes 16 ready-to-run, full-explained projects illustrating a wide variety of artificial intelligence techniques; a plain-English Introduction to artificial intelligence, robotics, and LISP; a complete glossary of artifiCial intelligence terminology; easy-tO-follow examples and Show-how illustrations; plus a quick-look-up index for fast reference. 160 pp., 15 illus. 7" x 10".

Paper $12.95 Book No. 1883

O THE BASIC COOKBOOK-2nd Edition

Covers every BASIC statement, function, command, and keyword in easy-to-use dictionary form-highlighted by plenty of program examples-so you can cook up a BASIC program in just about any dialect, to do any job you want. Whether your interests are business, technical, hobby, or game playing, this revised 2nd edition of our all-time bestselling BASIC guide contains exactly what you want, when you want it! 168 pp., 57 illus.

Paper $7.95 Book No. 1855
Hard $12.95

O COMMODORE 64™ PROGRAMMING; A HANDS-ON APPROACH TO BASIC

Here's a user-friendly guide for computerists who want to get the most from the C-64's features and capabilities from programming concepts to editing procedures and program debugging. It gives practical insight Into how programs are developed, tips and "tricks" used by profeSSionals, and techniques for writing original software that is crashproof! 192 pp., 69 illus. 7" x 10".

Paper $9.95 Book No. 1831

O DEBUGGING BASIC PROGRAMS-Cecll

With the expert advice provided by this guidebook, you'll be amazed at how eesy it is to write bug-free programs ... and to trap those errors that do creep in! Vou'll discover that the most common errors are in syntax, assignment, and placement of program statements-then find detailed techniques for correcting them. Vou'll get all the how-to's for finding the correcting errors in arithmetic, strings, input/output, even disk errors. 176 pp., 20 illus. 7" x 10".

Paper $9.95 Book No. 1813

O FROM FLOWCHART TO PROGRAM-Todd

Master the skills of effective, "bug-free" programming with this practical approach to program design and development. Using the flowcharting system of structuring a program, you'll learn step-by-step how to break down the problem logically, enabling you to tackle even large-scale programs with confidence. It's your key to writing programs, that are easier to debug, easier to change and expand, easier to understand, and faster in execution. 192 pp., 190 illus. 7" x 10".

Paper $12.95 Book No. 1862
Hard $19.95

O 1001'THINGSTO DO WITH YOUR COMMODORE64

Here's an outstanding sourcebook of microcomputer applications and programs that span every use and interest from game playing and hobby use to scientific, educational, financial, mathematical, and technical applications. It provides a wealth of practical answers to the question-what can my Commodore 64 computer do for me? Contains a goldmine of actual program! 256 pp., 47 illus.

Paper $10.95 Book No. 1836

O SERIOUS PROGRAMMING FOR THE COMMODORE 64-Simpson

Serious programming means writing programs that are user friendly, well documented, and designed to take full advantage of all of the resources offered by the BASIC language, your DOS (disk operating system), and assembly language routines. And that's what you'll find here-everything you need to get more programming power from your C-64! 208 pp., 124 illus. 7" x 10".

Paper $9.95 Book No. 1821
Hard $15.95

O SECRETS OF SOFTWARE DEBUGGING

How to find and eliminate the errors that keep computer programs from working the way you want them to! This excellent learn-by-doing guide even shows how to keep those bugs from happening in the first place! Vou'lllearn step-by-step, how to logically identify your problem, how to figure out what's wrong so you can fix it, and how to get your program up and running like it should using tested & proven techniques from an expert in the field! 288 pp., 58 illus. 7" x 10".

Paper $13.95 Book No. 1811
Hard $21.95

Other Bestsellers From TAS

O INCREASING PRODUCTIVITY WITH PFS® -Burton

Here's an important money-saving guide that provides expert guidance on the most effective ways to use all six PFS modules to increase efficiency and productivity in your business, nonprofit agency, or service organization ! Plus there are 10 time- and money-saving templates or applications models that make it possible for you to prepare more than 20 specific forms! 192 pp., 92 illus. 7" x 10".

Paper $14.95 Book No. 1789
Hard $21.95

DUSING. PROGRAMMING THE COMMODORE 64, INCLUDING READY- TO-RUN PROGRAMS-Herriott

If you own or use a Commodore 64, here's a book that you will want to keep permanently next to your micro. This is the key to unlocking the enormous potential of the C-64 and using it to your best advantage! Set up and change fascinating color, exciting original graphics, even make music with your micro!Includes ready-to-run programs!176 pp., 7" x 10".

Paper $9.25 Book No. 1712

O COMMODORE 64 GRAPHICS AND SOUND PROGRAMMING

Here's a hands-on, learn-by-doing approach to mastering the full graphics potential offered by your Commodore 64 (sprite, character, and bit mapped graphics), *plus* learning to take fullest advantage of the machine's remarkable built-in three-voice music synthesizer chip ... using this collection of 68 fascinating programs developed by the author. 256 pp., 300 illus. 7" x 10".

Paper $15.50 Book No. 1640

O COBOL-Jackson

Vou can be writing and running programs in COBOL (COmmon Business Oriented Language) in a matter of hours with this handy, plain-English reference guide. All you need to know are the fundamentals of computer programming and how to use a standard computer keyboard. *Every* phase of the language is covered ! If you're serious about learning computer business language, then this introduction to COBOL is a must! 300 pp., 32 illus.

Paper $10.25 Book No. 1398

O BASIC STATISTICS-AN INTRODUCTION TO PROBLEM SOLVING WITH YOUR PERSONAL COMPUTER

Here is an introduction to BASIC programming that uses statistical problem-solving techniques and programs as the basis for mastering this popular high-levellanguage. It's also an introduction to statistics that provides you with all the fundamental concepts needed to easily find solutions to a wide variety of mathematical problems. 462 pp., 325 illus.

Paper $15.95 Book No. 1759

O PERSONAL MONEY MANAGEMENT WITH YOUR MICRO-Milner

Here, at last, Is a collection of personal money management programs that are practical, useful tools ... programs that, for once, aren't more complicated and time-consuming than the old paper-and-pencil method! Vou'll be amazed at how Simple it is to put any household on the road to sound money management using an Apple II/lie, or other home computer. 240 pp., 77 illus. 7" x 10".

Paper $13.50 Book No. 1709
Hard $18.95

O BASIC COMPUTER SIMULATION-McNitt

Use your computer to find answers to questions that simply don't have neat, easily found solutions. Now, this exceptional sourcebook introduces you to the how-to's of modeling and creating simulations, and programs written in a universal subset of BASIC that can be used on any BASIC microcomputer. Loaded with easy-to-follow explanations, detailed illustrations, and specific programming examples ! 352 pp., 63 illus. 7" x 10".

Paper $15.50 Book No. 1585

***Prices subject to change without notice.**

Look for these and other TAB BOOKS at your local bookstore.

TAB BOOKS Inc.
P.O. Box 40
Blue Ridge Summit, PA 17214

Send for FREE TAB Catalog describing over 900 current titles in print.

ISBN 0-8306-0983-0