APPLE INFORMATION SERIES - VOLUME I

Winfried Hofacker . Ekkehard Floegel

the custom apple®

& OTHER MYSTERIES

A guide to customizing **APPLE** software and hardware

APPLE INFORMATION SERIES - VOLUME I

Winfried Hofacker . Ekkehard Floegel

the custom apple®

& OTHER MYSTERIES

Charles Trapp — Editor

David Moore — Technical Editor

D. J. Smith — Additional Graphics

H. C. Pennington and D. J. Smith — Cover Design

First Edition
First Printing
June 1982
Printed in the United States of America

ISBN 0 936200 05 7

Published by
IJG Inc
1953 West
11th Street
Upland, CA
91786 (714)
946-5805

IMPORTANT

Read This Notice

Any software or computer hardware modifications are done at your own risk. Neither the PUBLISHER nor the AUTHOR assumes any responsibility or liability for loss or damages caused or alleged to be caused directly or indirectly by applying any modification or alteration to software or hardware described in this book, including but not limited to any interruption of service, loss of business, anticipatory profits or consequential damages resulting from the use or operation of such modified or altered computer hardware or software. Also, no patent liability is assumed with respect to the use of the information contained herein.

While every precaution has been taken in the preparation of this book, the PUBLISHER and the AUTHOR assume no responsibility for errors or omissions.

The reader is the sole judge of his or her skill and ability to perform the modifications and/or alterations contained in this book.

Editor's Note

ABOUT THE AUTHORS

The authors have previously published several books in Germany, including: *Programming in Basic and Machine Language with the ZX81, Pascal Handbook Programming in Machine Language with the 6502* (by E. Floegel), as well as *Transistor Logic and Construction Handbook* (2 volumes) and *Basic for Laymen* (by W. Hofacker). In addition, Winfried Hofacker operates a publishing firm (with offices in Holzkirchen, Bavaria and Los Angeles) specializing in computer books written in both English and German.

This book is the first to be written in English by the two authors, and it had a spectacularly unlucky beginning as a result. Several chapters were composed verbally on the spot by the two authors in German, then dictated in literal English-German translation to the technical editor, who in turn, dictated into a cassette. Some weeks later the cassettes were typed into disk files by a person unfamiliar with computers, and the resulting manuscript given to myself was really unbelievable.

I'd like to say an especial word of thanks to Ekkehard Floegel, who spent a week helping me with the "too long German sentences" and numerous technical points, as well as his interesting stories of Bavaria.

I'd also like to thank:
David Moore, for teaching me to speak English again when this project was completed;
Nancy DeDiemar of Helen's Place and Eric Jorgensen of Clymer Publications, for their useful suggestions and the 'TLC' they gave our typesetting;
Muriel Brock, for laying out the majority of the book;
and all the fine folks of IJG, Inc., for helping in so many ways.

Charles Trapp
June, 1982

Table of Contents

General Information:

The Tools You Will Need 7
Reading Schematics 9
Be Tolerant 14
Those Colors:
What They Mean and How to Read Them 15
Copacetic Comprehension 17
Number Systems 17
Converting Binary to Decimal 20
Digital Logic Devices 21
Reading the Pins 25
Wire-Wrapping Technique 26
Soldering Technique 28
Tips on Handling Integrated Circuits 31

Introduction:

Data Acquisition and Control Applications 33
An A/D and D/A Convertor 33

Chapter One:

The 6522VIA I/O Board 35
Programming the Ports of the 6522VIA Board 39
Programming a Visual Display Indicator 39
Using the LED Visual Display 40

Chapter One:
Bar Graph 2 Demonstration 40
Programming the 6522 Internal Timer 41
Timer Operating Modes 42
Line Comments: Square-Wave Generator Using the 6522 42
Another Project With the 6522 Timer 42
Using the Timer as a Counter 44
Programming the Internal Shift Register 45
A Variable Duty-cycle Square-wave Generator 46
Constructing the 6522 I/O Board 51

Chapter Two:
Sound and Noise Generation Using the AY-3-9812 53
How the Internal Registers Work 55
Programming the GI Soundchip 57
Programming an Example Gunshot 60
Program PIANO 62
Sound Demo for the AY-3-8912 64
Program Description 65
Assembling a Sound Generator Board 65

Chapter Three:
An 8-Bit D/A and A/D Convertor 67
Resolution 75
Accuracy 75
Quantizing Error 76
Conversion Rate / Clock Rate 76
Using Two D/A Convertors 81
A/D Conversion with the ADC1210 86

Chapter Four:
An Eprom Burner for the Apple Computer 95
Using the Eprom Burner 98
Using the Software 98
Testing an Eprom 100
Programming the Eprom 100
Assembling the Eprom Burner Board 101

Chapter Five:
Assembling an Eprom/RAM Board 113
Bank-Select Circuitry and Programming 119
How to Assemble the Board 120

Chapter Six:
The Apple Slot Repeater 121
How to Assemble the Board 124

Chapter Seven:
The Coupling of Two 6502 Systems 131
Program Description 131

Chapter Eight:
Connecting Other Microprocessors 135
The 8212 8-Bit I/O Port 135
The 8253 Programmable Interval Timer 136
Mode 0: Interrupt on Terminal Count 139
Mode 1: Programmable One-Shot 139
Mode 2: Rate generator 139
Mode 3: Square-wave Rate Generator 139
Mode 4: Software Triggered Strobe 139
Mode 5: Hardware Triggered Strobe 140
Additional Information on the 8253 140
The 8255 Programmable Interface 141
The Interface Adapter PIA 6821 148
The Apple as a Logic Tester 153

Chapter Nine:
The Control of Step Motors 161

Appendix 182
Glossary 184

General Information

For those of you who have not previously done many hardware modifications or detailed analyses of schematic diagrams, this general information section gives easy to understand tips on the tools you will need, logic diagrams, binary and decimal numbering systems, and wire-wrapping and soldering techniques.

The Tools You Will Need

Your basic APPLE II Computer, with some attachments and software, is a thousand-dollar item. So I'll not encourage you to use dime-store tools. Buy the best you can afford, keep them clean, and reserve them just for use on the APPLE. Don't double up tools with the family auto. You may not need them all, but here is my customizer's toolbox:

A medium-sized flat-blade screwdriver and **Phillips blade screwdriver** (a reversible combination is ideal). With these you open cases and remove cabinets.

A jeweler's set of flat and Phillips blade screwdrivers; **hex nut drivers** are optional. These drivers can be used to align tape heads, help make delicate wire bends, adjust trimmer controls and even repair watches.

One very thin screwdriver for lifting integrated circuits out of sockets. This will be its only purpose, but the first time you break the pins off a $10 jumper cable, you'll wish you'd used it!

Small scissor-type cutters (manicuring types are excellent). These will be used for snipping leads in tight spots.

Small diagonal wire cutters and/or front-cutting **'nippers'.** Your general purpose cutters. They are fast and easy to use, but not to be used for heavy wire around the house.

Needlenose pliers (two pairs, normal and 90-degree types). You'll need these for bending leads, also extracting bits and pieces you've dropped into a nest of wiring.

An X-acto type knife, with a strong blade and handle you feel comfortable with. Since this will be used to cut delicate solder traces, you should be able to handle it deftly. I use a single edged razor blade, but have leather fingers!

A scalpel, if you can get one. For very delicate trimming and scraping; a dental pick for pulling off solder balls or lifting parts off a board (get this item from an obliging dentist — they are often discarded when worn); tweezers and needle point hooks. The latter comes in handy for tracing incorrect wire-wrapping connections.

Rat-tail, triangular, and flat files. These are only for sprucing up the cosmetics, so if you don't care how it looks, save a few bucks.

A wire-wrapping tool. The decision on this can be tough. If you can afford it, get one of the electrically operated slit-and-wrap types, stay away from 'just wrap' tools, since they depend on the sharpness and quality of the sockets; also they are useless for wrapping capacitors or resistors. I use a simple double-ended tool sold by Radio Shack for about $5. It wears out after a thousand or so connections, but it fits my hand well, and is not clumsy like some electric units.

A soldering iron. The decision is not easy. Should you spend top dollar and get an expensive one or buy a cheap unit that can be discarded when it wears out? I use a $5 soldering iron which can be junked when it gets beat, but my editor uses the best he can get (a $30 temperature-controlled one). I file a set of $1 tips to my satisfaction, and lubricate the threads with white heat sink grease. This way I have a few different tips at my disposal. You **never** file plated tips.

A Multimeter. The voltage regulators in your Apple are very good, so any problems will usually show up as gross errors. This offers you a way out of buying an expensive multimeter; for most of these projects, the $10 pocket variety will suffice. However, for lots of repair work a better meter is in order; I use a $40 type (not digital!) for my work.

An oscilloscope. For the projects, no. But for repairs, yes. Don't panic thinking of a thousand dollars for a digital scope, because an old color television scope will do perfectly well; they can be found in the bargain bins for $50 to $100. If it saves you a $100 repair bill, you've paid for it. Mine is an old RCA type WO-90Q, built for early color TV, and just fine for the bulk of your Apple work.

You will also need supplies in the tool box. Among these are:

Solder. Get the best you can afford. There's nothing so unpleasant as a great glob of the stuff between two traces on a board. Order the multicore rosin flux type, and stay away from most of the off-the-shelf stuff. Remember, multicore rosin type only, and the finer the gauge the better. **Never use acid flux solder,** as used by plumbers and electricians.

Soldering wick. Marketed under the names *Spirig, Solder Up* and *Solder Wick,* it's a copper braid impregnated with soldering flux. When heated with the soldering iron it absorbs Solder off the board, thus freeing components. Don't do without this stuff unless you like fried circuit boards and burnt fingers.

Wirewrap wire. Also called by the tradename *Kynar,* this is 28-or 30-gauge single-strand wire used to interconnect the pins of wire wrap sockets. It comes in an assortment of colors; get them all, so you can keep data, address, power and ground lines separate.

Multiconductor cable. The more flexible wire is easier on the coordination, but also the most expensive. Best buy is *Spectra Twist,* and its kin, from surplus houses. If you need jumper cables, buy them; Making a two-ended, 40-pin jumper cable can be three hours of maddening work.

Bus wire. This is solid, uninsulated stuff. A small roll will do for a lifetime. I use it for wiring, securing bulky capacitors to circuit boards, holding bundles of things together and for making special tools.

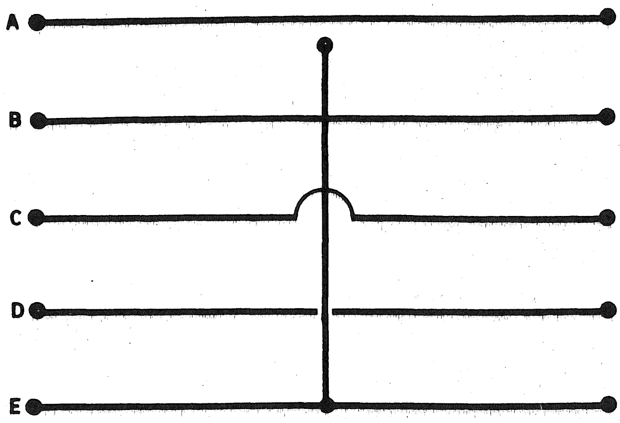
Miscellaneous. Sockets, perforated board, mounting hardware, and such will always be needed.

Details about supplies needed for each project in this book will be presented with the project. Except for integrated circuits, most of the items are available right off the shelf at a local Radio Shack or other electronics supply house.

Schematics

Schematic drawings of electronic circuits are identical to maps. They show routes, direction, junctions, relative importance and functions of locales, two-way and one-way streets, traffic flow and congestion and so forth. At first, the symbols may seem like the mysterious hieroglyphics of a secret society, but their symbolism can soon become as familiar as a roadmap. Even strange places can be assessed from afar.

First, the symbols. A line is a wire running from some point in the circuit to another. Consider the sketches below:

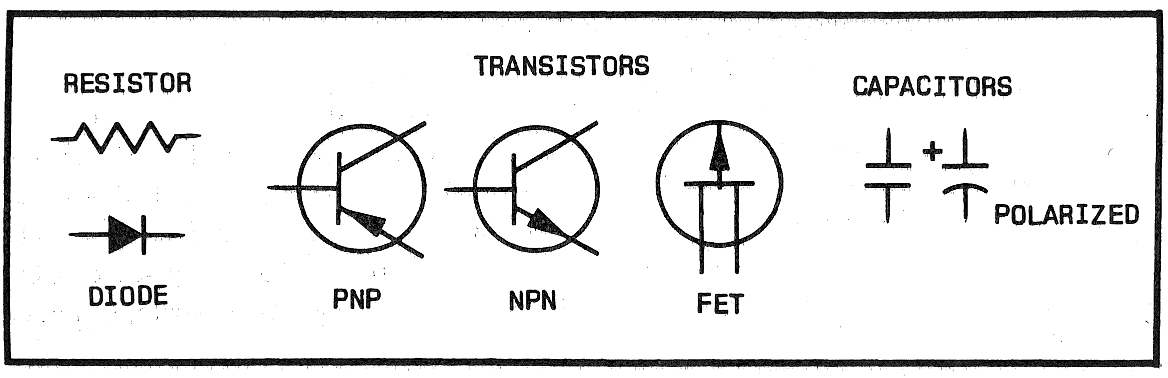


The first drawing is a simple wire. The electrical path moves from one point to another, in either direction. By following the path of a wire point to another, in either direction. By following the path of a wire through a circuit, the pattern of connections can be discovered. When wires are forced to cross one another, but not to connect with each other, it must be made clear. On a roadmap, non-intersecting roads are shown either by a break in one of the intersecting lines, or in showing interstate highways, merely by crossing one 'below' the other in a different color.

Sketches b, c and d are the three ways of drawing wires which do not connect to each other. The first, simple crossing them, is the most common. The second method places a semicircular bump in the crossing path, and it used by Sams Publications in this country and commonly in Europe. Occasionally the broken path crossing shown in sketch d is used.

When wires connect, a dot is used to clarify that a connection is to be made. Occasionally, you may come across earlier schematics which use the 'bump' method of showing unconnected wires. On these schematics, the lack of a bump indicates wires are connected.

The wires (or patterns of copper etched on circuit boards) connect electronic components. Some of them are:

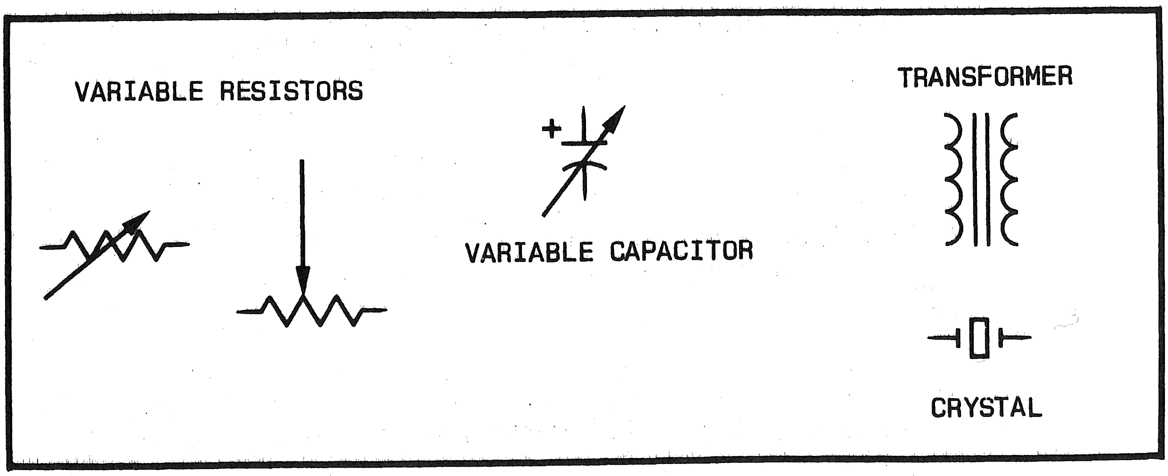


Since this is a lesson in reading schematics and not electronic theory, I recommend that you turn to an excellent book by Forrest Mims, 'Engineer's Notebook', sold by Radio Shack, for an introduction to what each of these parts does. Briefly, the symbol for a resistor has the flavor of a long wire being compressed, meaning the electrical flow is somehow being resisted. The innards of a capacitor generally consist of metal foil separated by a non-conducting paper or plastic, and the capacitor's schematic symbol is fairly representative, with two plates facing each other but not joining.

Some capacitors are designed to fit into a circuit in only one direction; these capacitors are identified on their bodies by a positive or negative sign. Another one direction (polarized) device is the diode. It consists of an arrowhead striking a barrier, implying that current may flow in the direction of the arrowhead, but not back across the plate. The body of a diode may have the diode symbol imprinted on it, or a band to indicate the 'barrier' end.

The transistor usually has three connections (such connections are called 'leads' on small parts such as these). These leads are identified as collector, base and emitter or source, gate and drain, depending on the transistor type. This will be shown on the diagram, and the transistor will be imprinted with the information, or it will be provided on the package in which the transistor is sold.

A few other symbols are:

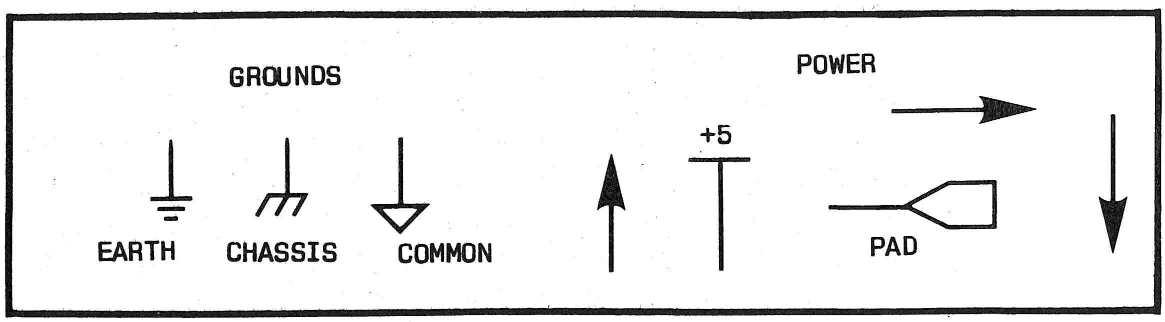


The first is a transformer, whose job it is to take current fed into one coil and induce that current, into a second coil. An iron or ferrite center (the parallel lines in the symbol) aids in efficent transfer of that current.

The next three symbols look like resistors and capacitors, which they are. The added arrows show that their values may be varied; hence, they are called variable resistors and variable capacitors. The variable resistor is best known as the volume control on a television, and the variable capacitor is found as the tuning control on a table radio.

The last symbol is a crystal, a piece of cut quartz capable of vibrating (resonating) under certain electrical conditions. Because a crystal is a very accurate, fixed, molecular device, it's capable of resonating (also called oscillating) at precise intervals. It is used for the master control of all pulses in the APPLE.

A few directional symbols are now in order:



The first are known as grounds, and they are used to indicate a potential of zero or neutral voltage. The first of the trio is an earth ground, commonly used in radio, television and hi-fi schematics, but purist use it only describe an actual connection to a ground spike or cold water pipe. The second is a chassis ground, indicating an electrical connection to the metal case which encloses the circuit. It is often (though incorrectly) interchanged with the earth ground.

The last of the three grounds is a 'common' or neutral ground, and the one which is used to indicate the zero voltage line in the computer. All other voltages within the computer system are described in terms of their relation to this ground.

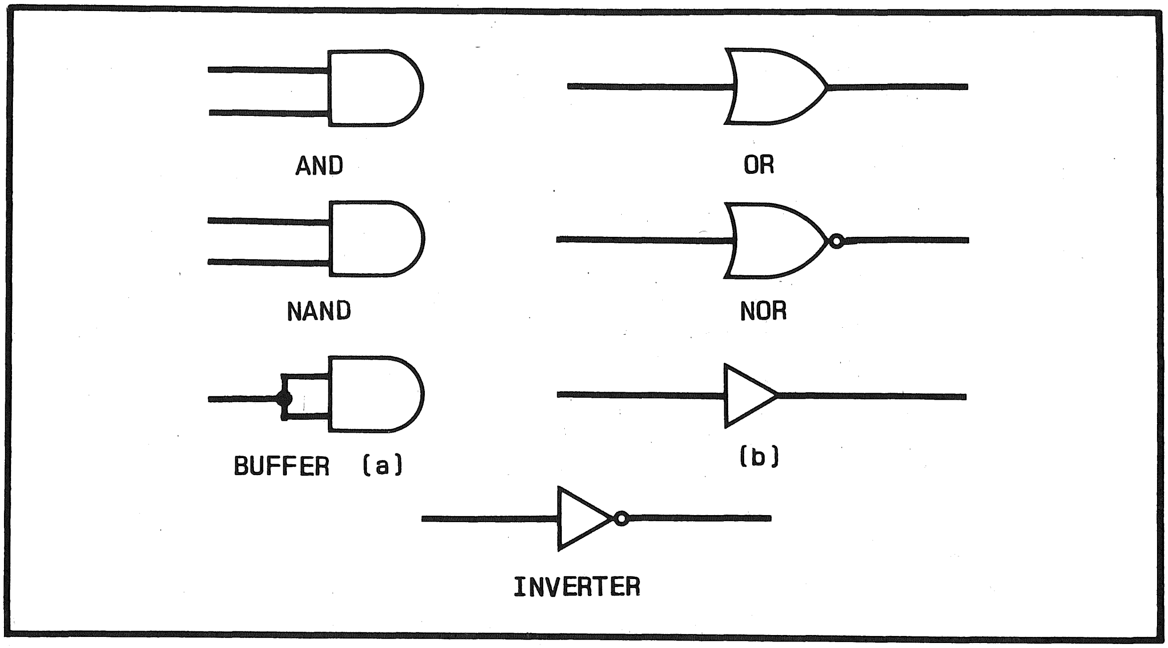
The next quartet of symbols indicate power. The up arrow generally points to an actual voltage value (such as +5 or +12). The horizontal line indicates merely a 'high' is made to the normal positive power supply for the circuits in the system (+5 volts in the TRS-80).

Non-positive voltages have no standard symbols. Negative (or below ground) voltages can have either a horizontal arrow or a down arrow, pointing to the voltage desired at that point. The schematic tells you that a connection is made to the voltage level shown.

Another use of a horizontal arrow is to point to important connections to be made elsewhere on the schematic or on other sheets of the schematic. In the former case, the arrow is used because actually drawing the wire may clutter the schematic, making it illegible. When you see an arrow, be sure to find the other end of the connection described (indicating words such as 'clock', 'mem' or 'port FF' may be used as guides to where the connection is made).

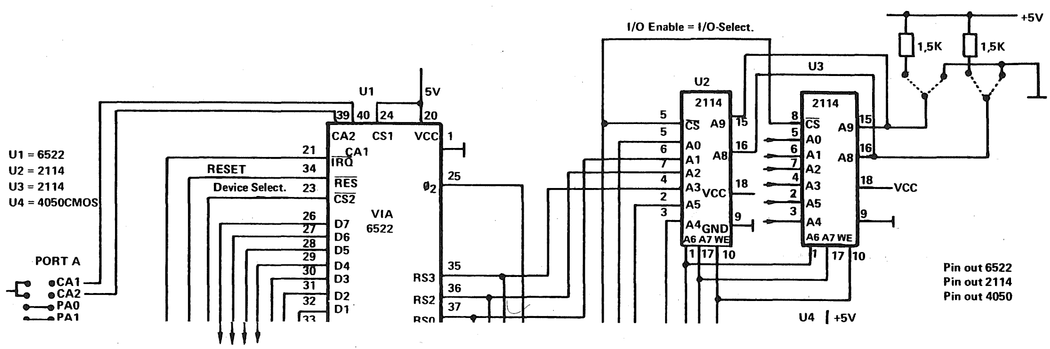
Another useful symbol is the last of the group above, the pad. It indicates a significant connection, usually to another device or circuit board. Using this symbol makes it clear that the connection is to be made somewhere off the board on which you are working.

The most common families of parts found in computer circuits are shown below:



These symbols represent integrated circuits, those multiple lead, buglike packages that handle the bulk of the work in the computer. Briefly, these are logical building blocks. Sometimes there are several blocks in one integrated circuit, and these various blocks may be scattered throughout the circuit diagram. This can be confusing when actually building a circuit, but since pin (lead) numbers are given, you only have to remember where you put the part.

Basically, that covers reading a schematic roadmap. Below is a section of circuit. See how the logic elements are connected to each other. An arrowhead indicates a wire leading off the board, and power and ground connections are shown. The numbers on the logic elements are the pin numbers for the component connections:



Be Tolerant

Every electronic component is manufactured to work within specific limits, whether they be accuracy, temperature, speed, power use or other limit. These are the components parameters or *tolerance*. The circuits in this book have been designed to use the most commonly available parts, so the matter of tolerances is rarely important. However, sometimes those tolerances are important, such as when talking about memory speed or power supply voltages.

Power supply should be within five percent of the voltage specified; a supply indicated at five volts may vary only from 4.5 volts to 5.5 volts. By using the power supply regulators shown in the schematics, these voltages should not be of concern. Unless you are familiar with power supply design, do not attempt to use other methods of regulation.

Very few of the resistors have tolerances noted on the schematics. The rule of thumb is one quarter watt at five percent, but if you can only obtain half watt units, or 10 or 20 percent resistors, don't be concerned. The quarter watt resistors are a bit less costly and are a bit more aesthetically appealing. Consider also that if a resistor is specified as 1,000 ohms, a 20 percent deviation gives a range of 800 ohms to 1,200 ohms. Thus, the standard values of 910 ohms or 1,200 ohms should do as well.

Capacitors are notoriously sloppy in their tolerances, especialy electrolytic types (those whose polarity is marked on the schematics). These normally vary from 20 percent low to more than 100 percent high — thus, when a 500 microfarad capacitor is noted, it can range from 400 to 1,000 microfarads. Also, there is some revision in the standard numbering method used for parts values: 470 microfarads is now being called 500 microfarads, for example. So when you try to obtain a capacitor value marked in the parts list, remember that a nearby higher value is fine.

Voltage parameters for polarized (electrolytic) capacitors are important. Never get an electrolytic capacitor with a value less than that specified, but do not hesitate to take one with a higher voltage parameter. That is, a capacitor specified at 47 microfarads, 16 volts, can be replaced with one specified at 50 microfarads, 35 volts. It may be physically larger, but it will work equally well.

If you walk into a store and hand the sales clerk a parts list, don't be surprised if you are asked a few more questions. You might be faced with chosing between parts which are identical as far as the parts list in this book is concerned, but which include other parameters.

Resistors can be carbon composition, carbon film, glass or wire-wound. These days, carbon film is common and cheap, and that's your first choice. Carbon composition is the next choice at a lower quality, and glass is excellent but at a higher cost. Forget wire wound, because they can contribute unwanted side effects.

Ordinary capacitors are manufactured in many ways: ceramic, polstyrene, polyester, silver mica, polycarbonate and paper. For the bypass capacitors necessary for all the circuits in this book, ceramic types are your choice. Cheap. If you get silver mica, so much the better, but you'll pay a price. Watch out for polystyrenes or polyesters if you plan to solder, because they are delicate and you can damage them with too much heat. Otherwise they are excellent, but quality overkill. Polycarbonates are slick types, and you might consider using these if you build the 8-track mass storage system. Run the other way if you see paper capacitors.

Electrolytic capacitors come in two basic types — metal cans (covered with plastic), and those manufactured using tantalum (an expensive metal of great strength and purity). For most digtal projects, choose the ordinary cans. Tantalums of the same value, although smaller, high quality, and very pert looking, are costly and not required here.

Digital integrated circuit part numbers are generic, which means that a 74LS00 circuit might be sold as an SN74LS00 or an NEC-74LS00. The prefix characters refer to manufacturers. On the other hand, those parts whose numbers contain 'LS' may not be substituted by parts marked 'S' or 'C' or by those with no markings. 74LS00 may not be replaced by 7400, 74S00,or 74C00, nor may they be exchanged for each other. When integrated circuits are specified, try not to substitute with other circuit 'families'.

This section will not make you a master schematic reader; only practice will do that. Pick up copies of the Engineer's Notebook mentioned above, as well as various of the project books sold by Radio Shack and others.

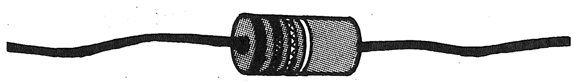
Those Colors: What They Mean and How to Read Them

The color codes used for resistors, capacitors and other parts are brought to you by the same folks that brought your phrases like 10W-40 and RS-232C: the standards-setting powers of the engineering industry. It becomes an international shorthand.

The colors are black, brown, red, orange, yellow, green, blue, purple, grey and white. If you can't immediately remember that, then pick up a piece of multi-conductor "rainbow" cable. The colors are all there in the same order. The table below presents the color codes and how they can be read on the bodies of resistors, capacitors and diodes.

FIRST AND SECOND COLOR BANDS		THIRD COLOR BAND	
BLACK	0	BLACK	0
BROWN	1	BROWN	X 10
RED	2	RED	X 100
ORANGE	3	ORANGE	X 1000
YELLOW	4	YELLOW	X 10,000
GREEN	5	GREEN	X 100,000
BLUE	6	BLUE	X 1,000,000
VIOLET	7	SILVER	100
GRAY	8	GOLD	10
WHITE	9		

FOURTH COLOR BAND IS THE TOLERANCE
GOLD = 5% SILVER = 10% NONE = 20%



What do these values mean? Resistance is a kind of objection to electron flow, measured in ohms (pronounced with a long O). The abbreviation is a Greek omega (Ω). Thousands of ohms are kilo-ohms, or just kilohms and abbreviated K (k in Europe). Millions of ohms are megohms, abbreviated simply M. The ability of a resistor to withstand electrical current is measured in Watts (W). Most computer work is done with 1/4 Watt resistors.

For resistors without color bands, the values are stamped on using R (instead of omega) for ohms, K and M.

Capacitance is the inclination of a non-conducting object to store an electrical charge, measured in Farads. The abbreviation is a capital F. Since this is a very large amount of capacitance, real work is generally done in millionths of Farads, or microfarads (mF), and millionths of millionths of Farads, called picofarads (pF). Since many of the more popular capacitance ranges for computer work fall between these two figures, the abbreviation for thousandths of millionths of Farads, or nanofarads (nF) is common in Europe. The ability of a capacitor to withstand voltage is measured in voltage tolerance (V).

Capacitance is usually printed on the capacitor in mF; color bands are rare. Picofarads are marked "p"; the absence of an abbreviation indicated microfarads. Note that these capacitor "base values" are equivalent: 18=20, 27=30, 39=40, 47=50.

Copacetic Comprehension

There will doubtless be a day when books like this will be unnecessary. Personal computers will probably develop into the appliance area, with programmers, hobbyists, hardware designers and language specialists present only in the distant background of the market. But until then, we are all faced with being either frustated users or solderer-programmers, tailoring machines according to our personal demands.

To do this, certain skills are inevitably required. Among these are an understanding of non-decimal number systems, digital logic devices, machine-level languages, and a smattering of diagnostic sense. There are some fine books that cover all these topics, so this chapter will only deal with them as far as needed to put this book to work. Among them are:

- Binary, decimal and hexadecimal number systems, how they arose, how and why they can be used, and where understanding them is essential.

- Common digital logic devices that appear in the Apple and these projects, and how and where to use them.

- Some of the basic elements of machine language, and a few personal considerations on where it is best applied, and when BASIC is a better choice.

Number Systems

Numbering is the single most overrated problem in computer programming. The answer (posed before the question) is this: numbers are merely *counting names*. That is, it makes no difference whether we think in tenths of a mile or eighths of an inch. Nor does it bother us that a day is made up of 24 hours, while an hour is 60 minutes. That a year is 365 days frightens us not, nor that months are a motley collection sizes.

In parking lots, does it bother us that our vehicle may be parked in Row N as opposed to Row 14? There is no mystery when we mark off points with four scratches and a crosshatch. And does a dozen always conjure up ‘twelve’, or is a dozen something we have understood since youth?

Names are sizes are numbers; so it is with the number systems that we arbitrarily assign for the convenience of working with computers. When we are talking about electrical signals, it is clearest and easiest to think about ons and offs. Ons look pretty much like ones, and offs look like zeros. It’s a nice, clean concept and one that illuminates the way we can refer to the machinery.

There's more convenience to naming a computer data condition 10110100 than to calling it an on off on on off on off off. Were data the only consideration, the binary one and zero method might have been satisfactory, without resorting to other means of stroking our memories.

Finding a location in a computer's memory is a much more difficult task. Although a memory location called . . .

1110100010011010l0

. . . might be easier to think about than . . .

onononoffonoffoffoffonoffoffononoffonoffon

. . . it could use another step forward. In music, a long string of sixteenth notes like this —

Illustration of Illegible Musical Notation

— is broken up to make it legible, so it looks instead like this —

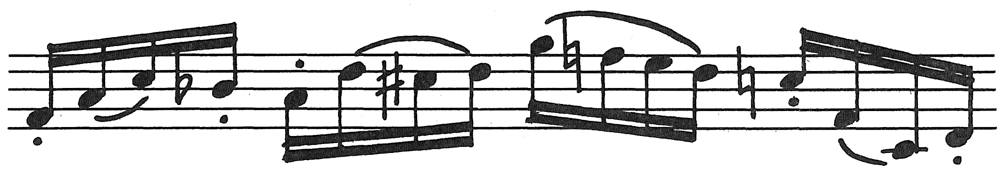


Illustration of Legible Musical Notation

Likewise, that long binary string can be broken up from 1101000100110101 into convenient groups . . .

1101 0001 0011 0101

. . . although the legibility is improved, the human spark, the ability to look and recognize (that *aha!*) is not there. So the next step is to set about naming the sections. Since these on-off conditions can be written down as binary numbers, why not write them down in their decimal equivalents?

The question is rhetorical, of course, because not only can it be done, it is done. The only question is how to do it. Were a computer capable of swallowing all sixteen of those binary digits (bits) in one gulp, that question might be easily answered by calculating the conversion of 1101 0001 0011 0101 using a binary-to-decimal conversion table. The result, we find, is 53557.

But the computer, alas, cannot swallow all those bits in one bite . . . it can only swallow one bite full of bits (pardon). In other words, though a computer may need numbers sixteen bits long, only eight data lines exist to carry that data. The component parts of the number 1101000100110101 are needed, eight bits at a time: 11010001 00110101.

There's the mathematical rub. 11010001 is 209 decimal, and 00110101is 54 decimal. This seems hardly related to 53,557. Another solution is necessary, and it is a naming system as much as a numbering system. It names each of the sixteen possible combinations of four binary digits:

0000	is named	0	and is equal to decimal	0
0001	is named	1	and is equal to decimal	1
0010	is named	2	and is equal to decimal	2
0011	is named	3	and is equal to decimal	3
0100	is named	4	and is equal to decimal	4
0101	is named	5	and is equal to decimal	5
0110	is named	6	and is equal to decimal	6
0111	is named	7	and is equal to decimal	7
1000	is named	8	and is equal to decimal	8
1001	is named	9	and is equal to decimal	9
1010	is named	A	and is equal to decimal	10
1011	is named	B	and is equal to decimal	11
1100	is named	C	and is equal to decimal	12
1101	is named	D	and is equal to decimal	13
1110	is named	E	and is equal to decimal	14
1111	is named	F	and is equal to decimal	15

This may seem overdone; but A, B, C, D, E, and F are darn good names for binary values which exceed the number nine. If you don't have a name, make one up. For practical purposes, keep it within the symbols everyone has on the typewriter.

Back to the number 1101000100110101. Crack it into those four legible pieces (1101 0001 0011 0101), and it can be named **D135**. To convert it to decimal, remember the old rule: the 5 is in the ones place, the 3 is this time in the sixteens place, the 1 is in the two-hundred-fifty-sixes place, and the D is in the four-thousand-ninety-sixes place. Thus, **D135** is 5 plus 3 x 16 plus 1 x 256 plus (see the chart) 13 x 4,096, or . . . 53,557!

So, now that long binary number can actually be digested by the computer as a byte of **D1** and a byte of **35**. After a while, the number system comes easily. My personal recommendation: work in it. Convert to decimal only when you absolutely must. Think in hexadecimal and binary. They are the tools with which you can speak to the computer.

Throughout this book, numbers in hexadecimal are printed in **BOLD**.

Converting Binary to Decimal

In the grade school years, students used to learn that a number like 5,163 contained a 3 in the ones place, a 6 in the tens place, a 1 in the hundreds place, and a 5 in the thousands place. It was to remind them that 5,163 was really 3 plus 60 (6 x 10) plus 100 (1 x 10 x 10) plus 5,000 (5 x 10 x 10 x 10).

The way other number systems are written follows this same pattern for their own bases. In base eight the number 5,163 would have a 3 in the ones place, a 6 in the eights place, a 1 in the sixty-fours place, and a 5 in the five-hundred-twelves place. That means that 5,163 is really 3 plus 48 (6 x 8) plus 64 (1 x 8 x 8) plus 2,560 (5 x 8 x 8 x 8). But notice how that's decimal thinking! Really in base eight there could be no '8' . . . it would have to be called '10'! 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, and so on. So 5,163 in base eight is still 3 plus 60 plus 100 plus 5,000!

The binary system sneaks in the same way. A number like 1101 0001 0001 0011 turns into a 1 in the ones place, a 1 in the twos place, a 0 in the fours place, a 0 in the eights place, all the way up to a 1 in the thirty-two-thousand-seven-hundred-sixty-sevens place. In binary, the one on the far left is still a 1 in the quadrillions place, don't forget. But the message is how to convert all this to decimal. And here it is:

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	1

Just add the numbers: 1x1 + 1x2 + 0x4 + 0x8 + 1x16 . . . +1 times 32,768 + 41,619. Voila. No matter how long the number is, and in whatever base:

1. Start at the left and produce a chart of the base number's powers, starting with 0 (X to the 0 power is always 1).

2. Lay the number to be converted underneath the base number chart.

3. Multiply each base number power by the digit in its place.

4. Sum the resulting numbers.

Does it work? Certainly. What is 163,341 in base 9? And in base 10?

```
Base 9 powers:          5       4       3       2       1      0
9 to that power:      59049    6561    729     81      9      1
Number to convert:    1        6       3       3       4      1
Multiplication:     1x59049 6x6561 3x729    3x81    4x9    1x1
Subtotals:            59049  +39366 +2187   +243    +36    +1
Converted result:                  100882, base 10

Base 7 powers:          5       4       3       2       1      0
7 to that power:      16807    2401    343      49     7      1
Number to convert:      1       6       3       3       4      1
Multiplication:       1x16807 6x2401 3x343   3x49    4x7    1x1
Subtotals:            16807 +14406   +1029   +147   +28     +1
Converted result:                   32418, base 10

Base 10 powers:         5       4       3       2       1      0
10 to that power:     100000 10000    1000    100     10      1
Number to convert:      1       6       3       3       4      1
Multiplication:       1x100000 6x10000 3x1000 3x100 4x10 1x1
Subtotals:            100000+60000   +3000   +300    +40    +1
Converted result:                  163341, base 10
```

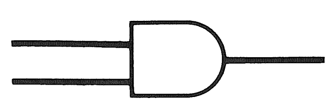
Digital Logic Devices

The binary number system and digital logic devices were developed together as a way of solving a practical dilemma: how to mass produce computers which could work quickly and accurately, and yet be inexpensive. The problems of creating consistently accurate circuits, working with many different voltages levels, are formidable. Thus, simple yes-no, on-off logic was developed.

The intimidating term *Boolean algebra* is being used for the first, and last, time in this book — right in this sentence. You'll probably hear the phrase from time to time, but no matter — it's a professional's buzzword to keep the masses out. Forget it.

Back to digital logic devices. The essence of digital logic is to evaluate binary, on-off input; sometimes to determine a pattern of similarity or difference, sometimes to sense a change and sometimes to search for a signal. An appropriate result is produced as a result of the logical operation.

One of the logic building blocks is called a gate. A gate electronically evaluates its input to determine the pattern of similarity and difference of signals, and produces a specific output. A simple gate is shown below:



Simple AND Gate

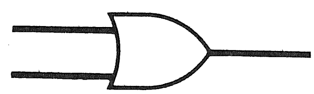
Its job is to determine if the first AND second inputs are both at the one (high) level. Only under that condition will its output produce a high (one) signal. The table below shows how this AND gate works.

AND Gate

If input #1 is -	If input #2 is -	The output result is -
0	0	0
1	0	0
0	1	0
1	1	1

AND Gate Action

The table is called a *truth table*, and its purpose is to present every possible input and output condition for a given gate. Below is an OR gate. Stated in words, if either the first OR the second input is high, the output will be high. Examine the OR gate truth table; it really is quite logical.



Simple OR Gate

OR Gate

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

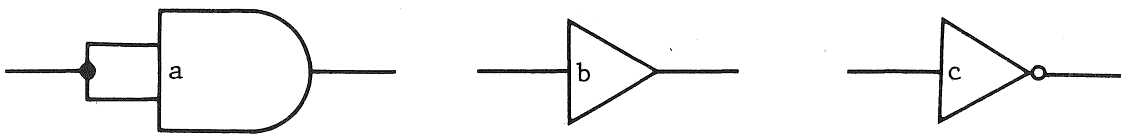
OR Gate Action

Given a huge set of interconnected gates and their known inputs, the final output of the group can be determined by using truth tables like these. Gates may have more than two inputs (some have sixteen), and may produce the opposite results from the two described above (NOT-AND and NOT-OR gates, known as NAND and NOR gates). Truth tables reveal how the integrated circuit's design engineer specified the pattern of binary logic inside the circuit.

In this way, given a desired output and a known number of input signals, it is possible to determine what set of input values will trigger the desired output.

There are a number of other types of digital circuits. Most are created out of gates like those described above, but their features are unique enough to think about them separately. Among these other digital logic circuits are buffers, flip-flops, counters, latches, multiplexers and shift registers.

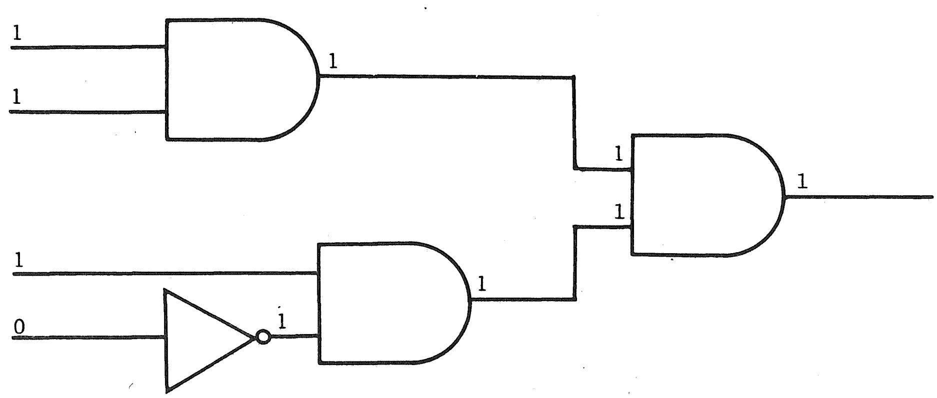
A buffer can be thought of as a two-input gate with both inputs tied together, like this:



Buffer as (a) Two-input Gate, (b) Buffer and (c) Inverting Buffer

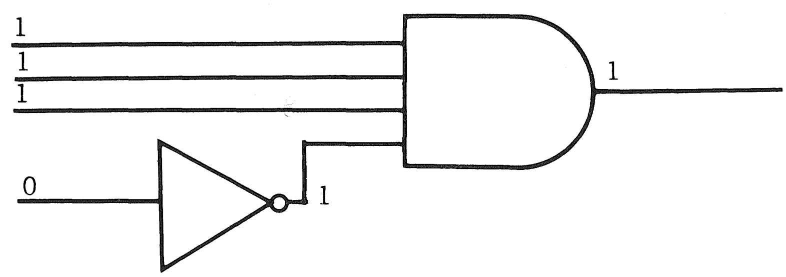
Its truth table is much simpler than that for two-input gates, because there are now only two input conditions. Either both inputs are high, or both inputs are low. Gates with 'true' outputs (AND, OR) will merely follow the input condition. When the inputs are high, the output is high; if the inputs go low, the output becomes low. Separate logic devices are manufactured that perform this 'follow-the-leader' function, and they are called *buffers*. They serve to isolate sections of a circuit, or rejuvenate a signal so it can feed many dozens of inputs in a large machine.

When a buffer reverses the condition of its input, (a high input is output low, and vice versa), the device is called an *inverter.* This kind of circuit can save the day in some cases, as when trying to locate a given binary number. Assume a circuit needs the binary number 1110 to turn on a pilot light. It is possible to choose four separate gates, each of which would provide an output matching the desired number. These would be connected through more gates, and eventually the number could be discovered when the final signal was triggered properly. One way of detecting 1110 is shown below:



Bad Decoding Scheme for 1110

But, although this circuit works, economy of cost and space and simple clarity dictate another solution. The last input could be inverted before it is evaluated, resulting in a pattern (1111) which could be quickly recognized by a multiple-input gate. The result is electronic simplicity and legibility; an improved decoding circuit is shown below. The ultimate result is the same.



Good Decoding Scheme for 1110

A flip-flop is a 'black box' which provides two outputs. When an input value is high (one), the first output will be high, and the second will be low. When the input value switches low (zero), the outputs will reverse. In other words, two opposite outputs for the price of one. But there is another significant use of the flip-flop.

Flip-flops also have an important input called a clock trigger, which is triggered only when its input returns to a given level. Only then will the outputs of the flip-flop reverse. That is, a given flip-flop clock may receive a 'zero' pulse. Its outputs will reverse. Then the zero pulse changes to a 'one' pulse. Nothing happens, but the trap is set to spring. When the one pulse changes back to a zero, the outputs reverse again. For every two changes at the clock, there will be but one change at the output. It takes four clock changes to produce two output changes.

Why is this useful? Because it is electronic, binary division. The truth table here shows how it works.

Binary Division with a Flip-Flop
Output of First Flip-Flop Connected to Clock of Second
Flip-Flops Change State Each Time Input Returns Low

Clock Input	Flip-Flop Output	Second Clock Input	Second Flip-Flop Output
0	0	0	0
1	0	0	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	1	1
1	1	1	1
(Input)	(Input/2)		(Input/4)

Binary Division with a Flip-flop

Digital logic devices known as *counters* are combinations of gates and flip-flops that allow certain patterns be counted: Binary, Binary Coded Decimal (BCD, where the highest number is decimal 10), Gray code and others.

Latches are very much like flip-flops, except that the input is 'captured' at the output by a trigger signal called an enable, a select, or a gating pulse. The input may change continuously, but the output only reflects the input when the enable is activated. Latches are very useful when hundreds of thousands of signals are flying around on one set of lines, and the computer must select only certain groups of signals. The cassette output of data is a latch; only the 500-baud (bits per second) pulses of data reach the cassette output, even though many different signals reach its input.

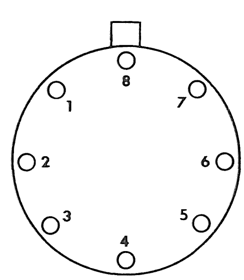
Multiplexers are sometimes misunderstood, but mostly because of their formidable name. A traffic light is a multiplexer — it allows several streams of traffic to meet at one intersection, but only one stream to proceed. The multiplexer is the electronic equivalent, having several inputs. Gating signals select which of the inputs may reach the output. In a computer, this allows several devices to share a circuit (like the video, which must be sent to the screen, but also sends and receives characters from the rest of the computer).

Finally, *shift registers* treat bits of data like a bucket brigade sends up water: it goes in one end, and at each electronic 'go!', the bucket is sent along one position. The dots which make up the video display are produced by circuits which shift them out to the screen one bit at a time, in synchronization with the monitor's sweeping electron beam.

Reading The Pins

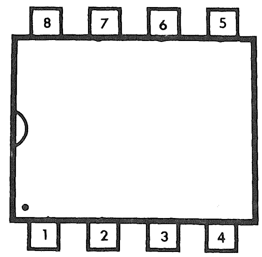
Finding your way through digital circuits is much easier than finding your way through an ordinary table radio. Industry standards have made the process simple. Consumer integrated circuits are packaged in small, rectangular, plastic or ceramic cases with anywhere from 8 to 40 external connections known as 'pins'.

Earlier integrated circuits — and many of the audio types currently being produced — were packaged in small metal cans and looked like transistors, with many wires protruding from the bottom. The wires were arranged around a keying tab on the edge of the can, and numbered like so:



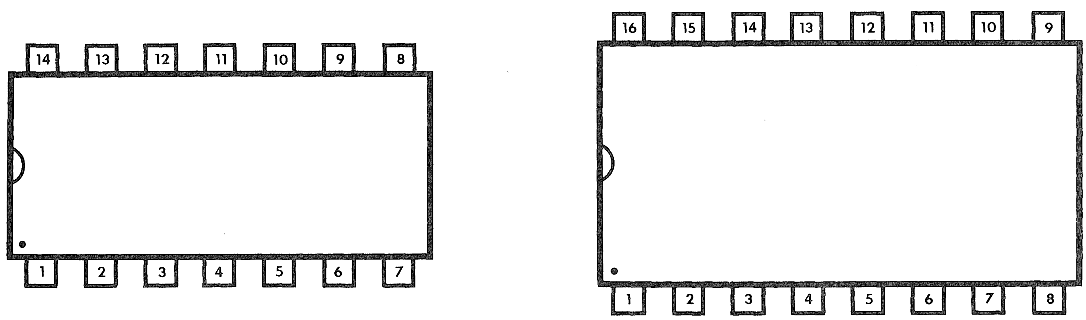
Can-type IC Pin Numbering

As such circuits developed into more sophisticated and powerful devices, more pins were needed for input and output. A rectangular package was developed, but it was still numbered in a circle, starting (when looking down from the top) from left of the notch, so:



Dip-type IC Pin Numbering (8 Pins)

All modern integrated circuits can be read from the top in this same way. 14- and 16-pin types start from the top left and read around:



14- and 16-Pin Dip IC Pin Numbering

You can read the pinouts of 18-, 20-, 24-, 28-, and 40-pin circuits in the same manner. The highest numbered pin sits just opposite the lowest numbered pin. In the beginning this practice may seem confusing; it is. But after using the circuits — and counting their pins over and again — you will probably feel comfortable with the pin arrangement.

Just one thing: when you assemble Apple add-ons, most of your work will be done from the bottom . . . which means reading backwards!

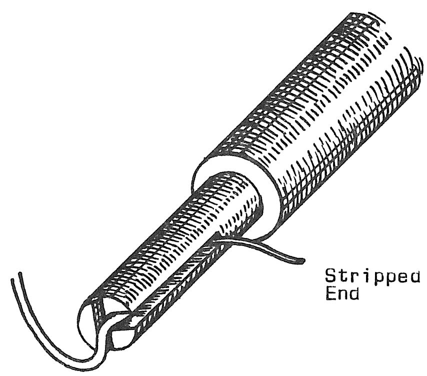
Wire-Wrapping Technique

It's not without a bit of hesitation that I attacked many of the hardware projects presented in this book. Some are simple, but many, particularly those using memory circuits, need many connections. The wiring is not complicated, just tedious.

If you work carefullly, all is likely to be well; but even a touch of haste will encourage confused connections. It is in these cases especially that wire-wrapping is the technique to use.

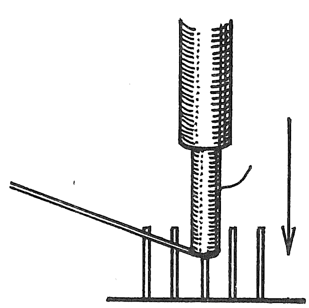
Wire-wrapping is not only easier than soldering, it is secure, simple, easier for correcting mistakes — and less costly. For wire-wrapping, you will need wire-wrap sockets, which are sold by most hobbyist supply houses including Radio Shack. Likewise, wire-wrap wire and a simple hand tool are used for the process. Here are the steps:

1. The wire, still connected to the spool, is inserted in the V-shaped stripping slot. Insert beteen one half and one inch of wire. Pull downward from the V, and the wire will slip out, leaving a piece of insulation in the stripper, where it can be shaken out.



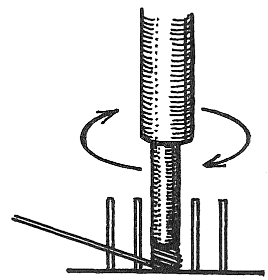
1. Insert stripped wire.

2. Look carefully at the end of the wire-wrap tool. There is a small hole, meant to fit over the pins of a wire-wrap socket. Next to it is a half-circle, into which you must slide the stripped wire. The stripped portion will slide up a groove in the side of the tool, stopping where the insulation begins.



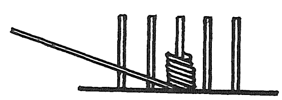
2. Slip over pin.
Hold wire firmly,
and slide fully down.

3. When the wire is in place, pull it sharply but gently upwards, and slide the tool on the wire-wrap socket. Holding the wire firmly, spin the tool in your hand. The wire will wind up on the socket pin, freeing itself from the tool. Remove the tool. The wire-wrapping is complete for that end of the connection.



3. Spin wire - wrap tool.
Wire rises along pin.

4. Cut the wire to a length that will comfortably reach its destination, and then some. Strip the end of it, and repeat the process above. The connection is complete. Don't forget to use different colors (white, yellow, red and blue are generally available). This will help you distinguish your connection patterns if changes become necessary.



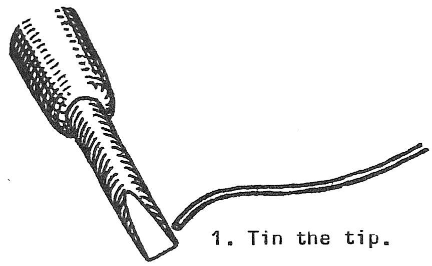
4. Finished connection
has no bare wire
protruding.

Soldering Technique

For projects from scratch, soldering should be considered the final process, the actions of a self-assured, confident hobbyist. For modifications, it is a necessity. In either case, and whether you are a micro-acrobat or distinctively clumsy like me, you can solder well. The requirements are patience and good solder.

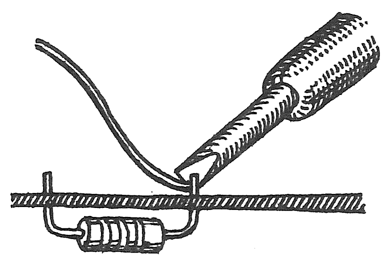
To start, make sure you are using an iron in the 25 to 40 watt range, never a soldering gun. The solder should be high quality, multicore solder. It is expensive, but will save many grief stricken hours tracing 'cold solder joints', or removing globs of dull solder from between and under integrated circuits.

1. Clean the soldering iron tip, and heat the iron. Flow fresh solder on the tip to 'tin' the tip, which will help the solder flow from the tip of the iron to the part to be soldered. If the iron has been used, clean any encrusted material from the tip, and use coarse emery paper to shine the solder. If the point gets deformed, bent, or very corroded, file it sharp with a fine file, and re-tin the tip.



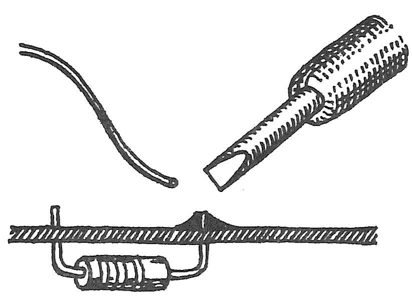
1. Tin the tip.

2. Keep an old sponge handy, slightly damp. Run the tip of the iron quickly over it as you solder to remove the excess flux. Always use a soldering iron holder (usually provided with an iron); if you don't, you'll wish you had the first time you burn a large hole in your imitation walnut, vinyl-topped desk.



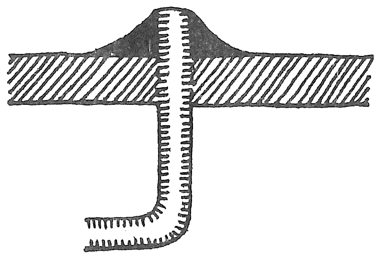
2. Bring solder, parts and iron into contact.

3. In the olden days, the rule was 'heat the parts, not the solder'. Forget it. Make sure the iron is no hotter than 40 watts (and remember never to use a soldering gun) and that the parts you are about to solder are very clean. Place the iron against the part, making as much contact with it as possible along the angled tip of the iron. Place the end of the solder at the junction of the iron and the part, and flow just enough solder to make a clean, shiny, flowing connection.



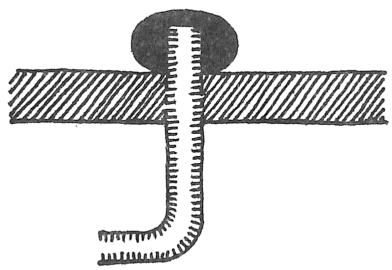
3. Lift iron and solder simultaneously.

4. Remove the iron immediately and let the part cool. If a wire is being soldered, hold it still until the solder becomes cloudy and cool, or else an incomplete connection may result.

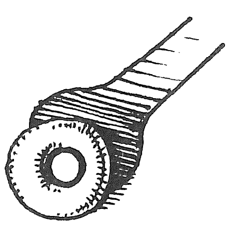


4. Finished
solder connection.

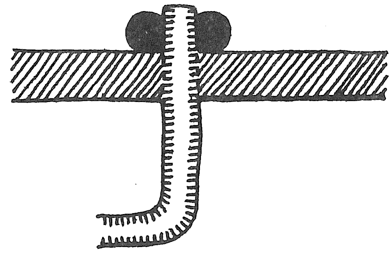
5. If solder bridges develop between connections that are very close together, don't try to suck up the solder with the iron; you can only overheat the parts that way, and end up with blobs of solder and flux. Instead, use solder wick or solder-up to remove the excess solder, and start again. Let the parts cool before soldering again (a half minute should be enough).



5. Bad solder connection –
no contact with board
(sideview).



(6A topview)



(6B sideview)

6. Bad solder connection –
no contact with pin

Tips on Handling Integrated Circuits

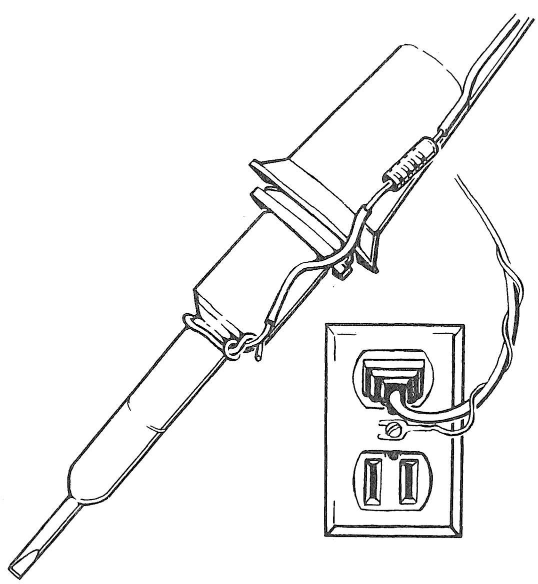
In the early days of microcomputers, there was a lot of user hesitation about installing memory chips because of warnings about static electricity damaging the memory devices. At that time the fear was reasonable; but today (with just a little caution) there need be no problem.

1. Never place any integrated circuit on highly charged plastic material, especially styrofoam.

2. Handle memory chips, CPU's (such as the 6502), LSI devices (large-scale integrated circuits, usually those with 28 or 40 pins), or any marked MOS, CMOS or NMOS (metal-oxide semiconductors), with care. Hold them by their ends, never by the connection pins.

3. Purchase a static-free workbench, which is a conductive cloth sheet with a wrist strap and safe grounding cable. These can be obtained from Wescorp for about $18.

4. Ground your soldering iron to an earth ground *but only through a series-connected one-megohm resistor– never directly!* The grounding is not absolutely essential, but helps if you live in a very dry, static-producing environment.



Grounding a Soldering Iron

5. Work with any integrated circuits with the power off. Make sure the integrated circuit's ground and power pins are all connected (soldered or in sockets) before turning on the juice! A difference of a mere half a volt between certain pins can **kill** an IC.

6. Use high-quality sockets for integrated circuits wherever you can. This will not only keep excessive heat away from them, but will also save the day if one is damaged. Unsoldering a 40-pin integrated circuit is not pleasant.

7. Above all, work slowly and carefully. By far the greatest villian is haste. Oh yes — do keep furry animals out of the area!

NOTES

Introduction

Why expand the Apple Computing System at all? What proud Apple owner has never wished that the computer would do just this one more thing, to somehow be able to perform the magic necessary to do that certain thing that would just exactly fit your particular application. While there are a lot of interfaces and expansion modules available on the market, none was really designed with the particular application you had in mind. The purpose of this book is to provide you with an expansion module that will be flexible enough that you will be able to adapt it to any specific application you have in mind. Most people, when faced with the arduous task of trying to make their Apple do one particular thing that would make it perfect for their system, are really dismayed by how much special knowledge they would need and how really complex it appears. A lot of people will simply decide, "Oh well, I can probably get by without it." The authors, in writing this book, are providing a much better alternative to simply doing without that special little goodie you would like. They are going to lead you step by step through a series of projects and applications that will allow you to custom design exactly the piece of hardware you need for that special application you have been wanting to do ever since you got your computer.

Data Acquisition and Control Applications.

The Apple was originally called 'The Appliance Machine'; however, it was designed, at least to some degree, to also be used for data acquisition and control applications. The way the Apple is usually configured, you will find that there are four empty slots inside, and it would really be nice to utilize them in order to expand the capabilities of your Apple II Computer.

An A/D and D/A Convertor

The analog to digital and digital to analog convertor will be one of the most important projects you can put together and one of the most useful applications presented in this book. The reason for this is that the real world is analog, not digital, but the computer deals exclusively with digital information. Examples of analog would be temperature control and sensing, light control and sensors, and the measurement of voltage levels. Virtually any type of sensor could be hooked to an analog/digital convertor, allowing the computer to 'see' what's going on.

There are things that would be really handy around the house: perhaps a hobby environment such as model railroad control, a burglar alarm system that could be monitored by your computer, and all of the peripheral devices that are already available for the computer user at home. This book will prove invaluable to people who have just bought a strange new device or a new printer, and wonder, "How can I hook that to my Apple?" This book will give you the opportunity to control even the most complex industrial or home applications at a very low price. Gastromatic is a relatively new application where the home computer can be used in lowering the energy costs of running your furnace. The ability to do this, before the advent of the small home computer, would have cost many thousands of dollars and been prohibitive for most people. With the interfaces and applications described in this book you will find you have the ability to control machines in a way that only a few years ago would have been absolutely impossible. Examples of this would be driving step motors, automatic monitoring and remote control of drive motors and fans, or the control of any machine that was previously controlled by mechanical means. The basic concept of this book is to vastly improve the Apple II Computer's ability to communicate with and control the real world.

1

The 6522VIA I/O Board.

The Apple II Computer, as configured at the factory, has practically no way to interface with the real world, with the possible exception of playing a game with the joysticks. Games are very impressive and fun, but after awhile you will begin to wonder, "Now how do I get this nifty little machine to do something practical and prove I didn't just waste my money on a game-playing machine?" One of the biggest problems with trying to use the game playing input ports for transfer of data is that they are limited to four bits or one nybble, which really limits the amount of data that can be transferred in a given period of time. Because of the severe I/O limitations of the Apple computer, the authors intend to show you how to use the 6522 versatile application interface I/O board to move large amounts of data in relatively short periods of time. Consequently, you will have the ability to do a great many of the things people said couldn't be done.

One of the first problems you will encounter, which is not known to many people, is that the 6522 I/O chip is not fast enough to pick up the clock pulse from the 6502 microprocessor chip. In order to make the 6522 compatible with the 6502 microprocesor, it is necessary to incorporate a time delay. We will use the small 4050 CMOS chip. This solution will work in 99% of the cases. For that 1% of the time when it doesn't, never fear, there will be further help outlined later in the book. The 6522 I/O board also has 1K of RAM built into it, of which 1/4 is usable at any time. These 256 bytes are suitable for applications such as a small machine-language monitor that you want to tuck safely out of BASIC's way.

Figure 1.2 demonstrates how you can use the 1K byte RAM on your 6522 I/O board. On each board there are two 2114 1K by 4-bit static RAMs for your machine-language programs. But out of this 1K RAM you can really only use 256 bytes at a time. The addresses for that 1/4K bytes of RAM depend on the slot in which the board is plugged. For instance, if you want to put a small machine-language program in the RAM on the board while it's plugged into slot 4,

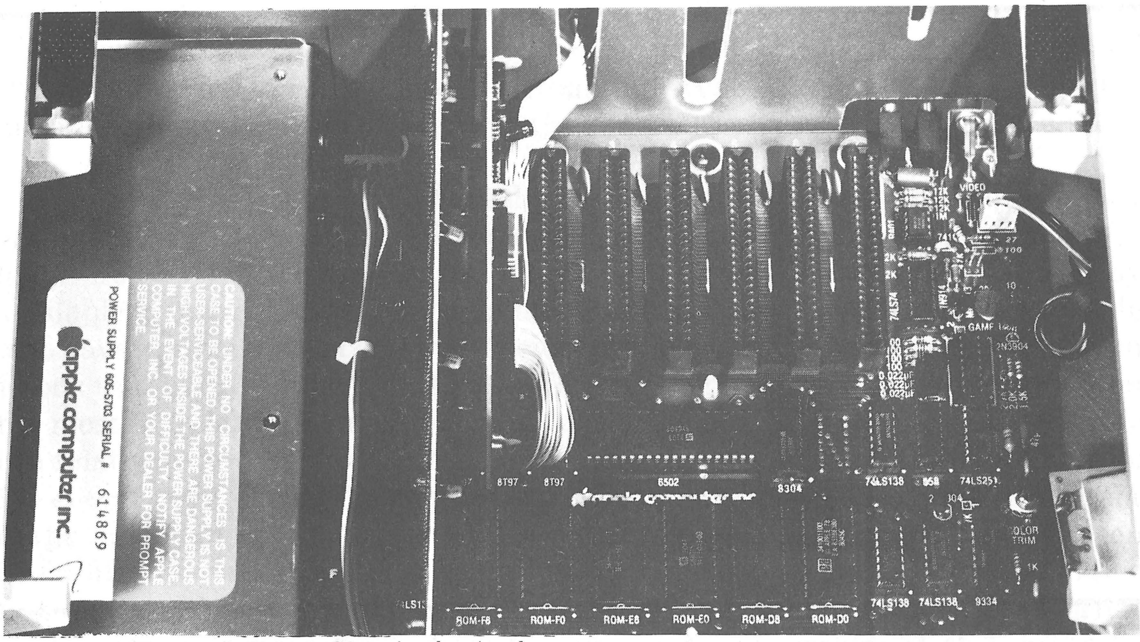


Figure 1.1 The Four Empty Slots in the Apple

you can write your program into the RAM area starting at **C400**. You need not be concerned about which 1/4 of the RAM your program is in, because you may select any 1/4 you wish by using the two switches on the I/O board. Note that every 1/4K block on each board is addressed using similar addresses (for example, **C500-C5FF** in slot 5).

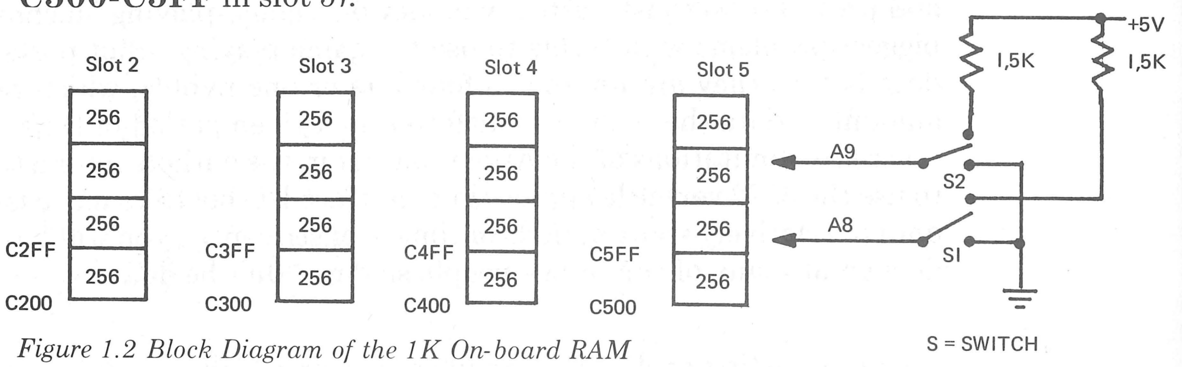


Figure 1.2 Block Diagram of the 1K On-board RAM

	SI	S2
1. 1/4 k	off	off
2. 1/4 k	on	off
3. 1/4 k	off	on
4. 1/4 k	on	on

Suppose you need four different machine-language programs for a particular application. You could write these four routines into address **C500-C5FF** (with the I/O card in slot 5) while setting the two switches to the four different positions. Then the four programs (each being 256 bytes or less) are in that 1K RAM block. By setting the switches, you can now address four different programs in the same area of memory.

Different 1/4's of the 1K RAM in 256 Byte chunks can easily be accessed by simply flipping the switches on the board itself.

The clear area on the left side of the board is a prototype area free for you to use for your own experimentation and custom applications. The 6522 I/O board can be programmed and controlled from virtually any language, whether it's store instructions from machine-language or POKE and PEEK commands used with the higher level languages. A section of this book is devoted to showing you how this is done, whether it's from machine-language, or a higher level language such as PASCAL or BASIC. The 6522 has two ports, A and B, and 8 bi-directional data lines. It also contains 2 timers, 1 eight-bit shift register, and 4 hand-shaking lines. The hand-shaking lines are used to communicate with the other devices that are capable of sensing a READY or NOT READY condition.

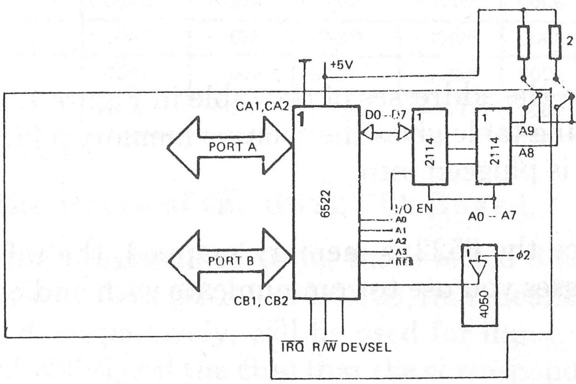


Figure 1.3 Block Diagram of the 6522 Board

Figure 1.4 Photo of the 6522 Board

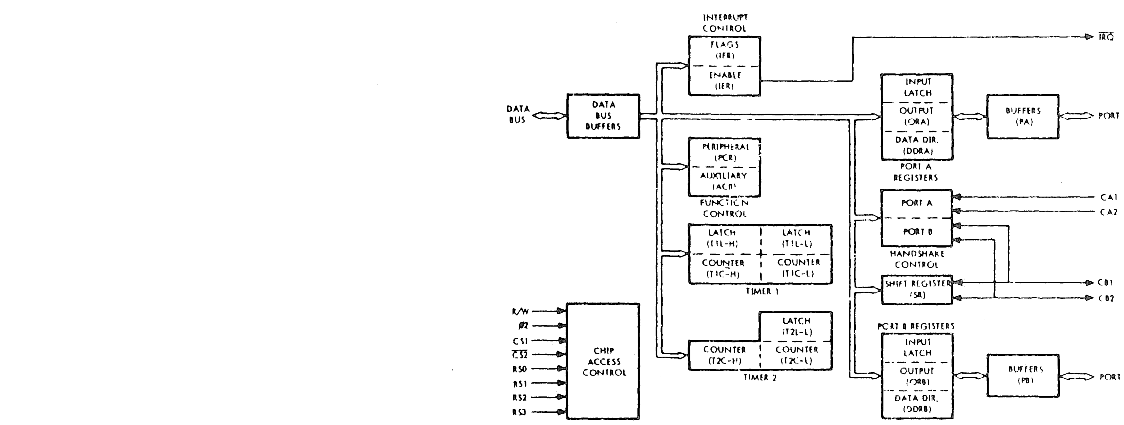
board in slot	6522 Hex from	to	Decimal from	to
2	C0A0	C0AF	−16224	−16209
3	C0B0	C0BF	−16208	−16193
4	C0C0	C0CF	−16192	−16177
5	C0D0	C0DF	−16176	−16161

board in slot	RAM Hex		Decimal	
2	C200	C2FF	−15862	−15617
3	C300	C3FF	−15616	−15361
4	C400	C4FF	−15360	−15105
5	C500	C5FF	−15104	−14849

Figure 1.5 Address Table

Also, the addresses of the table in Figure 1.5 are from 0 to 15, or **00** to **0F**. The following table gives the relative memory addresses, depending on which slot the board is plugged into.

Since the 6522 is memory mapped, the table above gives the actual memory addresses you use to communicate with and control the 6522 I/O board.



From '6502 Programming Manual' for Rockwell R6500 Microcomputer System.

Figure 1.6 Block Diagram of the 6522

Register Desig.	Description		SLOT 2		SLOT 3		SLOT 4		SLOT 6	
	Write	Read	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
ORB/IRB	Output Register "B"	Input Register "B"	C0A0	−16224	C0B0	−16208	C0C0	−16192	C0D0	−16176
ORA/IRA	Output Register "A"	Input Register "A"	C0A1	−16223	C0B1	−16207	C0C1	−16191	C0D1	−16175
DDRB	Data Direction Register "B"		C0A2	−16222	C0B2	−16206	C0C2	−16190	C0D2	−16174
DDRA	Data Direction Register "A"		C0A3	−16221	C0B3	−16205	C0C3	−16189	C0D3	−16173
TIC-L	TI Low-Order Latches	TI Low-Order Counter	C0A4	−16220	C0B4	−16204	C0C4	−16188	C0D4	−16172
TIC-H	TI High-Order Counter		C0A5	−16219	C0B5	−16203	C0C5	−16187	C0D5	−16171
TIL-L	TI Low-Order Latches		C0A6	−16218	C0B6	−16202	C0C6	−16186	C0D6	−16170
TIL-H	TI High-Order Latches		C0A7	−16217	C0B7	−16201	C0C7	−16185	C0D7	−16169
T2C-L	T2 Low-Order Latches	T2 Low-Order Counter	C0A8	−16216	C0B8	−16200	C0C8	−16184	C0D8	−16168
T2C-H	T2 High-Order Counter		C0A9	−16215	C0B9	−16199	C0C9	−16183	C0D9	−16167
SR	Shift Register		C0AA	−16214	C0BA	−16198	C0CA	−16182	C0DA	−16166
ACR	Auxiliary Control Register		C0AB	−16213	C0BB	−16197	C0CB	−16181	C0DB	−16165
PCR	Peripheral Control Register		C0AC	−16212	C0BC	−16196	C0CC	−16180	C0DC	−16164
IFR	Interrupt Flag Register		C0AD	−16211	C0BD	−16195	C0CD	−16179	C0DD	−16163
IER	Interrupt Enable Register		C0AE	−16210	C0BE	−16194	C0CE	−16178	C0DE	−16162
ORA/IRA	Same as RegA Except No "Handshake"		C0AF	−16209	C0BF	−16193	C0CF	−16177	C0DF	−16161

Figure 1.7 Register Addresses of the 6522 Board

Programming the Ports of the 6522VIA Board.

Ports A and B are programmed using the internal data registers DDRA and DDRB. If the bit is set to 1 in DDRA or DDRB, that means the corresponding line in Port A or Port B, respectively, will be used for input. If the bit in DDRA or DDRB is set to 0, it will signal the chip that the corresponding line in Port A or B, respectively, will be used for output. As an example, loading DDRA with 255 or **FF** will signal the chip that all lines of Port A are used for output. Loading either of the data registers can be accomplished (in machine-language) by loading the Accumulator with the number desired, then storing it in that memory location. It can also be done from BASIC by POKEing the corresponding memory address with the number desired. Once the bits and the data registers are set, they will remain in the same configuration until the computer is forced through its power-up sequence. This can be accomplished by resetting the machine, by shutting it off and turning it back on, or by loading a new number into the data register. Upon reset or power-up of the computer all Port lines will set to 0. That will indicate all lines are to be used for input. They will remain in that state until altered by software running within the computer.

Programming a Visual Display Indicator.

To get you right into using the 6522VIA board, the first application will be a visual display indicator. We will show you how to light any configuration of 8 LED's, depending on the conditions existing within the 6522VIA chip. In order to do this you will need 8 LED's plus 8 current-limiting resistors.

Connect the anode of each LED to a corresponding bi-directional data line on the 6522. Connect the cathode of each LED through a 220 ohm limiting resistor to ground.

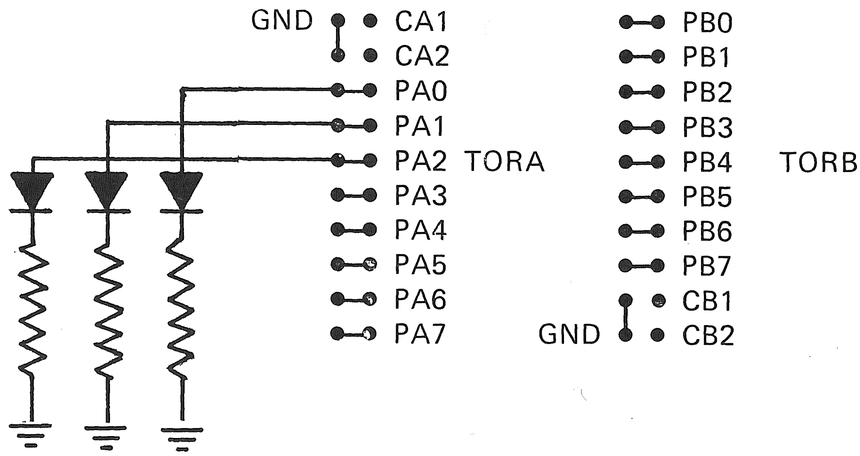


Figure 1.8 How to Connect LEDs to the Port

Figure 1.9 Bar Graph 1

```
10  REM  BARGRAPH 1
20  REM  BOAARD IN SLOT 4
30 DDRA =  - 16189:TA =  - 16191
40  POKE DDRA,255
50 A = 1
60  POKE TA,A
70  GOSUB 200
80 A = A * 2
90  IF A = 256 THEN A = 1
100  GOTO 60
200  REM  TIME DELAY
210  FOR I = 1 TO 50
220  NEXT I: RETURN
```

Using the LED Visual Display.

This demonstration program assumes that the 6522 I/O board is in slot 4. In line 30 we assign a variable to the internal register DDRA. TA is also initialized to the memory location memory-mapped to Port A at this time. In line 40 the POKE statement sets all of the Port A lines to outputs. Line 50 assigns the value of 1 to the variable A to be used in line 60 to output the number 1 to Port A. Line 70 calls a time delay routine at line 200. This is necessary so that we can see the LED'S change. Through each loop of the program, the variable A will be shifted left one place in order to turn off the light that was on and to light the next one in sequence. The way this is set up only one light will be on at a time. Line 90 is used to re-initialize the variables to start the lights through their pattern again.

Bar Graph 2 Demonstration

This demonstration program will show you how to make a true bar graph display. This means that the highest light lit will cause all lights lower than it in the sequence to be on at the same time. Line comments of the Bar Graph 2 demonstration program follow:

Figure 1.10 Bar Graph 2

```
 LIST

10   REM   BARGRAPH 2
20   REM   BOARD IN SLOT 4
30 DDRA =   - 16189:TA =   - 16191
40   POKE DDRA,255
50 B = 1:A = 1
60   POKE TA,B
70   GOSUB 200
80 A = A * 2:B = B + A
90   IF A = 256 THEN 50
100   GOTO 60
200   REM   TIME DELAY
210   FOR I = 1 TO 50
220   NEXT I: RETURN
```

Up through line 40 the programs are identical. In line 50, A will be set to 1 as in the previous program, and variable B will also be set to 1. In line 60 the variable B will be output to Port A. The GOSUB 200 will still be a time delay as in the previous program. In line 80 the value of A is multiplied by two to shift it left. Then the variable B will be set equal to B plus A. The reason for this is to insure that all less significant lights will be lit whenever a more significant light is lit. Line 90 is used as a counter reset to re-initialize the variables when A reaches 256.

Programming the 6522 Internal Timer

The 6522 internal timer consists of two eight-bit latches and a 16-bit counter. The two latches are referred to as T1L/L and T1L/H. The 16-bit counter is divided into two eight-bit parts, referred to as T1C/L and T1C/H. The lower part of the counter T1C/L has a different function depending on whether you are reading or writing. Writing into T1C/L is the same as if you had written into T1L/L. It behaves much the same way as the memory location would. If you read T1C/L you will get the low byte of the counter. A write command to T1C/H will cause the counter to start. During this operation the contents of T1L/L are transferred to T1C/L. The contents of the counter T1C/L are decremented with each clock pulse received from B2. Each time the counter is decremented by one, a check is made to see if the counter has reached zero. If, after decrementing, the counter is zero, then one of two things will occur, depending on the operating mode that was set prior to initializing the counter sequence. Either an interrupt will be generated or bit 7 of Port B will be set. At this time the contents of T1L/L and T1L/H will be transferred into the counter again. This will have the effect of causing the timer to continously cycle. The operating mode is determined by setting bit 6 and bit 7 of the auxiliary control register ACR. The following table shows the different configurations possible and what the status of the operating mode is for each configuration.

ACR7	ACR6	Mode
0	0	Oneshot, only Interrupt, no Signal at PB7
0	I	Running Interrupts, no Signal at PB7
I	0	Oneshot, Interrupt, negative Pulse at PB7
I	I	Free running, square wave at PB7

Figure 1.11 Operating Modes of the Timer

Timer Operating Modes

If bit 6 of the auxiliary control register is 1 and bit 7 is also equal to 1, then the operating mode of the timer will be in a free-running or continously cycling state. Every time the lower 8 bits of the timer register become zero, the polarity of the signal at bit 7 of Port B will reverse. This causes pin 7 of Port B to act as a square-wave generator. The value entered into the timer controls the duration of the cycle of the square wave being generated. For instance, if a 2 is placed in the timer, a square wave with a 2 microsecond positive peak followed by a two microsecond negative peak will be generated, giving you a full cycle of 4 microseconds. The total square wave cycle generated will always be double the value placed in the timer. The following program listing is an example of making a square-wave generator using a 6522VIA board. The square-waves generated by this program will be 100 millisecond cycles.

Line Comments: Square-Wave Generator Using the 6522.

In lines 12-15 we use the pseudo-Op to equate and assign the values to the labels used in the program. In line 18 we set the operating mode with LDA C0H. In lines 20-22 we load the timer with the values to be used in this demonstration program. The timer will be loaded with **C47F** or 51023. Line 23 starts the timer. Note that in the listing, instead of putting 50,000 into the timer, we put 51023 in the timer. The reason for this is that the clock of the Apple II computer is not exactly one megahertz. You will be happy to hear that your Apple runs a little faster than advertised. Once the timer sequence has been initiated, the timer will continue to run without any help from the CPU and will run independently of whatever else is going on in the machine at that time. It will continue to run until the computer is reset, or forced through its power-up cycle, or the registers are changed. Any one of these three conditions signal the timer to stop its free-running or continous cycling mode. If you wish to change the frequency at which the program is running, you only need to load the new values into the two latches, T1L/L and T1L/H. When you load in the new values, the cycle of the square-wave already being generated will be completed. But once the timer has reached zero, the new values will be accepted, and the new frequency will be generated.

Another Project with the 6522 Timer

In this application we will use the timer as a single-shot or mono-flop square-wave pulse generator. In order to do this we need to change the operating mode from its current value of **C0** to a new value of **80**. The program listing to make the mono-flop or single-shot square-wave pulse generator follows:

Figure 1.12 Square-wave Generator

```
0800                   DCM "PR#1"
0800           ;
0800           ;
0800           ;****************************
0800           ;*                          *
0800           ;*  SQUAREWAVE GENERATOR WITH *
0800           ;*  A PERIOD OF 100.0 MS      *
0800           ;*                          *
0800           ;****************************
0800           ;
0800           ;
0800           ACR     EQU $C0CB
0800           T1CL    EQU $C0C4
0800           T1CH    EQU $C0C5
0800           MONITO  EQU $FF59
0800           ;
0800           ;
0800 A9C0              LDA #$C0            ;SET OPERATION MODE
0802 8DCBC0            STA ACR
0805 A94E              LDA #$4E            ;LOAD LO BYTE
0807 8DC4C0            STA T1CL
080A A9C4              LDA #$C4            ;LOAD HI BYTE
080C 8DC5C0            STA T1CH            ;AND START TIMER
080F 4C59FF            JMP MONITO
0812           ;
0812           ;
```

Figure 1.13 Monoflop

```
0800                   DCM "PR#1"
0800           ;
0800           ;
0800           ;****************************
0800           ;*                          *
0800           ;*     MONOFLOP/ONESHOT      *
0800           ;*                          *
0800           ;****************************
0800           ;
0800           ;
0800           ACR     EQU $C0CB
0800           T1CL    EQU $C0C4
0800           T1CH    EQU $C0C5
0800           IFR     EQU $C0CD
0800           ;
0800 A980      MONOFL  LDA #$80            ; SET OPERATIONMODE
0802 8DCBC0            STA ACR
0805 A94E              LDA #$4E            ; LOAD LO BYTE
0807 8DC4C0            STA T1CL
080A A9C4              LDA #$C4            ; LOAD HI BYTE
080C 8DC5C0            STA T1CH            ; START TIMER IFR6 SET TO 1
080F ADCDC0    M       LDA IFR
0812 2940              AND #$40
0814 F0F9              BEQ M
0816 60                RTS
0817           ;
0817           ;
                       END
```

Using the Timer as a Counter

The timer can be used to count negative pulses which appear on bit 6 of Port B. Bit 5 of the ACR determines whether the timer will be used as a mono-flop square-wave pulse generator or as a pulse counter. If this bit is set to 1, the timer will be a pulse counter, and if the bit is set to 0, it will be a mono-flop pulse generator. The following program will illustrate how to use one of the timers to generate a pulse that can be counted by the other timer. If we connect pin 6 of Port B to pin 7 of Port B, and we use timer 2 as the counter and timer 1 as a free-running continous cycle pulse generator, we can create an ideal timer to measure the running time of various routines and programs. The following demonstration program to illustrate using the timer as a stopwatch will consist of two parts: a short BASIC program and a machine-language program. The machine-language part sets the operating mode and starts the timer with its address at **C40C**. The two programs are very similar. The part of the program that will have the elapsed time in it starts at **C4C6**. The time value is stored as one-hundredth of a second and is stored in **C4FE** and **C4FF**. The BASIC program accesses this data, using it to calculate the amount of time that has elapsed during the running of the program. The machine-language program we are describing is stored in the RAM on the interface board, currently in slot 4. This makes it completely independent of BASIC and the rest of the memory in the machine, so you don't have to worry about it being overwritten by the BASIC programs you have running. Line 1000 is the test subroutine that we are going to measure the execution time of. In line 100 we start the time measurement. In line 110 we call the subroutine we are going to measure. When we return from the subroutine we call the routine to stop the timer; then the program goes to the routine that will calculate the amount of elapsed time that has occurred. Line 994 shows the routine that will calculate the time elapsed in hundredths of a second and then display it.

Figure 1.14 BASIC 'Running Time' Timer

```
1   REM  RUNTIME TEST
10 START =  - 15348:FIN =  - 15322:LO =  - 15106
15 HI =  - 15105
20 D$ =  CHR$ (4)
25  PRINT D$;"BLOAD ETIME"
100  CALL START
110  GOSUB 1000
200  CALL FIN: GOSUB 990: END
990  PRINT "EXECUTION TIME=";
992 H% =  PEEK (HI):L% =  PEEK (LO)
994  PRINT (H% * 256 + L%) / 100;" SECONDS"
999  RETURN
1000  REM  PROGRAM UNDER TEST
1010 Q = 2.5:B = 1.2:C = 3.4
1020 E = 1 / Q
1030  FOR I = 1 TO 100
1040 A = (B + C) * Q
1050  NEXT I
1060  RETURN
```

Listing Continued . .

Continued Listing

```
CALL-151

*C400LL

C400-   20 0C C4    JSR   $C40C
C403-   4C 59 FF    JMP   $FF59
C406-   20 26 C4    JSR   $C426
C409-   4C 59 FF    JMP   $FF59
C40C-   A9 E0       LDA   #$E0
C40E-   8D CB C0    STA   $C0CB
C411-   A9 01       LDA   #$01
C413-   8D C8 C0    STA   $C0C8
C416-   A9 00       LDA   #$00
C418-   8D C9 C0    STA   $C0C9
C41B-   A9 EC       LDA   #$EC
C41D-   8D C4 C0    STA   $C0C4
C420-   A9 13       LDA   #$13
C422-   8D C5 C0    STA   $C0C5
C425-   60          RTS
C426-   AD C8 C0    LDA   $C0C8
C429-   8D FE C4    STA   $C4FE
C42C-   AD C9 C0    LDA   $C0C9
C42F-   8D FF C4    STA   $C4FF
C432-   38          SEC
C433-   A9 00       LDA   #$00

 C400.C443

C400- 20 0C C4 4C 59 FF 20 26
C408- C4 4C 59 FF A9 E0 8D CB
C410- C0 A9 01 8D C8 C0 A9 00
C418- 8D C9 C0 A9 EC 8D C4 C0
C420- A9 13 8D C5 C0 60 AD C8
C428- C0 8D FE C4 AD C9 C0 8D
C430- FF C4 38 A9 00 ED FE C4
C438- 8D FE C4 A9 00 ED FF C4
C440- 8D FF C4 60
*
```

Programming the Internal Shift Register.

The internal shift register acts as a serial I/O Port. You can pass parallel information from the CPU to it and have it output it serially to an external peripheral device, or you can input serial data and then give it to the CPU in parallel, 8 bits at a time. In order to make the shift register function in this manner you can use an external clock, the clock of the CPU, or you could design your own timer clock pulse with the timer within the 6522. In this case the operating mode will be set by bits 2, 3 and 4 of the auxilary control register. The following table shows the different operating modes and the bit configuration that will set them.

ACR4	ACR3	ACR2	Mode
0	0	0	Shift Register Disabled
0	0	I	Shift in under control of Timer 2
0	I	0	Shift in at System Clock Rate
0	I	I	Shift in under control of external input pulses

Figure 1.15 Operating Modes of the Shift Register

The pin designated as CB2 on a 6522 is used as a serial I/O pin. Through this pin, serial I/O can be written to or read from the shift register. If you are going to use an external clock for your serial I/O you will need to feed the clock signal to CB1. In the internal clock you would use CB1 as a strobe to synchronize the data coming out of CB2 or going into CB2.

Whether CB1 is used as a sync pulse, outputs a clock pulse, or accepts an input of an external clock pulse depends on the bit configuration of bits 2, 3 and 4 of the auxilary control register (ACR). If you use the timer as your internal clock, it will only be an 8-bit timer used in conjunction with the shift register. The lowest shift frequency would then be about 0.5 milliseconds because reads or writes to the shift register can only be done on every other occurance of zero.

A Variable Duty-cycle Square-wave Generator.

Changing the bit configuration and shift register will alter the duty cycle of the square-wave being generated. Changing the counter latch, T2L/L, allows you to change the clock frequency of the square-wave generator. The following program, written in FORTH, you can use to control the 8 output pins of Port A. This program in the FORTH language is included because FORTH is a very common language in control applications. Also, writing a program in FORTH is much easier than writing in machine-language, and much faster than a BASIC program would be. To demonstrate this program we will perform the following tasks. There are 8 LED's connected to Port A of the 6522 chip. Instead of using LED's, any device could be connected provided there were an interface to assure the voltages were proper for operating the external device, without drawing too much current from the computer. The LED's are numbered from 1 to 8. LED 1 is controlled by bit 0 of Port A, the least significant bit and LED 8 is controlled by bit 7, the most significant bit of Port A. This program will make it possible to turn the LED's on and off by simply typing the number of the LED followed by the word ON or OFF.

The following is the line comments of the FORTH program. In the first line of the program we define the word START. This will set the data direction register for Port A, located at memory address **COC3**, with 255, signaling that it is to be used for output. We put zero as the first element on the top of the stack. In the second line we define the word AN, and we put it into location **COC1**. In the third line we define the variable NR as the number of the LED that should be switched on or off. Before calling NR, this number is on top of the stack. Entering the DO loop, the top of the stack is 1, and N is the upper boundary of the index limit of the

Figure 1.16 Variable Square-wave Generator

```
PR#1

0800                    DCM "PR#1"
0800            ;
0800            ;
0800            ;****************************
0800            ;*                          *
0800            ;* VARIABLE DUTY CYCLE      *
0800            ;* SQUAREWAVE GENERATOR     *
0800            ;*                          *
0800            ;****************************
0800            ;
0800            ;
0800            ;
0800            ACR     EQU $C0CB
0800            T2LL    EQU $C0C8
0800            SR      EQU $C0CA
0800            MONITO  EQU $FF59
0800            ;
0800 A9FF               LDA #$FF          ;SET TIMER 2 FOR SLOWEST
0802 8DC8C0             STA T2LL          ;FREQUENCY
0805 A910               LDA #$10          ;SET OPERATION MODE
0807 8DCBC0             STA ACR
080A A90F               LDA #$0F          ;4 TIMES ZERO AND 4 TIMES
080C 8DCAC0             STA SR            ;ONE TO THE SR
080F 4C59FF             JMP MONITO
0812            ;
0812            ;
                       END

***** END OF ASSEMBLY
```

Figure 1.17 FORTH Listing – Lamp Driver

```
: START HEX 00 FF C0C3 1! ;
: AN C0C1 ! ;
: NR 1 0 2 UNDER SWAP DO 2* LOOP 2/ ;
: NEW 2 UNDER OR DUP ;
: ON NR NEW AN ;
: NEC 2 UNDER SWAP COMPLEMENT AND DUP ;
: OFF NR NEC AN ;

START
2 ON
3 ON
2 OFF
```

loop. In the loop, the 1 on the top of the stack will be shifted left N number of times by multiplying by two, in order to indicate which LED is the target. For example, with N = 4 we set bit 4 of Port A to 1. This bit is assigned to LED 5. This is one too high, so we must shift right one time. This is done by dividing by 2. If you switch on another LED, all LED's that are already on should stay on. To switch the lamp off it

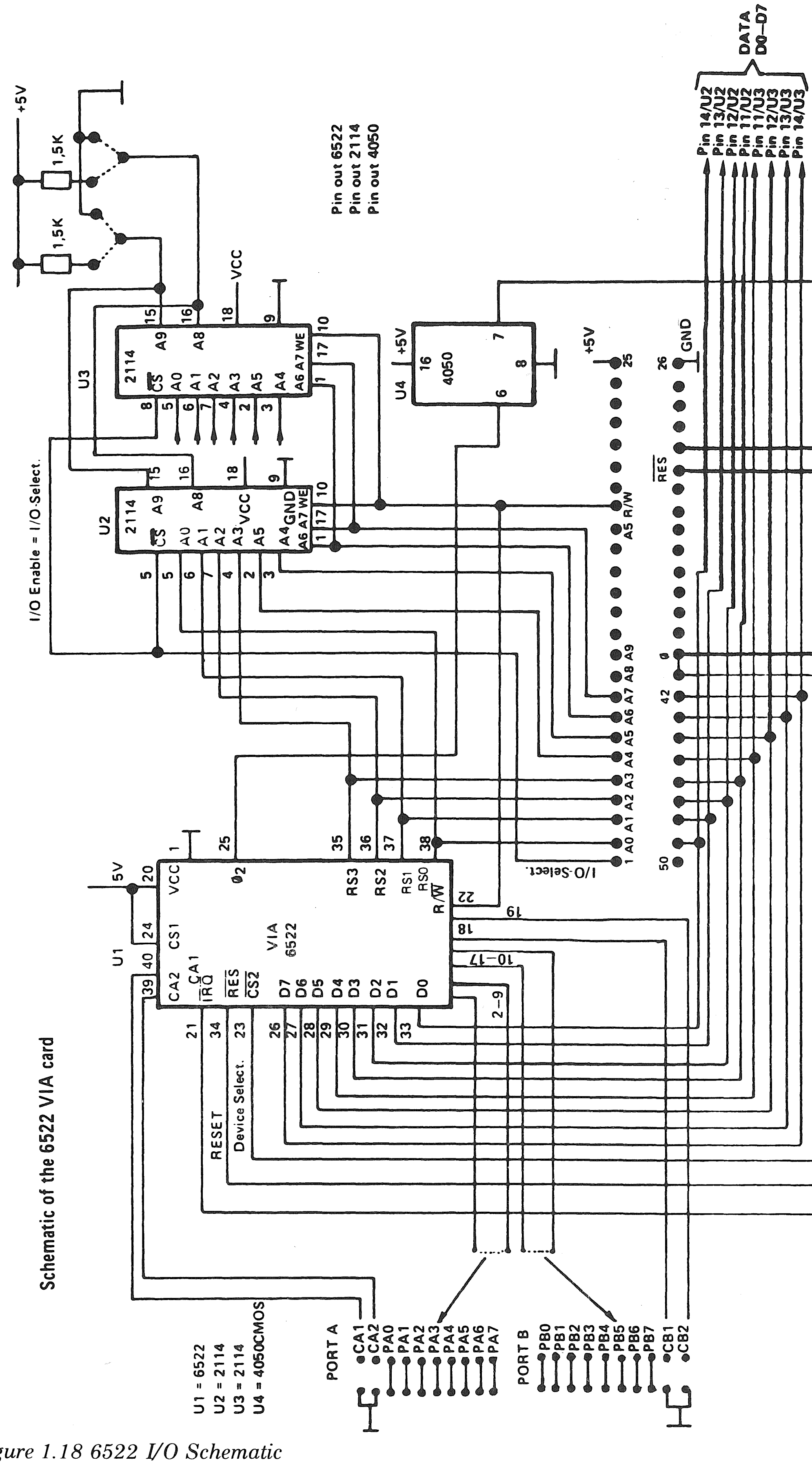
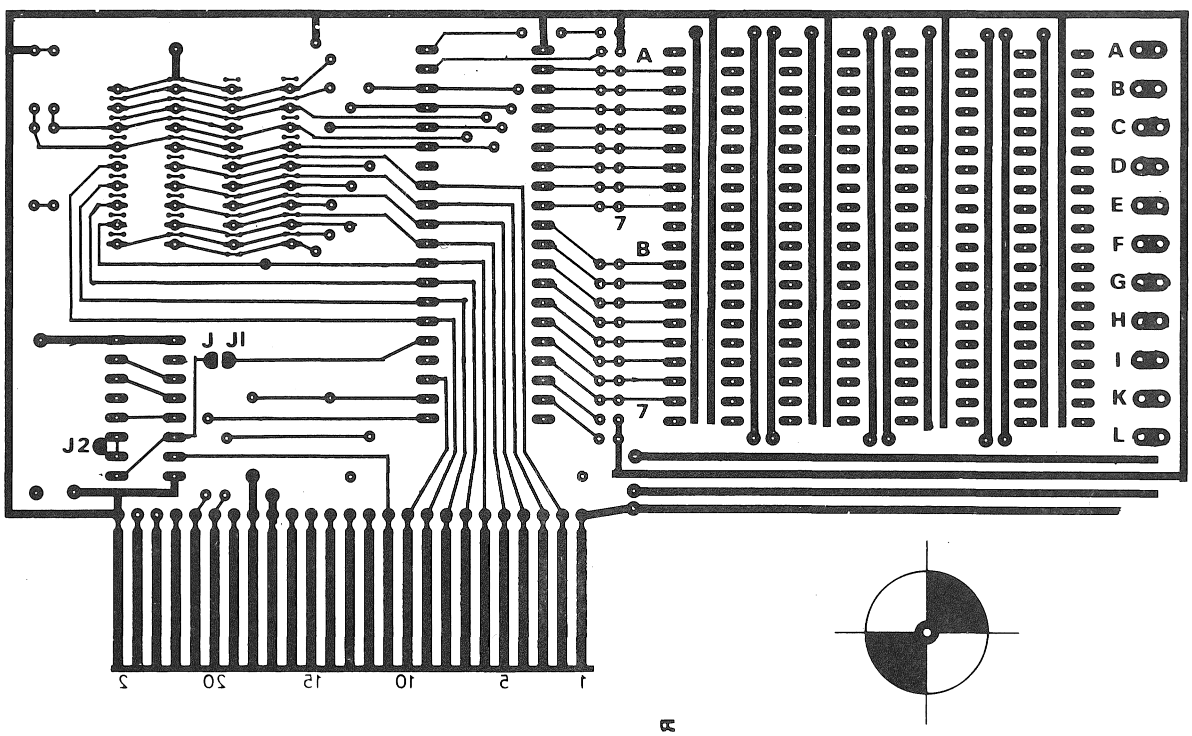
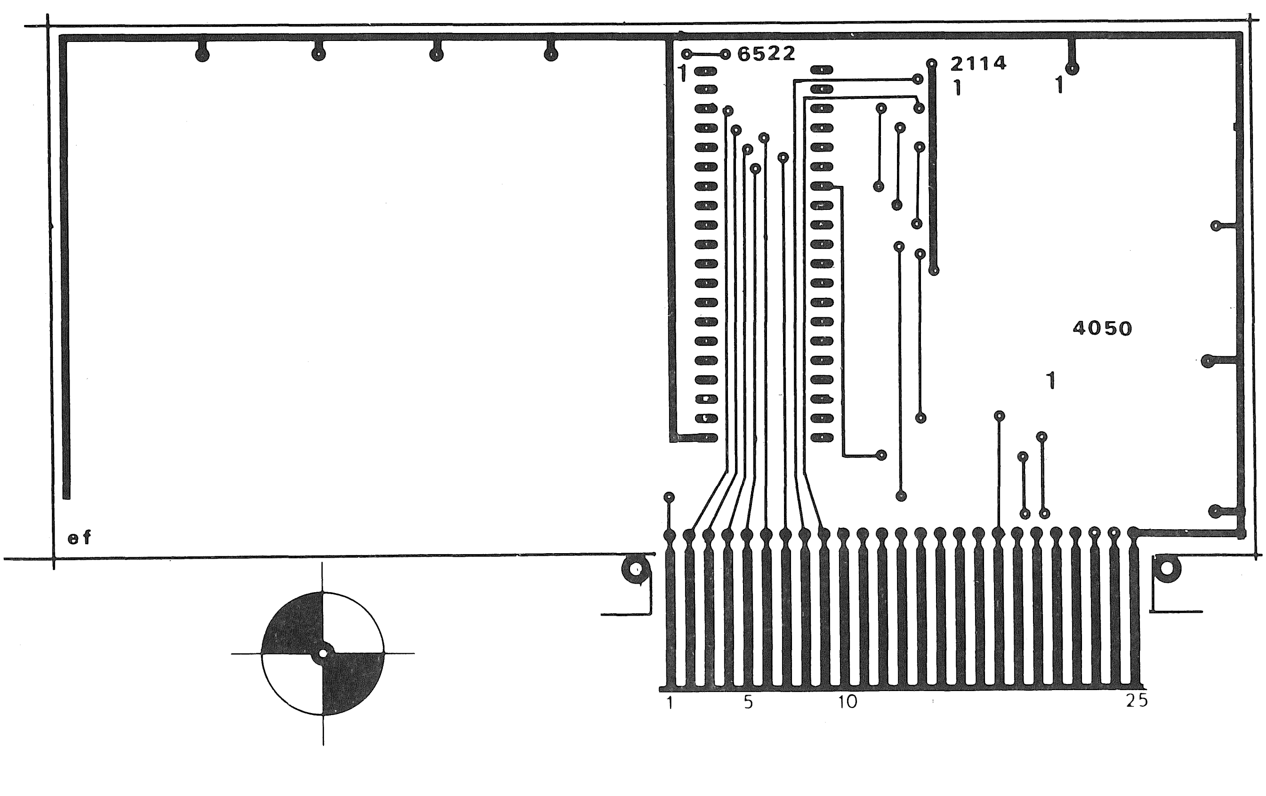


Figure 1.18 6522 I/O Schematic



(Bottom)

Figure 1.19 Printed Circuit Board



(Top)

"Is written on a blank page to avoid confusion" is written on a blank page to avoid confusion . . .

is necessary to complement the number used to switch it on; then erase the bit by doing an AND function to mask out the unwanted bit of the existing pattern. This is done in program part NEC. You can turn out LED 5 by typing in 5 OFF. The program is started by the word START, which initializes all of the ports of the 6522.

In Figure 1.18 you see the complete schematic of the 6522 I/O board. The two RAM'S are located in the upper right hand corner (if you are holding the board as though you were plugging it into the machine). They are numbered U2 and U3. They are selected by the IL select line from the Apple. The 6522 is selected by the device select signal from the Apple computer. The select lines RS0 to RS3 are connected to address lines A0 to A3. The U4, as previously mentioned, gives us the time delay for the Phi 2 clock. The output lines are brought out to two different connectors. You can identify each set on the left hand side by looking at the schematic.

Constructing the 6522 I/O Board.

The I/O board is available in kit form from Technopak. A picture of parts placement is provided with all the parts in the places where they should go, and we recommend putting each IC in a socket. There are also two places where you will have to attach jumper wires as shown in the parts placement figure.

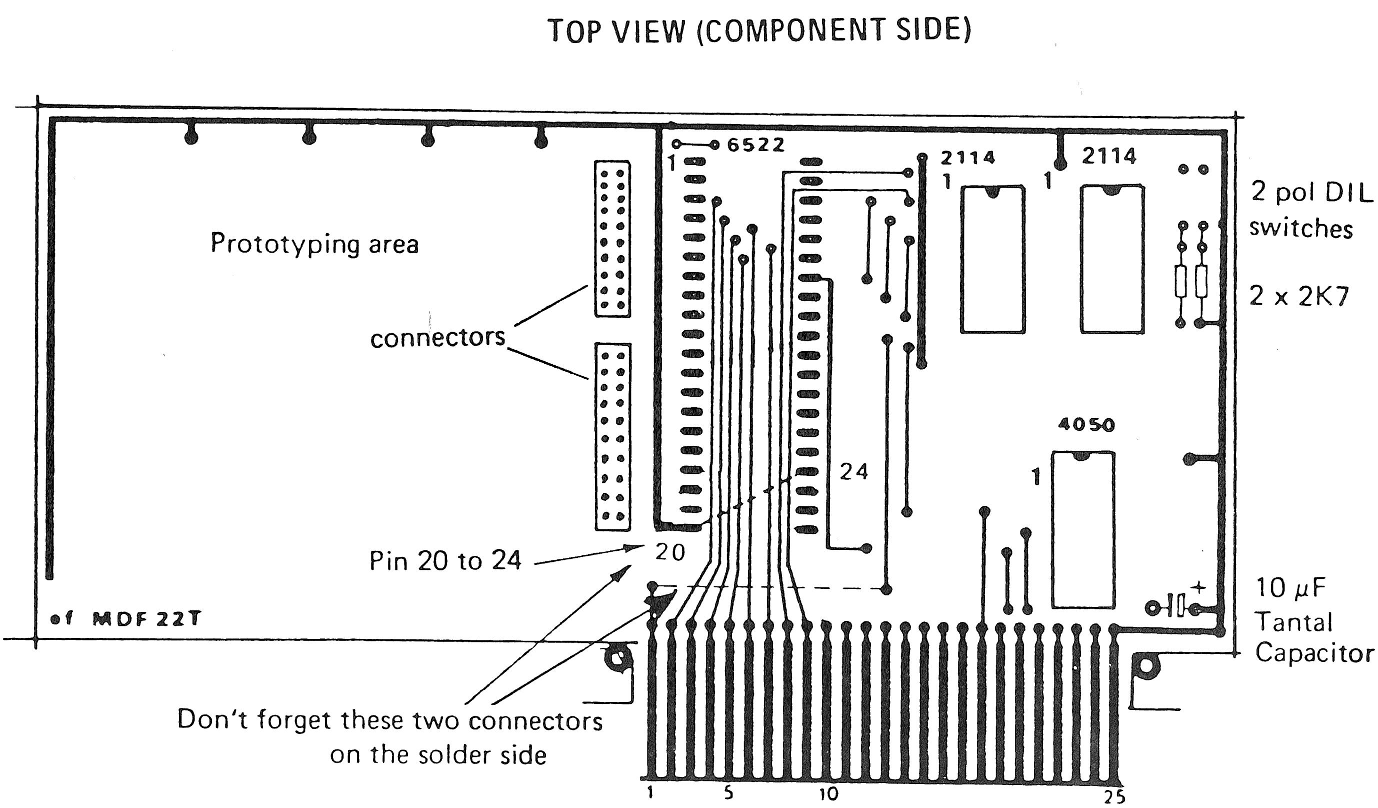


Figure 1.20 Component Layout

Figure 1.21 Parts List for the 6522 I/O Board

Qty	Description
1	Capacitor tantal 10 μF/35V
1	DIP switch, 2 poles / 3 poles
2	Connectors with 20 pin each, for port A and B connectors
1	40 pin socket DIL
2	18 pin socket DIL
1	16 pin socket DIL
1	6522 VIA (Rockwell)
1	4050 Motorola
2	2114 L RAM chips Synelec or Rockwell
1	6522 / I / Board

2

Sound and Noise Generation Using the AY-3-8912

The PSG (Programmable Sound Generator) generates sound or noise through mixing of three programmable square-wave frequencies and one noise generator. Using a D/A convertor, all three frequencies are output on three different channels. Each of the output channels can be connected to an amplifier separately, or all three channels can be tied together through one amplifier. The envelope of the output signal can be controlled by an envelope generator. All functions are controlled by 16 registers shown in the table in Figure 2.1.

REGISTER	BIT	B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-BIT Fine Tune A							
R1						4-BIT Coarse Tune A			
R2	Channel B Tone Period	8-BIT Fine Tune B							
R3						4-BIT Coarse Tune B			
R4	Channel C Tone Period	8-BIT Fine Tune C							
R5						4-BIT Coarse Tune C			
R6	Noise Period				5-BIT Period Control				
R7	$\overline{\text{Enable}}$	$\overline{\text{IN}}$/OUT		$\overline{\text{Noise}}$			$\overline{\text{Tone}}$		
		IOB	IOA	C	B	A	C	B	A
R10	Channel A Amplitude				M	L3	L2	L1	L0
R11	Channel B Amplitude				M	L3	L2	L1	L0
R12	Channel C Amplitude				M	L3	L2	L1	L0
R13	Envelope Period	8-BIT Fine Tune E							
R14		8-BIT Coarse Tune E							
R15	Envelope Shape/Cycle					CONT.	ATT.	ALT.	HOLD
R16	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							
R17	I/O Port B Data Store	8-BIT PARALLEL I/O Port B							

Figure 2.1 PSG Register Functions

The generation of a single tone is performed by frequency division. A clock signal, which has to be applied to the chip, must first be divided by 16, and then it will be divided by 12 using a counter. This 12 bit word for channel A will now be put into register R0 (8 lower bits), with the remaining 4 bits put into register R1. For a given clock frequency you can calculate the tone period (tp) as follows:

tp = fclock/(f*16)

f = the desired frequency
fclock = clock frequency applied to the chip
Both values used are in HZ

Example: f = 440 HZ
fclock = 1,000,000 HZ
tp = 1,000,000/440*16 = 142.04

If you convert 142 into a 12-bit binary number, you will get **8E** (in HEX). With an **8E** in register R0 and a 0 in register R1, you will get a signal with a frequency of 440 HZ. The rounding of 142.04 gives you an error of course, so the resulting frequency will be 440.14 HZ. The difference between the calculated and real frequency at different clock frequencies is shown in the following table:

Frequency	1 MHz	1.78977 MHz
1046.496 (C6)	1041.666	1045.428
7040.00 (A8)	6944.444	6991.299

Figure 2.2 Frequencies

To calculate the HEX numbers for the different clock frequencies, you may use the following table:

Figure 2.3 Clock Frequencies

```
LIST

10   REM   CALCULATING THE CONTENTS OF THE REGISTERS
20   REM  FOR THE PSG AY-3-8912
30   REM   CLOCKFREQUENCY 1MHZ (FC)
40   REM  OUTPUT OF THE 12-BIT VALUES IN HEX
50   REM  DESIRED AND TRUE FREQUENCY IS PRINTED
100  INPUT "F= ";F
110 FC = 1000000
120 TP = FC / (16 * F)
130 MSD =  INT (TP / 256)
140 TP = TP - MSD * 256
150 NSD =  INT (TP / 16)
160 LSD =  INT (TP - NSD * 16 + 0.5)
165 FI = FC / ((MSD * 256 + NSD * 16 + LSD) * 16)
170  GOSUB 200
180  END
200  IF MSD > 9 THEN MSD = MSD + 7
210 MSD = MSD + 48:A$ =  CHR$ (MSD)
220  IF NSD > 9 THEN NSD = NSD + 7
230 NSD = NSD + 48:B$ =  CHR$ (NSD)
240  IF LSD > 9 THEN LSD = LSD + 7
250 LSD = LSD + 48:C$ =  CHR$ (LSD)
260  PRINT F;"   ";A$;B$;C$;"    ";FI
270  RETURN
```

The next figure shows how to generate a clock frequency with a 3.579545 MHZ crystal. Then the signal is divided using the CMOS chip (4013). In most applications, it will be more than sufficient to use the 1 MHZ clock of your computer system.

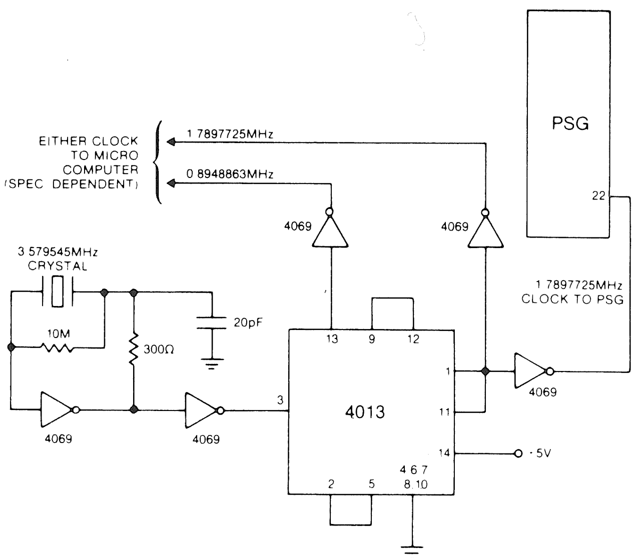


Figure 2.4 Clock Generator Circuit

How the Internal Registers Work.

The registers R0 - R5 are used to program tone periods for the three channels A, B, and C. Register R6 is used to program the noise generator; therefore, you only have to use the 5 lowest bits of this register. The lowest noise frequency will be achieved by placing a **1F** into the lowest 5 bits (All 5 bits are 1). The highest possible noise frequency is created by using a **01** in that part of the register. The clock frequency is now divided, first by 16, then by the 5-bit word. The noise period may be calculated with the following equation:

$$NP = fclock/(16*fn)$$

fclock = input clock frequency
NP = noise period.
fn = desired noise frequency

With a clock frequency of 1 MHZ you can generate noise within a range from 2 MHZ - 75 MHZ. Register 7 controls the sound and noise output of each separate channel. How the sound channels work with the sound or noise output is shown in the following chart:

Bit	7	6	5	4	3	2	I	0
	I/O		Noise			Sound		
			C	B	A	C	B	A

Figure 2.5

When one bit of register A is set to zero (0) the appropriate channel is opened.

Example: Sound on channel A = 00111110 = **3E**
Noise on channel B and
Sound on channels A and C = 00101010 = **2A**

The two most significant bits are used for the data transfer via the I/O port of the PSG chip. You don't need them for sound generation. Registers R8, R9 and R10 are responsible for the value of the sound output of channels A, B and C respectively. The first 4 bits set the volume to one of 16 different levels for each channel. This setting is not linear; rather, it is logarithmic.

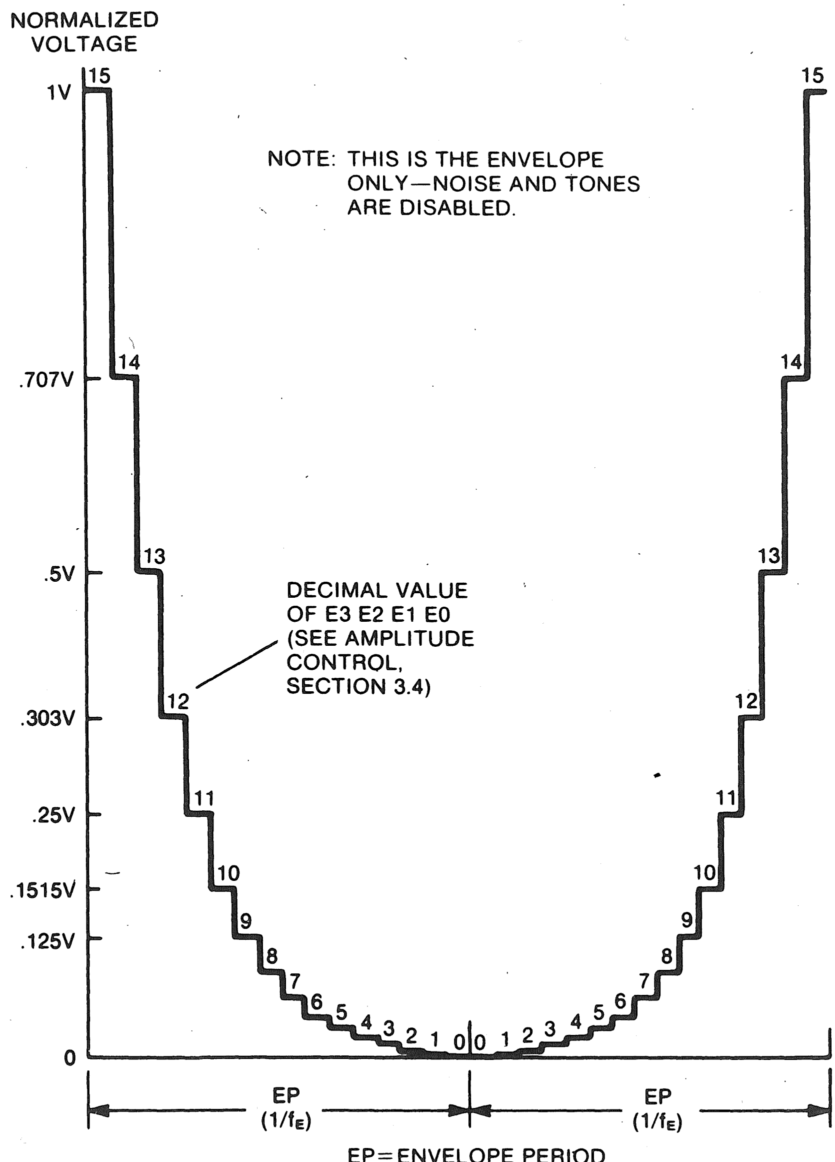


Figure 2.6 Envelope Period

If, in one of these registers, bit 5 is set to a logical 1, the amplitude of that channel is controlled by the envelope generator, which can be programmed via registers R11, R12 and R13. R11 and R10 form a 16-bit counter to generate the length of the period of the envelope. The clock frequency is divided by 256 and then by the contents of registers R11 and R12. R12 is now the least significant bit.

A 1 MHZ clock frequency gives you envelope periods from 0.06 HZ to 4000 HZ. To calculate the period use:

EP = fclock/(256*fe)

fe = frequency of the envelope
EP = Envelope Period (or Duration)

The 16 bit binary value for EP is written into registers R11 and R12. For that calculation, use the program above after changing line 120 to EP = FC/(256*F). The least significant bit of R13 defines the configuration of the envelope.

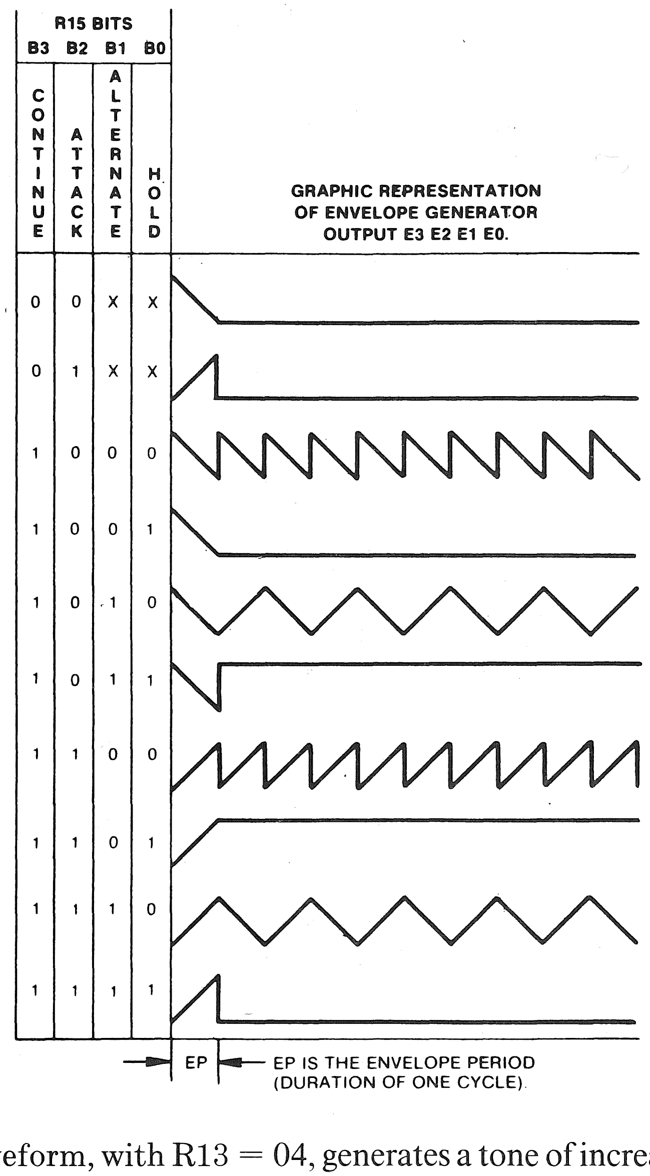


Figure 2.7 Envelopes

The second waveform, with R13 = 04, generates a tone of increasing volume with a period of EP. At the end of period EP the volume will suddenly decrease.

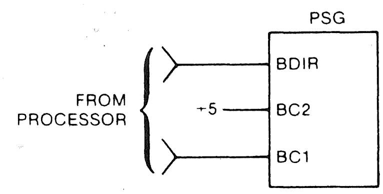
Programming the GI Soundchip.

Control lines BDIR and BC2 are used to select a register. The third control line is connected to +5V. Data lines and control lines can be controlled by the 6522 VIA.

In our application we used the Phi 2 clock of the 6502 microprocessor for our sound chip clock.

The data lines, DA0 - DA7, are connected to Port A of the 6522. The control lines BC1 and BDIR are hooked to PB0 and PB1. To feed the data into the appropriate register, you first have to send the address and data through the data lines. The data lines are controlled by the control lines BDIR and BC1 (see Figure 2.8).

BDIR	BC2	BC1	PSG FUNCTION
0	1	0	INACTIVE.
0	1	1	READ FROM PSG.
1	1	0	WRITE TO PSG.
1	1	1	LATCH ADDRESS.



ANALOG CHANNEL A, B, C (outputs): pins 4, 3, 38 (AY-3-8910)
pins 5, 4, 1 (AY-3-8912)

Figure 2.8 PSG Functions

The number of the appropriate register is stored in the X register, and the data is stored in the accumulator of the 6502 CPU and then passed to the subroutine called OUT.

Figure 2.9 Program OUT

```
PR#1

0800                      DCM  "PR#1"
0800              ;
C0C0                      ORG  $C0C0
C0C0              TORB    EQU  *
C0C0              TORA    EQU  *+!1
C0C0              DDRB    EQU  *+!2
C0C0              DDRA    EQU  *+!3
C0C0              ;
0800                      ORG  $800
0800 A8           OUT     TAY             ;<A> --> YREG
0801 A9FF                 LDA  #$FF       ;PORTA AND B ARE QUTPUTS
0803 8DC3C0               STA  DDRA
0806 8DC2C0               STA  DDRB
0809 8EC1C0               STX  TORA       ;OUTPUT ADDRESS
080C A903                 LDA  #$03       ;BDIR UND BC1 =1
080E 8DC0C0               STA  TORB
0811 A900                 LDA  #$00       ;BDIR UND BC1 =0
0813 8DC0C0               STA  TORB
0816 98                   TYA             ;<Y> --> AKKU
0817 8DC1C0               STA  TORA
081A A902                 LDA  #$02       ;BDIR=1 BC1=0
081C 8DC0C0               STA  TORB
081F A900                 LDA  #$00       ;BDIR=0 BC1=0
0821 8DC0C0               STA  TORB
0824 60                   RTS
```

Listing Continued . . .

The PSG at this time is not enabled. When the address is outputted, BDIR and BC1 go high for a very short period of time; when the data is outputted, only BDIR goes high.

Another way to program the PSG is to put the contents of the register into a table. Then you can use a program to write the values into the PSG.

Continued Listing

```
0825              ;
0825              TAB     EQU $1000
0825              ;
0825 A200         LOAD    LDX #$00
0827 BD0010       M       LDA TAB,X
082A 200008               JSR OUT
082D E8                   INX
082E E010                 CPX #16
0830 D0F5                 BNE M
0832 60                   RTS
```

The programs we have seen so far only affect the registers of the sound chip. To generate sound and noise you need a few more program parts. They will be comprised substantially of delay routines and checking procedures. Program WAIT in Fig. 2.11 shows such a delay loop.

Figure 2.11 Program WAIT

```
0833              ;
0833 38           WAIT    SEC
0834 48           W2      PHA
0835 E901         W3      SBC #$01
0837 D0FC                 BNE W3
0839 68                   PLA
083A E901                 SBC #$01
083C D0F6                 BNE W2
083E 60                   RTS
083F              ;
083F              ;
```

Example: Generating sound A with highest volume on channel A.

Figure 2.10 Generating Tone A

```
083F              ;
083F A98E                 LDA #$8E          ;440 HZ   AT FT=1MHZ
0841 A200                 LDX #$00
0843 200008               JSR OUT
0846 A93E                 LDA #$3E          ;SOUND ONLY ON CHANNEL A
0848 A207                 LDX #7
084A 200008               JSR OUT
084D A90F                 LDA #$0F          ;VOLUME SET TO MAXIMUM
084F A208                 LDX #8
0851 200008               JSR OUT
0854 00                   BRK
0855              ;
```

Via channel A, for approximately 1 second, a 440 HZ tone is outputted; after that a tone of 187 HZ is generated for 1 second (assuming the clock frequency is 1 MHZ). We use it in the following program called SIREN.

Figure 2.12 Program SIREN

```
0855                ;
0855                ;
0855 A93E           SIREN   LDA #$3E            ;ONLY CHANNEL A
0857 A207                   LDX #7
0859 200008                 JSR OUT
085C A90F                   LDA #$0F            ;VOLUME SET TO MAXIMUM
085E A208                   LDX #8
0860 200008                 JSR OUT
0863 A98E           S       LDA #$8E            ; 440 HZ
0865 A200                   LDX #$00
0867 200008                 JSR OUT
086A A900                   LDA #$00
086C A201                   LDX #01
086E 200008                 JSR OUT
0871 A9FF                   LDA #$FF
0873 203308                 JSR WAIT            ;WAIT FOR 350 MS
0876 A901                   LDA #$01            ;187 HZ
0878 A201                   LDX #$01
087A 200008                 JSR OUT
087D A94E                   LDA #$4E
087F A200                   LDX #$00
0881 200008                 JSR OUT
0884 A9FF                   LDA #$FF
0886 203308                 JSR WAIT
0889 18                     CLC
088A 90D7                   BCC S
088C                ;
```

Programming a Gunshot.

To simulate a gunshot, you only need the noise generator for the envelopes. We set up a table in memory, and if a button is pushed, the contents of the table are brought into the PSG. If you change the content of location **1006** (noise frequency) to **00** (highest noise period) and location **100C** to **40** (envelope approximately 2 seconds), you can simulate an explosion.

Figure 2.13 Program GUNSHOT

```
088C                ;
088C                KEY     EQU $FD35
088C                ;
088C 202508         SHOT    JSR LOAD
088F 2035FD                 JSR KEY
0892 18                     CLC
0893 90F7                   BCC SHOT
0895                ;
0895                ;
1000                        ORG $1000
1000 000000                 HEX 000000000000    ;NO SOUND
1003 000000
1006 0F                     HEX 0F              ;MEDIUM  NOISE FREQUENCY
1007 07                     HEX 07              ;NOISE ON ALL CHANNELS
1008 101010                 HEX 101010          ;VOLUME SET TO MAXIMUM
100B 0010                   HEX 0010            ;ENVELOP PERIOD 0.6 S
100D 00                     HEX 00              ;ONLY ONE CYCLE
                            END
```

HEX dump of all the demo programs with the following starting addresses:

083F . . . SOUND
0855 . . . SIREN
088C . . . GUNSHOT

```
0800- A8 A9 FF 8D C3 C0 8D C2
0808- C0 8E C1 C0 A9 03 8D C0
0810- C0 A9 00 8D C0 C0 98 8D
0818- C1 C0 A9 02 8D C0 C0 A9
0820- 00 8D C0 C0 60 A2 00 BD
0828- 00 10 20 00 08 E8 E0 10
0830- D0 F5 60 38 48 E9 01 D0
0838- FC 68 E9 01 D0 F6 60 A9
0840- 8E A2 00 20 00 08 A9 3E
0848- A2 07 20 00 08 A9 0F A2
0850- 08 20 00 08 00 A9 3E A2
0858- 07 20 00 08 A9 0F A2 08
0860- 20 00 08 A9 8E A2 00 20
0868- 00 08 A9 00 A2 01 20 00
0870- 08 A9 FF 20 33 08 A9 01
0878- A2 01 20 00 08 A9 4E A2
0880- 00 20 00 08 A9 FF 20 33
0888- 08 18 90 D7 20 25 08 20
0890- 35 FD 18 90 F7 90
*

 11000.100D

1000- 00 00 00 00 00 00 0F 07
1008- 10 10 10 00 10 00
*
```

Program PIANO

This program simulates the sound of a piano. The keys 1 - 8 refer to the musical notes of the C scale. The table of that program is placed in memory area **1010** to **1017**. Each tone is mixed with a tone of half the frequency and a tone which differs slightly from the basic tone. Then a descending envelope with about a 0.85-second period is superimposed. The program starts at **0900** and uses the routines OUT, LOAD and KEY.

Figure 2.14 Program PIANO

```
0800                          DCM "PR#1"
0800                ;
C0C0                          ORG $C0C0
C0C0                TORB      EQU *
C0C0                TORA      EQU *+!1
C0C0                DDRB      EQU *+!2
C0C0                DDRA      EQU *+!3
C0C0                ;
C0C0                KEY       EQU $FD35
C0C0                ;
0800                          ORG $800
0800 A8             OUT       TAY             ;<A> --> YREG
0801 A9FF                     LDA #$FF        ;PORTA AND B ARE OUTPUTS
0803 8DC3C0                   STA DDRA
0806 8DC2C0                   STA DDRB
0809 8EC1C0                   STX TORA        ;OUTPUT ADDRESS
080C A903                     LDA #$03        ;BDIR UND BC1 =1
080E 8DC0C0                   STA TORB
0811 A900                     LDA #$00        ;BDIR UND BC1 =0
0813 8DC0C0                   STA TORB
0816 98                       TYA             ;<Y> --> AKKU
0817 8DC1C0                   STA TORA
081A A902                     LDA #$02        ;BDIR=1 BC1=0
081C 8DC0C0                   STA TORB
081F A900                     LDA #$00        ;BDIR=0 BC1=0
0821 8DC0C0                   STA TORB
0824 60                       RTS
0825                ;
0825                ;
0825 A200           LOAD      LDX #$00
0827 BD5B08         M         LDA TAB,X
082A 200008                   JSR OUT
082D E8                       INX
082E E010                     CPX #16
0830 D0F5                     BNE M
0832 60                       RTS
0833                ;
0833 38             WAIT      SEC
0834 48             W2        PHA
0835 E901           W3        SBC #$01
0837 D0FC                     BNE W3
0839 68                       PLA
083A E901                     SBC #$01
083C D0F6                     BNE W2
083E 60                       RTS
```

Listing Continued . .

Continued Listing

```
083F              ;
083F              ;
083F 2035FD       PIANO   JSR KEY
0842 290F                 AND #$0F
0844 AA                   TAX
0845 CA                   DEX
0846 BD6B08               LDA FTAB,X
0849 8D5B08               STA TAB
084C AA                   TAX
084D CA                   DEX
084E 8E5D08               STX TAB+2
0851 4A                   LSR
0852 8D5F08               STA TAB+4
0855 202508               JSR LOAD
0858 4C3F08               JMP PIANO
085B              ;
085B              ;
085B 000000       TAB     HEX 000000000000     ; FILLED BY PROGRAM
085E 000000
0861 0038                 HEX 0038             ;SOUND ON ALL CHANNELS
0863 101010               HEX 101010           ;VOLUME SET TO MAXIMUM
0866 000A00               HEX 000A00           ;ENVELOPE DECAY 0.8 S
0869 0000                 HEX 0000
086B EFD5BE       FTAB    HEX EFD5BEB39F8E7F75 ;FREQUENCY TABLE
086E B39F8E
0871 7F75
                  FIN     END
```

```
0800- A8 A9 FF 8D C3 C0 8D C2
0808- C0 8E C1 C0 A9 03 8D C0
0810- C0 A9 00 8D C0 C0 98 8D
0818- C1 C0 A9 02 8D C0 C0 A9
0820- 00 8D C0 C0 60 A2 00 BD
0828- 5B 08 20 00 08 E8 E0 10
0830- D0 F5 60 38 48 E9 01 D0
0838- FC 68 E9 01 D0 F6 60 20
0840- 35 FD 29 0F AA CA BD 6B
0848- 08 8D 5B 08 AA CA 8E 5D
0850- 08 4A 8D 5F 08 20 25 08
0858- 4C 3F 08 00 00 00 00 00
0860- 00 00 38 10 10 10 00 0A
0868- 00 00 00 EF D5 BE B3 9F
0870- 8E 7F 75
```

Figure 2.15 BASIC Sound Demo

```
100   POKE 687,169: POKE 688,3
110   POKE 689,141: POKE 690,192: POKE 691,192
120   POKE 692,169: POKE 693,0
130   POKE 694,141: POKE 695,192: POKE 696,192
140   POKE 697,96
150   POKE  - 16190,255: POKE  - 16189,255
200   DIM D(14)
210   HOME : HTAB (3): VTAB (5)
220   PRINT "SOUND DEMO"
230   FOR X = 1 TO 3000: NEXT
240   READ G$
250   HTAB (3): PRINT G$
260   GOSUB 500
270   FOR X = 1 TO 5000: NEXT
280   IF G$ = "SUEF" THEN  FOR X = 1 TO 10000: NEXT
290 Y = Y + 1: IF Y < 5 THEN 320
300 A = 7:D(A) = 255: GOSUB 1000
310 Y = 0: RESTORE : GOTO 210
320 A = 7:D(A) = 255: GOSUB 1000
330  GOTO 240
500   FOR A = 0 TO 13
510   READ D(A)
520   GOSUB 1000
530   NEXT A
540   RETURN
1000  POKE  - 16192,0: POKE  - 16191,A
1010  POKE 688,3: CALL 687
1020  POKE  - 16192,0: POKE  - 16191,D(A)
1030  POKE 688,2: CALL 687
1040  RETURN
2000  DATA  "PIANO",200,0,201,0,100,0,0,248,16,16,16,0,20,8
2010  DATA   "EXPLOSION",0,0,0,0,0,0,31,7,16,16,16,0,20,0
2020  DATA "GUNSHOT" ,0,0,0,0,0,0,15,7,16,16,16,0,16,0
2030  DATA   "LOCOMOTIVE",0,0,0,0,0,0,15,199,16,16,16,180,2,12
2040  DATA   "SURF",0,0,0,0,0,0,31,199,16,16,16,16,255,60,14
```

Sound-DEMO for the AY-3-8912

This program shows you how to program the register in the GI sound chip in BASIC. The contents of the registers R0 - R13 are placed in data statements. The special feature of this program is that it contains a machine-language routine which supplies the pulse for bringing the information over to the sound chip. During program development, we found that a pulse which was generated with a POKE command in BASIC was too slow and caused unpredictable functions in the AY-3-8912 chip.

Program Description:

Lines 100 - 150 : Pokeing the machine-language
Line 200 : Setting the data direction registers
Lines 210 - 330 : Waiting loops and reading of the data
Lines 1000 - 1040 : Filling the registers with the data D(A) using the machine-language routine
Lines 2000 - 2040 : Data for the different sounds

Assembling a Sound Generator Board

To construct your sound generator board, you first have to assemble the 6522 VIA board previously described in this book. Then you use the prototyping area on the left-hand side of the board to assemble the sound circuitry. Place the AY-3-8912 sound chip so that the input lines DA0 - DA7 match with the outputs PA0 - PA7 of the 6522 VIA (See schematic). Next you cut the lines which connect the sound chip to the pins PB0 - PB3 (four lines). Pin 20 of the sound chip has to be connected to pin 10 of the 6522; pin 19 to +5V; pin 18 to pin 1 of the 6522; pin 17 to +5V; pin 16 to pin 34 of the 6522; pin 15 to pin 25 of the 6522; pin 6 to ground and pin 3 to +5V.

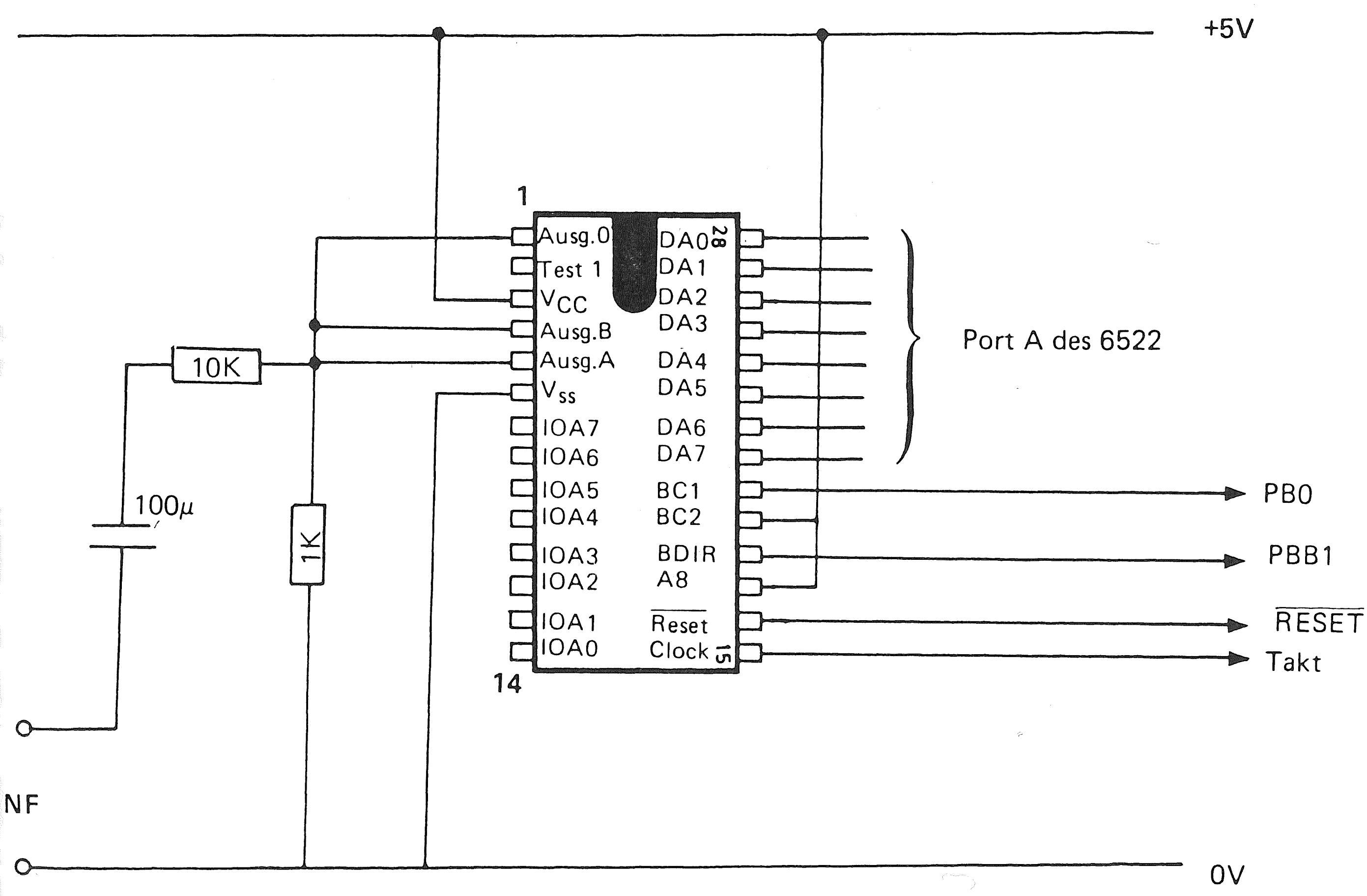


Figure 2.16 Schematic of the Sound Board

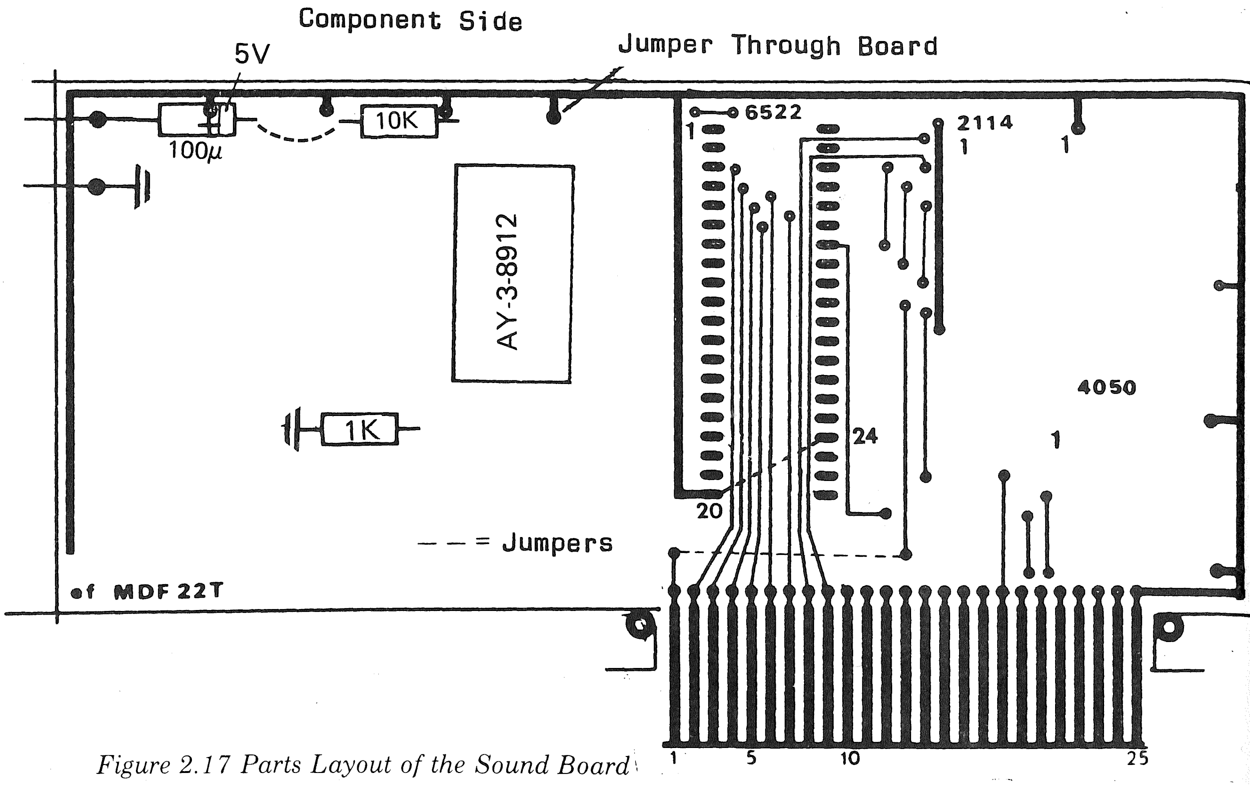


Figure 2.17 Parts Layout of the Sound Board

Pins 1, 4 and 5 are the common output of the AY-3-8912. You can hook them to the next convenient foil on the PC board. From this foil, connect a 1K resistor to ground. Then connect a 10,000 Ohm resistor with a 100 microfarad capacitor to the output which goes to your audio amplifier. At the 6522 VIA chip, connect pin 2 with pin 20. On the component side of the PC board you need jumpers (see schematic) and a wire through the board to bring the +5V supply voltage over from the soldering side.

3

An 8-Bit D/A and A/D Convertor

This chapter outlines an application using a digital to analog and analog to digital convertor. Our first project will be an 8-bit digital to analog convertor using the Ferranti Digital to Analog convertor kit (ZN428E). If you want to use your Apple II personal computer for data acquisition, sensing conditions and controlling systems in the home or industrial environment, you will often have to convert a certain number-value into a voltage level (a digital/analog conversion). For instance, if you want to convert a certain voltage level with your program, you have to generate a digital number first, then convert this digital number into a voltage level. The value of the digital number has to be made such that after converting it, the appropriate voltage level is achieved. The opposite of this function is the analog to digital convertor, which converts a voltage level into a digital number. Those conversions can be performed with the digital-analog convertor (ZN428E). The conversion itself is accomplished by software in the computer.

The picture below shows you the complete schematic of the 8-bit digital to analog and analog to digital convertor. In this project the 6522 VIA board is just the interface between the convertor and the computer. The data input lines of the ZN428E chip are connected with Port A of the 6522. Port line A0 is connected with the least significant bit of the data line of the D to A convertor, and port data line A7 is connected with the most significant bit or line of the digital to analog convertor. The 2N428E is enabled using pin PB0 of the 6522 VIA board. When PB0 = 0 all inputs of the digital/analog convertor can accept data from the computer through Port A of the 6522. If pin PB0 goes high (which means PB0 = 1) all inputs are locked immediately and must remain at that state until PB0 becomes 0. The value which was applied last is then stored in the convertor. The output voltage range is set by an operational amplifier (one quarter of a TLO74). The internal reference voltage (VRF) equaling 2.5 volts is used on the ZN428. Figure 3.2 is the block diagram, which shows how to set the output voltage range.

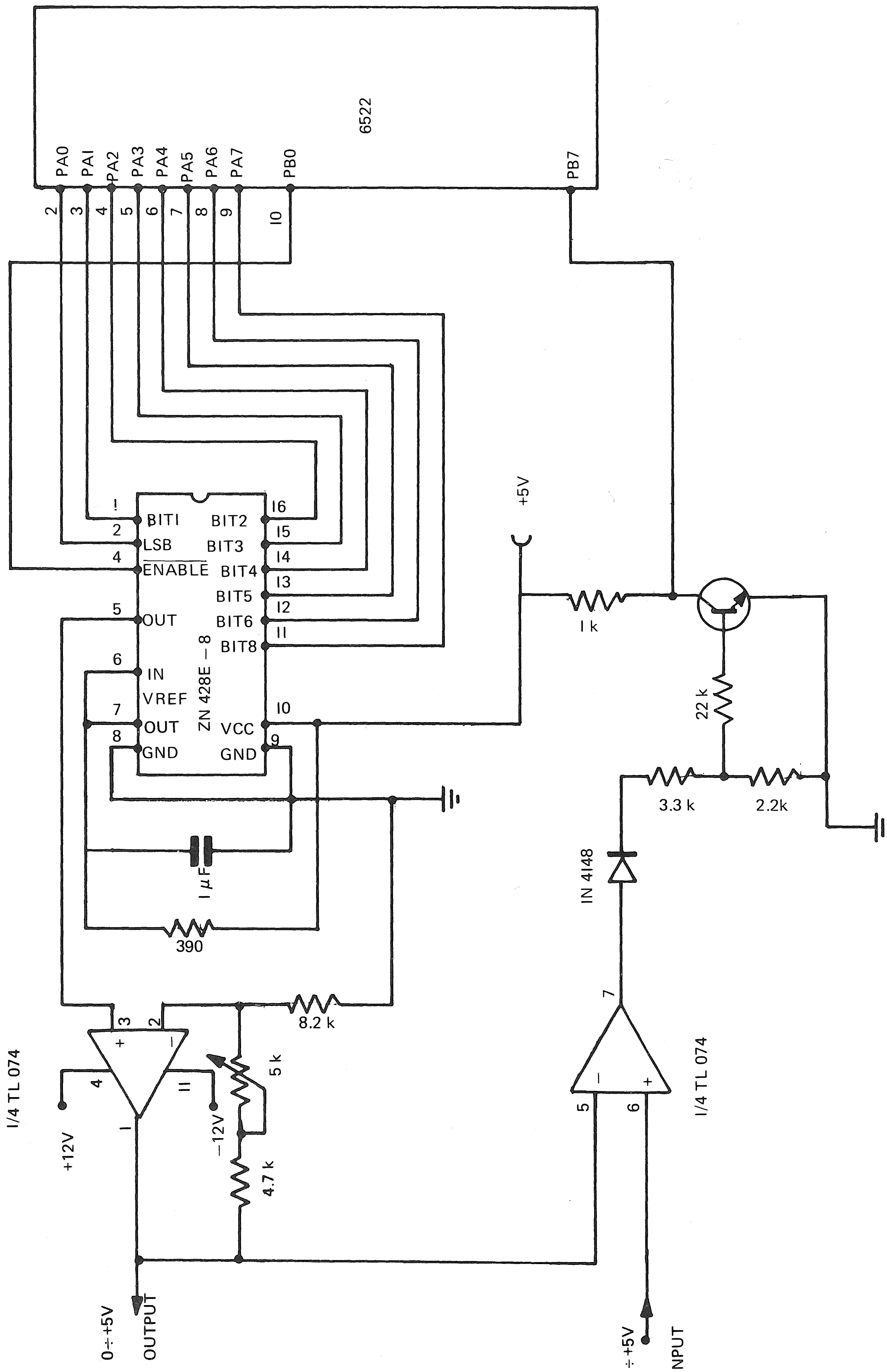


Figure 3.1 8-bit D/A and A/D Convertor Schematic

The schematic in Figure 3.1 shows a circuit that will deliver an output voltage which is variable between 0 and + 5 or 0 and - 5. It cannot be an alternating voltage, and will always be either positive or negative. The formula for calculating this unipolar output voltage (VFS) is:

VFS = (1+R1/R2)*VRF

The range of this voltage, calculated by the above formula, is between 0V and the maximum value, (VFS). Resulting resistance, created by resistors R1 and R2 in parallel, should approximately equal the internal resistance of the converting network. This resistance should be approximately 4000 ohms. For an output voltage range between 0 and + 5 volts and a reference voltage of VRF = 2.5 volts, R1 = R2 = 8000 ohms.

In our schematic R2 = 8200 ohms and R1 is equal to the combination of the 4700 ohm resistor and the 5000 ohm potentiometer in this series. With this configuration the maximum value of the output voltage is a +5 volts. To achieve this you can use the following program:

The ZN 428E is manufactured by Ferranti in the U.K.

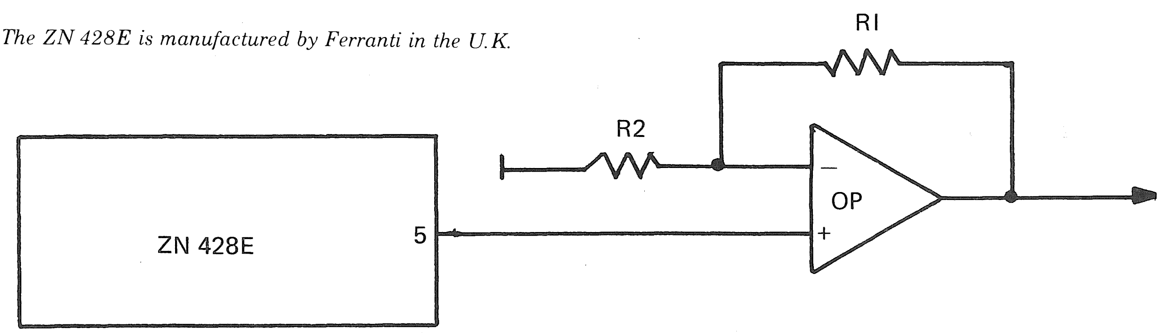


Figure 3.2 D/A Block Diagram

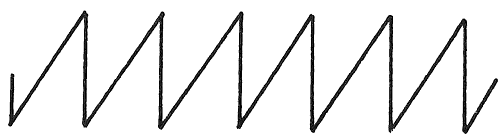
Figure 3.3 Convertor Adjustment

```
10   REM  ******************************
20   REM  *     CONVERTER ADJUST       *
30   REM  ******************************
100   REM  PROGRAMMING THE PORTS
110   REM    PORTA SET TO OUTPUT
120   POKE   - 16189,255
130   REM    PORTB SET TO OUTPUT
140   POKE   - 16190,01
200   REM  OUTPUT OF NUMBERS
210   INPUT " NUMBER=";Z
220   POKE   - 16191,Z
230   PRINT "MORE (Y/N)";: GET W$
240   IF W$ <  > "N" THEN 210
250   END
```

The addresses of Port A and Port B of the 6522 VIA are **C0C1** and **C0C0** when the board is plugged into slot 4 of the Apple. The equivalent decimal addresses are -16192 for Port A and -16191 for Port B. The addresses of the data direction registers DDRB and DDRA are **C0C2** (decimal is -16190) and **C0C3** (decimal is

-16189) respectively. After starting our little program (Figure 3.3) the computer asks us to put in a number. If we type in 255, we set the convertor to its maximum output voltage. Next we use the 5000 ohm potentiometer to adjust the voltage down to +5 volts minus 20 millivolts, which equals 4.98 volts. To make this precise voltage adjustment we recommend using a digital voltmeter. Because +5 volts equals 256, we can only come up to **FF**, which equals 255. Therefore we have to deduct the 20 millivolts from the maximum value. These 20 millivolts correspond exactly to one LSB (least significant bit). If you answer the question 'number' from the program above with an input of zero, the output voltage must be zero. If you want to fool around a little bit, try a few other values like 128 or 64 and so on, and watch the output at pin 8 of the TL074 operational amplifier. With an input of 128, the output voltage should be 2.5 volts.

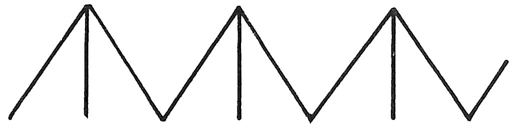
Now we are going to show you the following three programs in 6502 machine-code to demonstrate how your digital/analog convertor works in the Apple II computer:



1. A sawtooth generator

Figure 3.7 Program SAWTOOTH

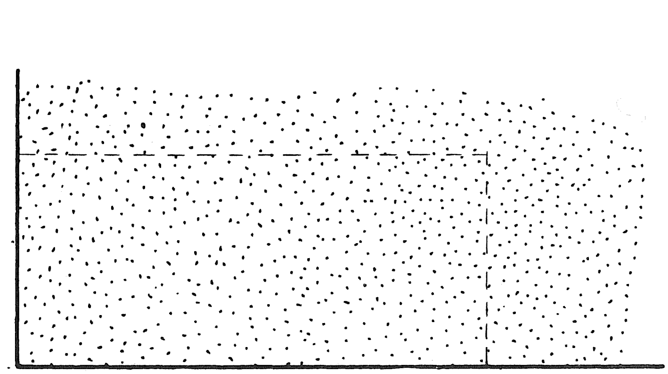
```
0800                    DCM "PR#1"
0800            ;
0800            ;
0800            ;*****************************
0800            ;*                           *
0800            ;*    SAWTOOTH               *
0800            ;*                           *
0800            ;*****************************
0800            ;
0800            ;
0800            DDRA    EQU $C0C3
0800            DDRB    EQU $C0C2
0800            TORA    EQU $C0C1
0800            TORB    EQU $C0C0
0800            ;
0800 A9FF               LDA #$FF
0802 8DC3C0             STA DDRA
0805 A901               LDA #$01
0807 8DC2C0             STA DDRB
080A A200               LDX #$00
080C 8EC1C0     M       STX TORA
080F E8                 INX
0810 18                 CLC
0811 90F9               BCC M
0813            ;
                        END
```



2. A triangle generator

Figure 3.8 Program TRIANGLE

```
0800                    DCM "PR#1"
0800           ;
0800           ;
0800           ;*****************************
0800           ;*                           *
0800           ;*       TRIANGLE            *
0800           ;*                           *
0800           ;*****************************
0800           ;
0800           ;
0800           DDRA     EQU $C0C3
0800           DDRB     EQU $C0C2
0800           TORA     EQU $C0C1
0800           TORB     EQU $C0C0
0800           ;
0800 A9FF               LDA #$FF
0802 8DC3C0             STA DDRA
0805 A901               LDA #$01
0807 8DC2C0             STA DDRB
080A A200               LDX #$00
080C 8EC1C0             STX TORA
080F EEC1C0    M1       INC TORA
0812 D0FB               BNE M1
0814 CEC1C0    M2       DEC TORA
0817 D0FB               BNE M2
0819 F0F4               BEQ M1
081B           ;
                       END
```



3. A binary noise generator

Figure 3.9 Program BINARY NOISE

```
0800                      DCM "PR#1"
0800             ;
0800             ;
0800             ;****************************
0800             ;*                          *
0800             ;*   BINARY NOISE           *
0800             ;*                          *
0800             ;****************************
0800             ;
0800             ;
0800             DDRA     EQU $C0C3
0800             DDRB     EQU $C0C2
0800             TORA     EQU $C0C1
0800             TORB     EQU $C0C0
0800             ;
0800             ZAHL     EPZ $10
0800 A9FF                 LDA #$FF
0802 8DC3C0               STA DDRA
0805 A901                 LDA #$01
0807 8DC2C0               STA DDRB
080A 201308      M        JSR RANDO
080D 8DC1C0               STA TORA
0810 18                   CLC
0811 90F7                 BCC M
0813             ;
0813 38          RANDO    SEC
0814 8511                 STA ZAHL+1
0816 6514                 ADC ZAHL+4
0818 6515                 ADC ZAHL+5
081A 8510                 STA ZAHL
081C A204                 LDX #$04
081E B510        Z1       LDA ZAHL,X
0820 9511                 STA ZAHL+1,X
0822 CA                   DEX
0823 10F9                 BPL Z1
0825 60                   RTS
0826             ;
                          END
```

The following is a description of the listings of the above three programs:

The sawtooth (Figure 3.7) is generated by incrementing the X register and storing the contents of that register in Port A of the 6522. The program starts by setting Port A and PB0 of Port B as outputs. This is done by loading the accumulator with **FF** and storing this to DDRA, and loading the accumulator with a one and sending it to DDRB.

The triangle generator program (Figure 3.8) starts the same way as the previous program, setting the Ports A and B to the same values. Then a zero is stored in Port A. The triangle is generated by incrementing the contents of Port A until it is zero. Then the port will be decremented until it again reaches zero. This loop is repeated indefinitely.

The binary noise program (Figure 3.9) uses a subroutine called RANDO to generate a random number between 0 and 255. The program uses the memory locations defined by the labels ZAHL to ZAHL + 5 to shift and add certain numbers. These numbers are transferred to the 6522 and then to Port A, which is connected to the digital to analog convertor.

You can easily generate other wave-form shapes when you set up your own tables. You can store the exact sequence of each value as numbers in a table in your Apple II computer. If you then pull these values out of the table, perhaps using a time delay, you can even generate very complex functions on your computer.

Until now we have only discussed the ZN428E digital to analog covertor in a digital to analog application. This powerful chip also allows you to construct an analog to digital convertor using special software within the Apple II. Digital computers operate with fixed voltages and can only recognize the binary digits, one and zero (low, high). Most of the signals around us are analog. If you think of such things as the temperature, pressure, light, sound intensity, and every signal which comes out of a transducer, these signals are voltages or currents in analog form. To feed that analog information into a computer, you have to convert the voltage level into digital information.

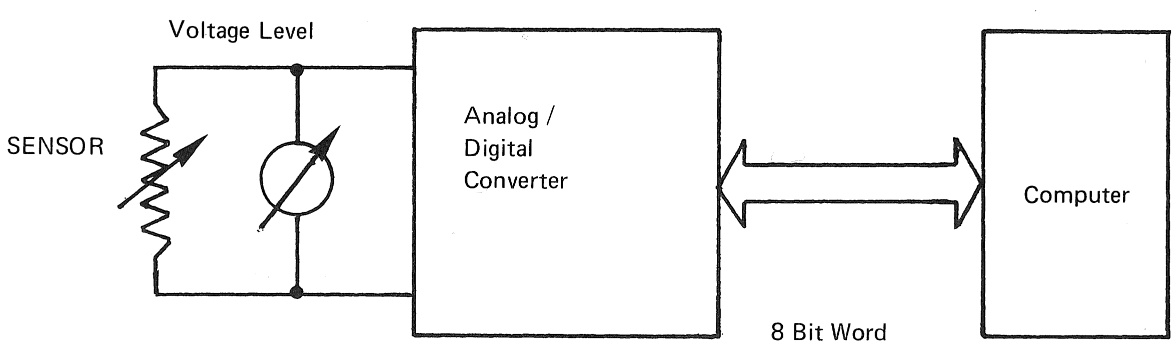


Figure 3.10 Block Diagram of the A/D Convertor

There are several ways to convert an unknown voltage to a digital number. First there are integrating ADC's. These convertors use an analog integrator and a comparator. When the switch (S) in Fig. 3.11 is closed by a pulse, the integrator starts a ramp function.

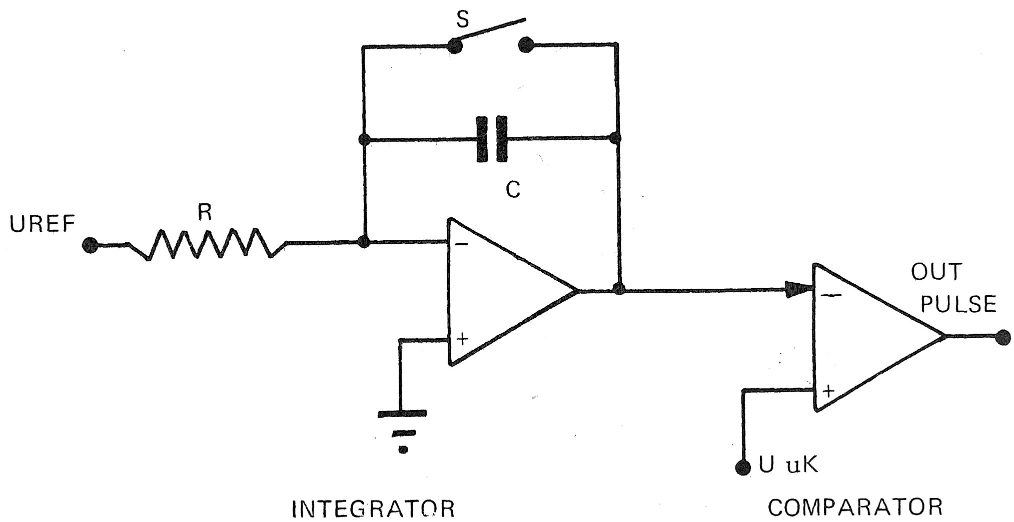


Figure 3.11 Schematic of the Integrator/Comparator

This voltage is compared with the unknown voltage, designated by U uK. When the ramp function voltage is equal to this voltage, the comparator switches from zero to one. The time between the start pulse and the switching of the comparator is measured with a digital counter.

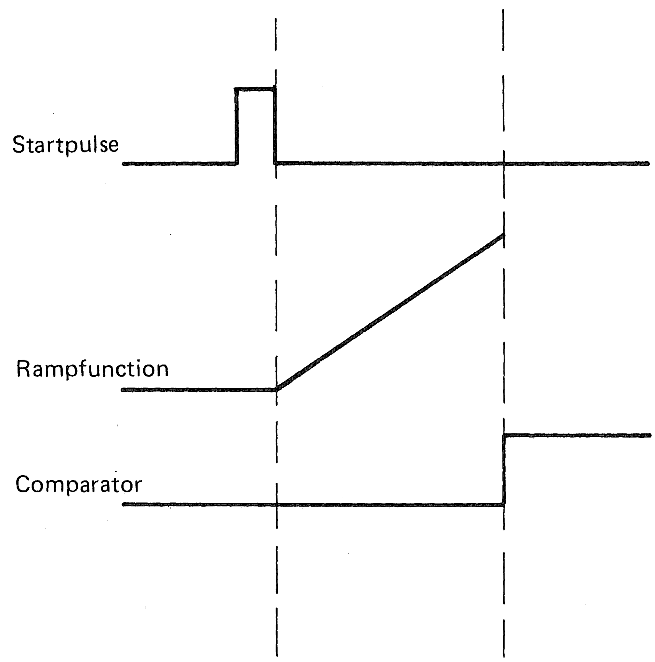


Figure 3.12 Digital Conversion with a RAMP Function

This basic circuit is used in several ways, such as a single slope, dual slope, or triple slope convertor. Another way to convert a voltage to a number uses a digital ramp function.

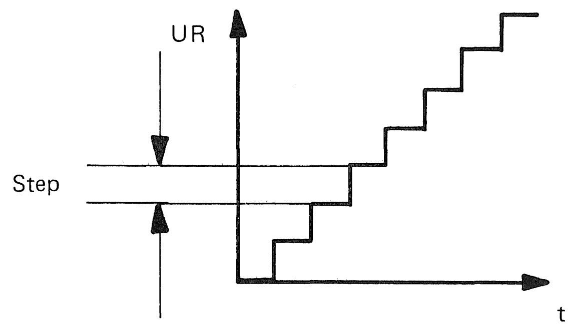


Figure 3.13 RAMP Function Waveform

This ramp function is compared with the unknown voltage. When they are equal, counting stops and the number of steps is equal to the unknown voltage. This is a very slow conversion. A third method is the successive approximation method, which we use in our application. Details are discussed later.

Today there are very cheap analog to digital convertors on the market. With a few resistors and a 555 timer circuit, you can even build one for less than five dollars. These convertors are not very precise and are used mostly for joysticks, paddles, and low quality temperature measurement and control applications. If somebody talks about analog to digital convertors, you always hear words like resolution, accuracy, linearity, settling time and clock rate. We will discuss the more important specifications here to give you a feeling of what an analog to digital convertor can do and what it cannot do.

Resolution

Resolution describes the amount of input voltage change that is required to increment the output of an A to D covertor between one code change and the next code change. A convertor with N switches can resolve one part in two to the Nth parts.

The input signal is simulated approximately by a series of digital steps. Resolution may be expressed in full scale or in binary bits. For example: an ADC with 12-bit resolution could resolve one part in two to the twelfth, which means one part out of 4096 (or 1/4096) equals 0.0245% of the full scale. A convertor with ten volts full scale could resolve a 2.45 millivolt input change. If you now compare this with an 8-bit ADC, you will only have one part out of 256 (1/256), which equals 0.3906%. On a ten-volt full scale this gives you a resolution of 39 millivolts. Resolution is a design parameter rather than a performance specification. It says nothing about accuracy or linearity.

Accuracy

Accuracy describes the difference between the actual input voltage and the full scale weighted equivalent of the binary output code. Included are quantizing errors and all other errors. A twelve-bit ADC is stated to be plus or minus one LSB accurate. This is equivalent to 0.0245%, or twice the minimum possible quantizing error of 0.0122%.

Quantizing Error

Quantizing error is the maximum deviation from a straight linear transfer function on a perfect ADC, as you will note in Figure 3.14

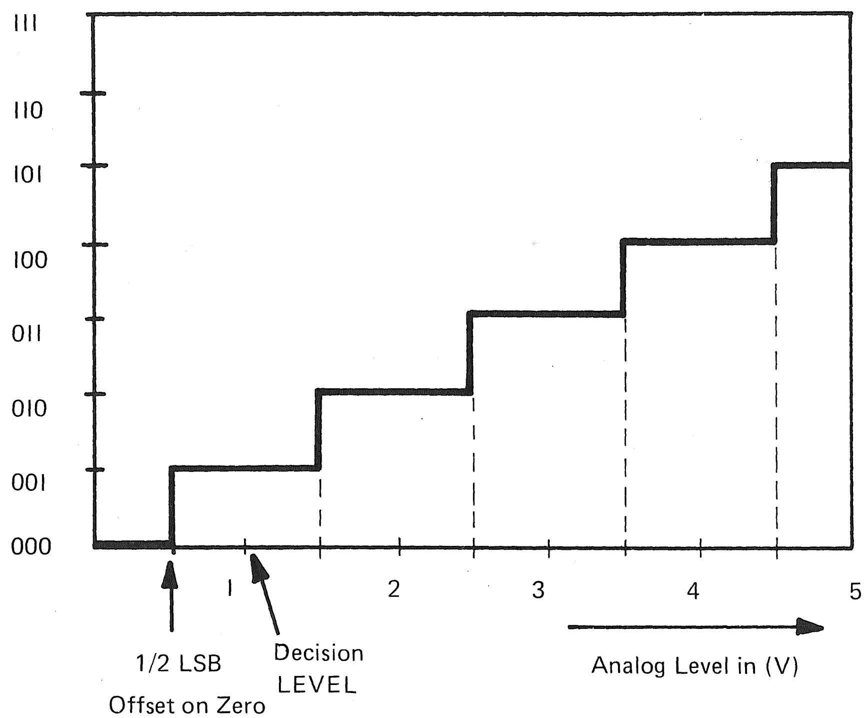


Figure 3.14 Quantized Input Signal

The ADC quantizes the analog input into a finite number of output codes.

Conversion/Clock Rates

Conversion rate is the speed at which the ADC can make repetitive data conversions. It is affected by propagation delay in counting events, ladder switches and comparators. The conversion rate is specified as the number of conversions per second or as the number of microseconds to complete one conversion (including the effects of settling time). The clock rate is the minimum or maximum pulse rate at which ADC counters may be driven.

The 8-Bit D/À and A/D Convertor, Part Two

For the analog to digital conversion we use a digital to analog convertor. Therefore, it must be supplemented by software in the computer itself. This program uses a technique called successive approximation. The unknown input voltage of the ZN428E is compared with one-half of the full range voltage. This voltage, in our case, is a positive 5 volts. If the input voltage is now higher than one-half of the full range, the computer starts another comparison with three quarters of the full range of the output voltage. If the input voltage is lower, a comparison with one quarter of the full range voltage will be performed. At the next comparison, the remaining interval is divided again by two and in this way the unknown voltage is approximated.

After eight comparisons, the conversion is finished. The input voltage is now recognized with a precision of ±20 millivolts.

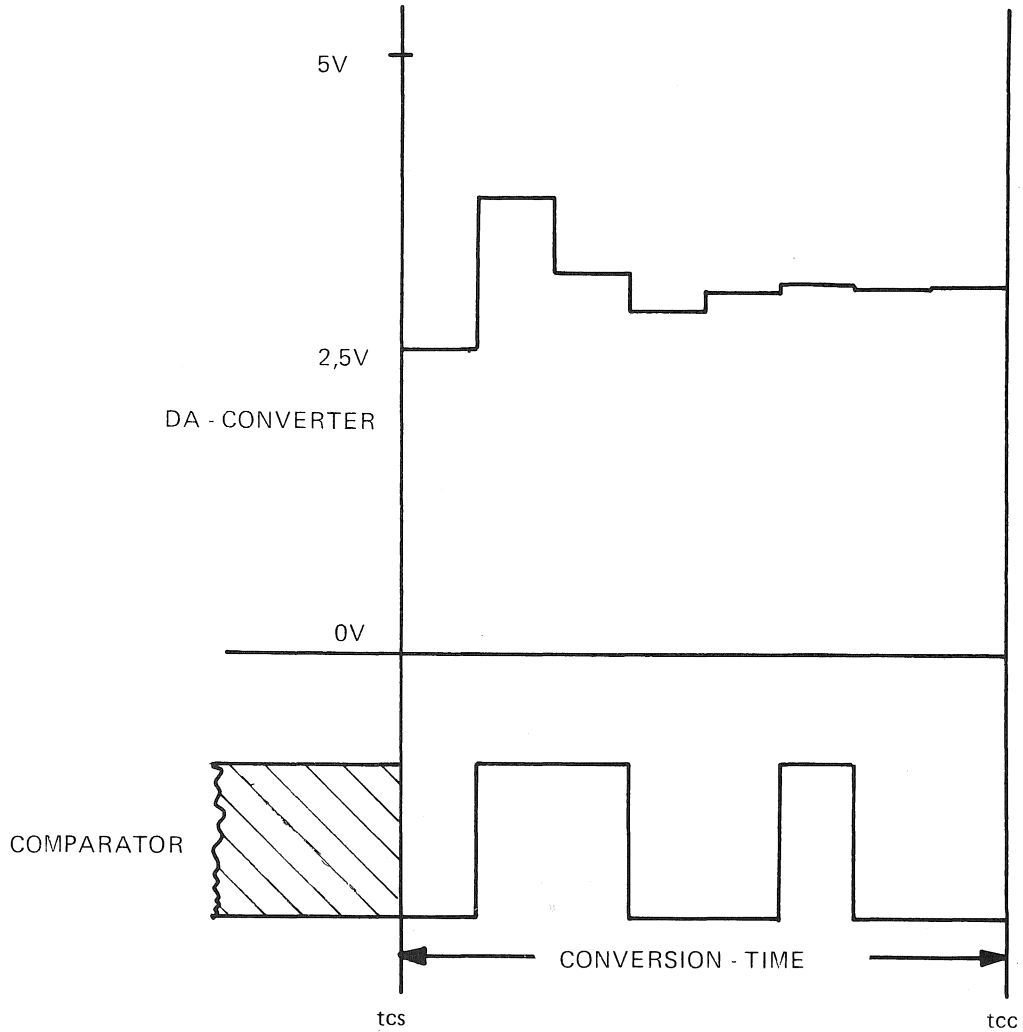


Figure 3.15 A/D Conversion by Successive Approximation

In Fig. 3.15 you can see the sequence of an analog to digital conversion utilizing a digital/analog convertor and a comparator. In the upper half is the output of the digital/analog convertor; in the lower-half the output of the comparator is shown. The conversion starts at time tcs. The state of the comparator prior to this time is undetermined. The input voltage is compared with 2.5 Volts and with a low level output from the comparator before the input voltage is accepted. First the input voltage is compared with 2.5V plus 1.25V (=3.75V). The comparator responds with a one, to show that this voltage is higher than the input voltage; so this voltage is not accepted. The second comparison is made with 2.5V plus 0.625V (= 3.125V). This voltage won't be accepted either, and the output of the comparator will be one. The next comparison voltage is then 2.5V plus 0.3125V (= 2.8125 Volts). The comparator accepts this voltage, responding with a zero at the output. In the computer, the acceptance of a voltage level is marked with a one. Up to this point, the four highest bits of the digital number are 1001. The conversion continues: accepting the next voltage level, refusing the next one, and accepting the two next ones. The whole digital number finally becomes 10011011 = **9B**. This corresponds to a voltage of 3.099 Volts. Because of the quantization error, the level of the input voltage lies somewhere between 3.099 ±20 millivolts. The conversion is completed at tcc.

If you want to measure a ten-volt input voltage, you have to use a voltage divider circuit, and your error will be doubled, (±40 millivolts). The output signal of comparator C2 (see Figure 3.1) will be a positive to negative 12 volts. To connect that output to the PB7 input of the 6522 chip we have to convert that level into a TTL compatible level. The program you need to perform the analog to digital conversion will be found below.

Figure 3.16 Successive Approximation Program

```
0800                       DCM "PR#1"
0800                ;
0800                ;***************************
0800                ;*                         *
0800                ;* ANALOG-DIGITAL-CONVER-  *
0800                ;* SION BY SUCCESSIVE      *
0800                ;* APPROXIMATION WITH A    *
0800                ;* 8-BIT DA-CONVERTER      *
0800                ;*                         *
0800                ;***************************
0800                ;
0800                DDRA    EQU $C0C3
0800                DDRB    EQU $C0C2
0800                TORA    EQU $C0C1
0800                TORB    EQU $C0C0
0800                VALUE   EQU $C4FF
0800                Z       EPZ $10
0800                PRTBYT  EQU $FDDA
0800                ;
0800 2000C4                 JSR INIT
0803 200BC4                 JSR CONVER
0806 ADFFC4                 LDA VALUE
0809 20DAFD                 JSR PRTBYT
080C 00                     BRK
C400                        ORG $C400
C400                ;
C400                ;SET THE 6522 PORTS
C400                ;
C400 A901           INIT    LDA #$01
C402 8DC2C0                 STA DDRB
C405 A9FF                   LDA #$FF
C407 8DC3C0                 STA DDRA
C40A 60                     RTS
C40B                ;
C40B                ; CONVERT
C40B                ;
C40B A980           CONVER  LDA #$80
C40D 8510                   STA Z
```

Listing Continued . . .

Continued Listing

```
C40F A97F      39          LDA #$7F
C411 8DC1C0    40   W0     STA TORA
C414 EA        41          NOP
C415 EA        42          NOP
C416           43   ;ONLY NECESSARY BECAUSE
C416           44   ;OF SLOW COMPARATOR
C416 EA        45          NOP
C417 EA        46          NOP
C418 ACC0C0    47          LDY TORB
C41B 1002      48          BPL W1
C41D 0510      49          ORA Z
C41F 4610      50   W1     LSR Z
C421 B004      51          BCS FIN
C423 4510      52          EOR Z
C425 90EA      53          BCC W0
C427 8DFFC4    54   FIN    STA VALUE
C42A 60        55          RTS
C42B           56   ;
C42B           57   ;
```

In lines 29 to 33 the data direction registers are set. The conversion program starts with line 37. We initialize memory location Z by setting bit 7 to logical 1. The first comparison takes place with **7F**. If the input voltage is higher, no BPL will be taken in line 48. Then the OR instruction in line 52 will set the bit-7 of the accumulator to logical 1. After Z is shifted right one bit, it is equal to **40**. By an EOR instruction, bit 6 in the accumulator will be cleared. The contents, which are now **BF**, are stored in Port A. Before you can read out the contents of Port B via the LDY instruction, the convertor must be allowed to settle. The ZN428E is very fast, so after 800 microseconds the new analog input can be read. But, on the other hand, the comparator built with the TL074 is slow. To solve that problem, you must insert four NOP instructions in the program. The conversion is finished when LSR Z brings the marked bit into the carry bit.

Figure 3.17 Plotting Program

```
10  REM  **************************
12  REM    * PLOTTING A CURVE     *
14  REM    * ON THE APPLESCREEN   *
18  REM  **************************
50 D$ =  CHR$ (04)
60  PRINT D$;"BLOAD ADWC400.B"
100 INIT =  - 15360:WA =  - 15349
110 VA =  - 15105
120  CALL INIT
200  HGR : COLOR= 15
210 X = 0
220  CALL WA
230 W =  PEEK (VA)
240 P = 160 - W / 2
250  HPLOT X,P
260 X = X + 1
270  IF X < 280 THEN 220
280  END
```

The BASIC program in Figure 3.17 brings the converted voltage values onto the Apple screen. Since there are 255 different voltages, but only 160 pixels available for us to use in a vertical direction on the screen, we will divide each voltage value by two before displaying it. This means that we will be using only 127 pixel range to display all voltage values. The zero point of the graph is located 160 pixels down from the top of the screen. After each measurement, the X value will be incremented by one. If you want to reduce the measuring rate, you can insert a delay loop before line 270.

Figure 3.18 ADW C400.B Program

```
C400-    A9 01         LDA    #$01
C402-    8D C2 C0      STA    $C0C2
C405-    A9 FF         LDA    #$FF
C407-    8D C3 C0      STA    $C0C3
C40A-    60            RTS
C40B-    A9 80         LDA    #$80
C40D-    85 10         STA    $10
C40F-    A9 7F         LDA    #$7F
C411-    8D C1 C0      STA    $C0C1
C414-    20 2A C4      JSR    $C42A
C417-    AC C0 C0      LDY    $C0C0
C41A-    10 02         BPL    $C41E
C41C-    05 10         ORA    $10
C41E-    46 10         LSR    $10
C420-    B0 04         BCS    $C426
C422-    45 10         EOR    $10
C424-    90 EB         BCC    $C411
C426-    8D FF C4      STA    $C4FF
C429-    60            RTS
C42A-    A2 10         LDX    #$10
C42C-    CA            DEX
C42D-    D0 FD         BNE    $C42C
C42F-    60            RTS

 C400.C42F

C400- A9 01 8D C  C0 A9 FF 8D
C408- C3 C0 60 A9 80 85 10 A9
C410- 7F 8D C1 C0 20 2A C4 AC
C418- C0 C0 10 02 05 10 46 10
C420- B0 04 45 10 90 EB 8D FF
C428- C4 60 A2 10 CA D0 FD 60
*
```

The conversion program ADWC400.B (see Figure 3.18) is put into a little on-board RAM on the 6522 I/O board. It is safe and protected against any collision with the BASIC program there. If you have plugged the 6522 VIA card into slot 4 of your Apple, the starting address of the program in the RAM area is **C400**. The subroutine INIT sets the data directional registers. WA is the conversion program. The converted value will be stored in the memory location **C4FF**, which equals decimal -15105 (see listing in Figure 3.17). From this location the value will be transferred to the BASIC program. Between the instructions **STA $C0C1** and **LDY $C0C0** in program ADW C400-B, a time delay is inserted to give the comparator time to settle. If you use a faster comparator, like an LM393, for the voltage comparison, you can eliminate this subroutine. Then after execution of the instruction **STA $C0C1**, you can get the result of the comparison immediately. If you use the circuit shown in Figure 3.19, then you must change the jump instruction in memory location **C41A** into a **BMI $C41E** instruction. The conversion time is then approximately 220 microseconds.

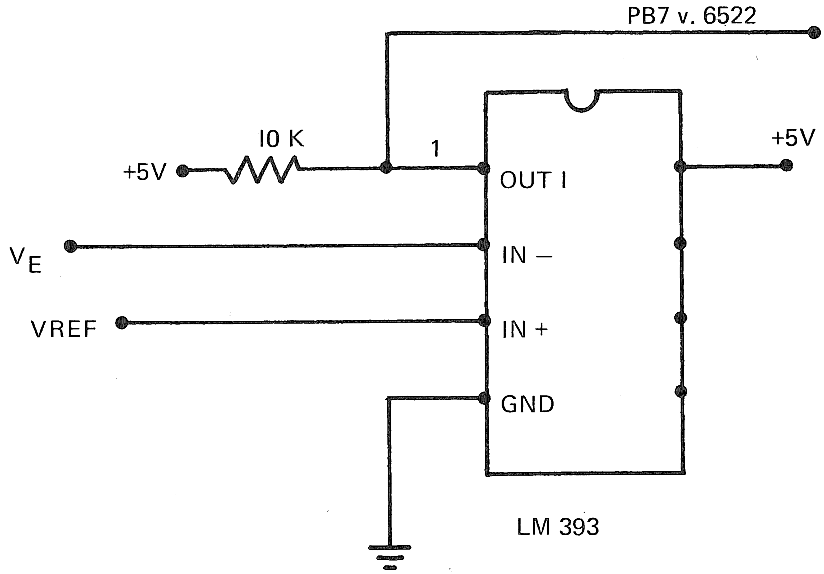


Figure 3.19 Block Diagram of the LM393

For very precise analog to digital conversion, changes in the input voltage should not exceed half the amount of the least significant bit during the conversion time. In our case, this means that there must be no change of more than 10 millivolts during the conversion time. From this we can calculate the fastest allowed voltage change as 45.5 volts per second. With a signal amplitude of 2.5 volts, we only obtain an upper frequency limit of 3 cycles per second. We can only measure rather slow events.

Using Two D/A Convertors

In many applications it is very useful to have two digital to analog convertors available at your computer. These applications may include plotting the results of a calculation on an X/Y plotter or an X/Y storage oscilloscope. Instead of looking at columns of numbers, you simply look at a picture and see what happens. Or, you may generate very complex wave forms for the control of several motors and robotics. This is illustrated in the following application in which the DC motor is driven by two amplifiers, A1 and A2. The input voltage of these amplifiers is provided by the digital/analog convertors, DAC1 and DAC2 (See Figure 3.20).

Then, for example, you can generate the following function of speed versus time.

This system could be easily expanded to a digital control system. With an analog/digital convertor you can measure the intensity of light, temperature, pressure and so on. A computer calculates the necessary reaction of the system and

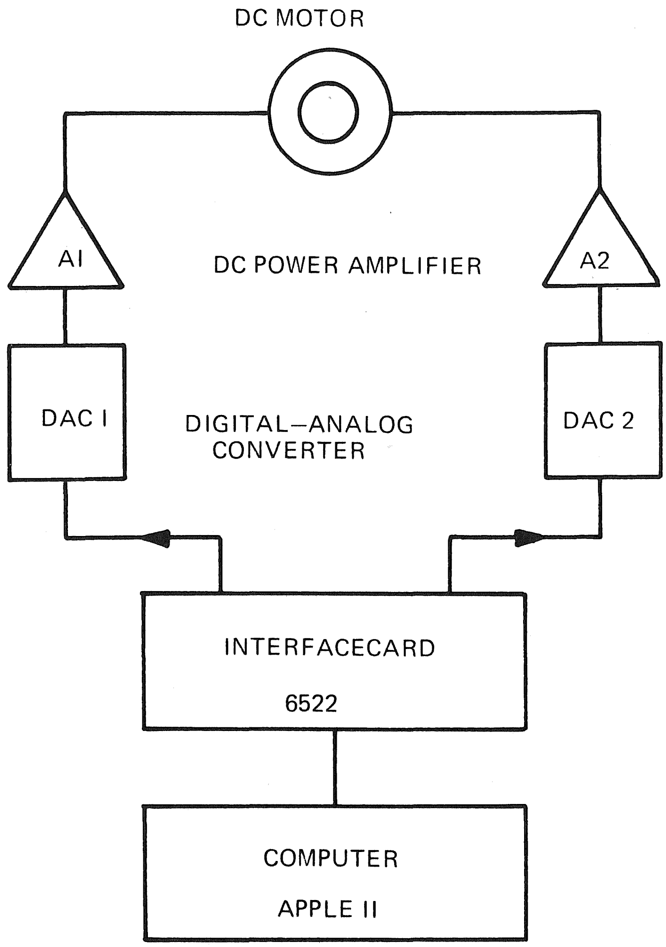


Figure 3.20 DC Motor Control

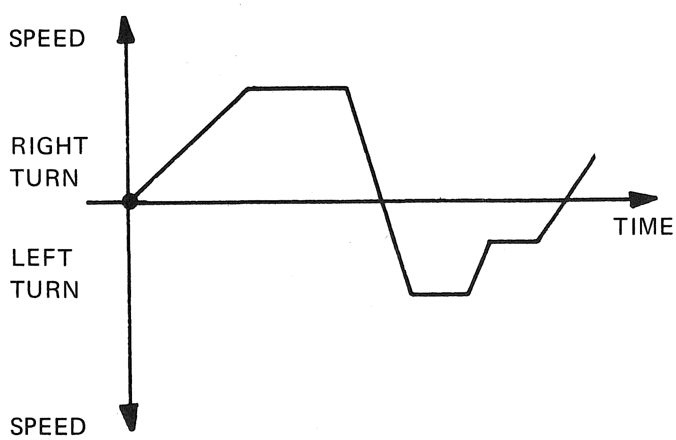


Figure 3.21 Speed/Time Function

then responds as described by the circuit above. For this application we use two ZN425E digital analog convertors which are mounted on the prototype area of our 6522 I/O board.

The data lines of U2 in Figure 3.22 are connected to Port A, and the data lines of U4 are connected to Port B of the 6522. The two operational amplifiers, U3 and

U5, measure the difference between the output voltage (Vout) and the reference voltage (Vref) from the ZN425E. The output voltage swing at pin 6 ranges from a +2.75 volts to -2.75 volts. The +2.75 volts is equal to an input of **FF**. From the keyboard the -2.75 volts is equal to **00**. An output voltage of 0 volts is achieved by **80** (or 128 decimal). In this demonstration we will consider three programs: one in BASIC and two in machine language. In the BASIC program (Figure 3.23) we will calculate a circle and use the two DAC's for plotting the values on the screen of an oscilloscope.

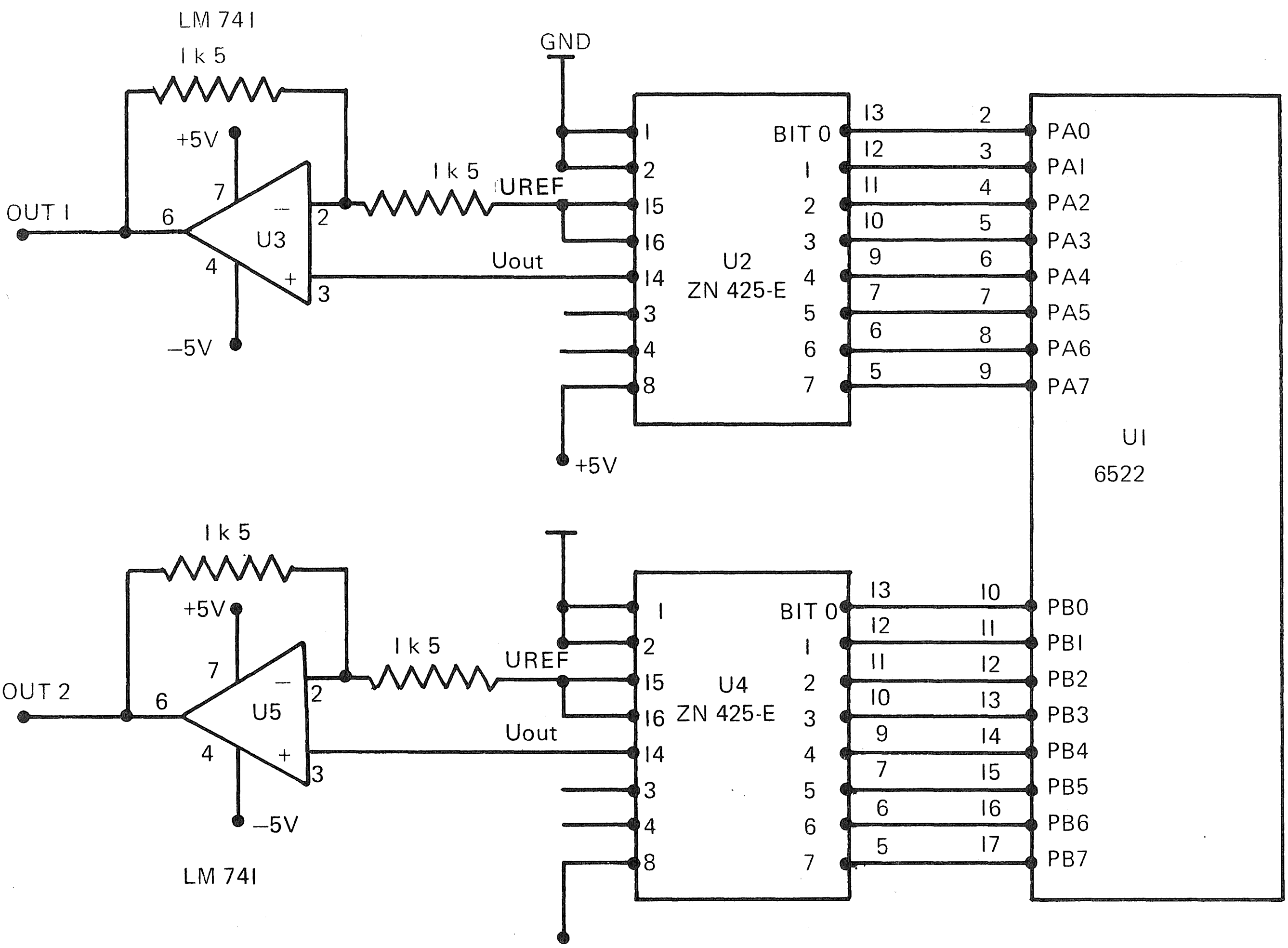


Figure 3.22 Connecting DAC ZN 425E's to the 6522

In lines 10 and 20 we set the data direction registers, the value of TA to the address of Port A, and the value of TB to the address of Port B. In the succeding lines we calculate the values of a circle, whose center is at X = 128 and Y = 128. This is the zero volt point for both the X and the Y coordinates. In lines 130 and 135 the calculated values for X and Y are POKEd into Ports A and B. The output voltage of Port B is connected to the X input of the oscilloscope, and the output voltage of Port A is connected to the Y input. When you look at the screen, you will see the beam wandering slowly around with a slight flickering in the two axes. This is due to the time delay between the two POKE instructions. You will notice that BASIC is not very fast. However, if you use an X/Y plotter instead of an oscilloscope, this would be the correct speed for plotting the values. Several changes can be made in the program by changing line 110 to FOR T = 0 TO 360: STEP 2. This would cause you to get much closer steps and a rounder circle. If you change line 120 to read X = SIN (2*T*F), you can create a Lissajous (or 'figure-eight') figure with a frequency ratio of 2 to 1.

Figure 3.23 Program CIRCLE

```
1   REM  *************************
2   REM  * PLOTTING A CIRCLE     *
3   REM  *************************
10  POKE  - 16190,255: POKE  - 16189,255
20 TA =  - 16191:TB =  - 16192
100 PI = 3.14159
105 F = 2 * PI / 360
110  FOR T = 0 TO 360 STEP 5
120 X =  SIN (T * F)
122 X = X * 127 + 128
125 Y =  COS (T * F)
127 Y = Y * 127 + 128
130  POKE TB,X
135  POKE TA,Y
140  NEXT : GOTO 100
```

Now let's take a look at the two machine-language programs. These programs run much faster. With the first one, we will plot a square on the screen of the oscilloscope.

Figure 3.24 Plotting a Square

```
0800                         DCM "PR#1"
0800                  ;
0800                  ;
0800                  ;*****************************
0800                  ;*                           *
0800                  ;*       SQUARE              *
0800                  ;*                           *
0800                  ;*****************************
0800                  ;
0800                  ;
0800                  DDRA   EQU $C0C3
0800                  DDRB   EQU $C0C2
0800                  TORA   EQU $C0C1
0800                  TORB   EQU $C0C0
0800                  ;
0800 200608                  JSR INIT
0803 4C0F08                  JMP SQUARE
0806                  ;
0806 A9FF             INIT   LDA #$FF
0808 8DC3C0                  STA DDRA
080B 8DC2C0                  STA DDRB
080E 60                      RTS
080F                  ;
080F                  ;
080F A000             SQUARE LDY #$00
0811 A200                    LDX #$00
0813 8EC1C0                  STX TORA
0816 8CC0C0                  STY TORB
0819 E8               S1     INX
081A 8EC1C0                  STX TORA
```

Listing Continued . . .

Continued Listing

```
081D E0FF                  CPX #$FF
081F D0F8                  BNE S1
0821 C8         S2         INY
0822 8CC0C0                STY TORB
0825 C0FF                  CPY #$FF
0827 D0F8                  BNE S2
0829 CA         S3         DEX
082A 8EC1C0                STX TORA
082D D0FA                  BNE S3
082F 88         S4         DEY
0830 8CC0C0                STY TORB
0833 D0FA                  BNE S4
0835 F0D8                  BEQ SQUARE
0837            ;
                           END
```

The program in Figure 3.24 above starts by initializing the data direction registers and the subroutine in the program. Then it begins to draw a square starting in the lower left hand corner of the screen. By incrementing the X register and storing that value to Port A, one side of the square will be drawn. After reaching **FF** the Y register begins to increment, and storing that value to Port B will cause the right side of the square to be plotted. The remaining sides of the square are drawn by decrementing first the X register and then the Y register, while storing those values in the appropriate Ports A and B. When you look at the screen you will see that the machine-language instructions are a lot faster than BASIC. You won't see the beam wandering around; you will see a very distinct fully-drawn square.

Figure 3.25 Program RANDOM WALK

```
0800                       DCM "PR#1"
0800            ;
0800            ;
0800            ;*****************************
0800            ;*                           *
0800            ;*   RANDOM WALK             *
0800            ;*                           *
0800            ;*****************************
0800            ;
0800            ;
0800            DDRA       EQU $C0C3
0800            DDRB       EQU $C0C2
0800            TORA       EQU $C0C1
0800            TORB       EQU $C0C0
0800            ;
0800            ZAHL       EPZ $10
0800            ;
0800 A9FF                  LDA #$FF
0802 8DC3C0                STA DDRA
0805 8DC2C0                STA DDRB
0808 201708     M          JSR RANDO
080B 8DC1C0                STA TORA
080E 201708                JSR RANDO
```

Listing Continued . . .

Continued Listing

```
0811 8DC0C0              STA TORB
0814 18                  CLC
0815 90F1                BCC M
0817             ;
0817 38          RANDO   SEC
0818 8511                STA ZAHL+1
081A 6514                ADC ZAHL+4
081C 6515                ADC ZAHL+5
081E 8510                STA ZAHL
0820 A204                LDX #$04
0822 B510        Z1      LDA ZAHL,X
0824 9511                STA ZAHL+1,X
0826 CA                  DEX
0827 10F9                BPL Z1
0829 60                  RTS
082A             ;
                       END
```

In the third program, 'RANDOM WALK' (Figure 3.25) we use the previously described subroutine RANDO for generating random numbers. These numbers are stored, one after the other, to Port A and Port B. When you look at the screen, you will see many points arranged in a square, moving right, as in Brownian molecular movement. The examples we have just discussed are only a few of the things that are possible when you have two digital to analog convertors connected to a computer.

A/D Conversion with the ADC1210

The following description is an industrial application that was actually done with a 12-bit analog to digital convertor. It was used for measuring a slowly varying voltage, once per second, with great accuracy. As we mentioned earlier, using a 12-bit ADC gives us much better resolution and accuracy than an 8-bit convertor.

The complete schematic is shown in Figure 3.26 below. The outputs of the ADC1210 are tied to the ports of the 6522. The least significant bits are connected to Port A, and the remaining 4 most significant bits to PB0 through PB3 of Port B. On PB4 the start convert pulse ($\overline{SC}$) is generated, while PB5 reads the conversion complete signal ($\overline{CC}$). The analog/digital conversion inside the 1210 is done in the same manner as we have done it with the eight-bit ADC and software. The ADC1210 also converts the analog signal to a digital number by successive approximation, but in this case it is being done by the hardware. It uses an external clock, whose frequency must fall between 60 and 70 kilohertz. Therefore we divide the one megahertz machine clock by four stages of the frequency divider circuit 4024. The input frequency for the 1210 is then 67.5 kilohertz. The output level of pins 1 through 12 of the 1210 is V+ for a logical zero and zero for a logical 1. At the input pins of the 6522 the voltage levels must not exceed the TTL voltage levels. Therefore the supply voltage, V+, which is internally equal to the reference voltage is set to +5.12 volts. This value is derived when the +12 volts power supply of the Apple is a voltage regulator, such as UA78G, or any other adjustable voltage regulator. The exact voltage is adjusted by the 5000 ohm potentiometer (P1). With

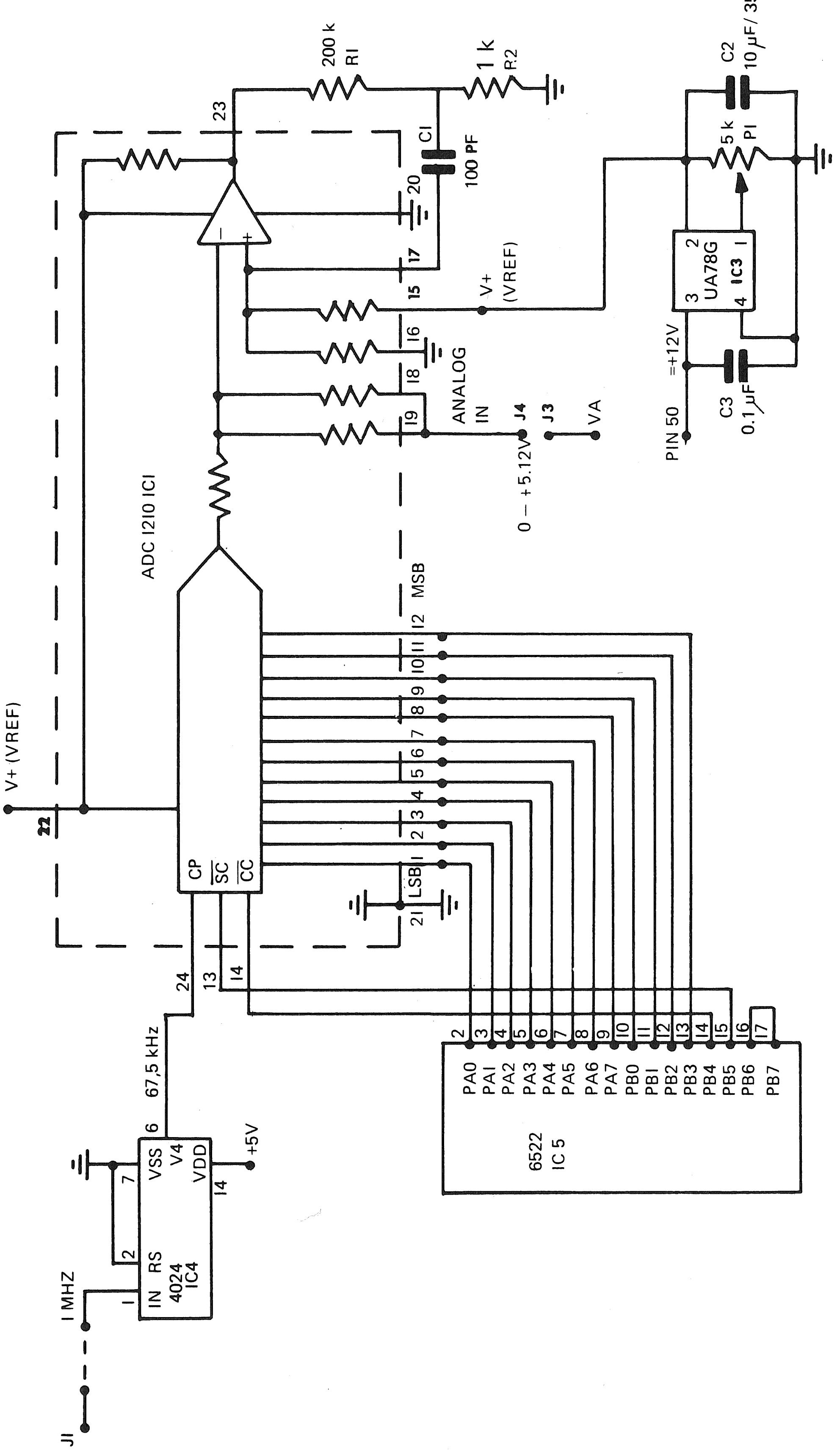


Figure 3.26 Schematic of the ADC 1210

the configuration shown, we have a unipolar input voltage swing from zero to +5.10 volts at pin 19 of the ADC. In most cases the sensors will not directly supply this input voltage. For amplifying low voltages you can use an on-board quad op-amp 4136 configured as an instrumentational amplifier.

The ADC 1210 is a product of National Semiconductor.

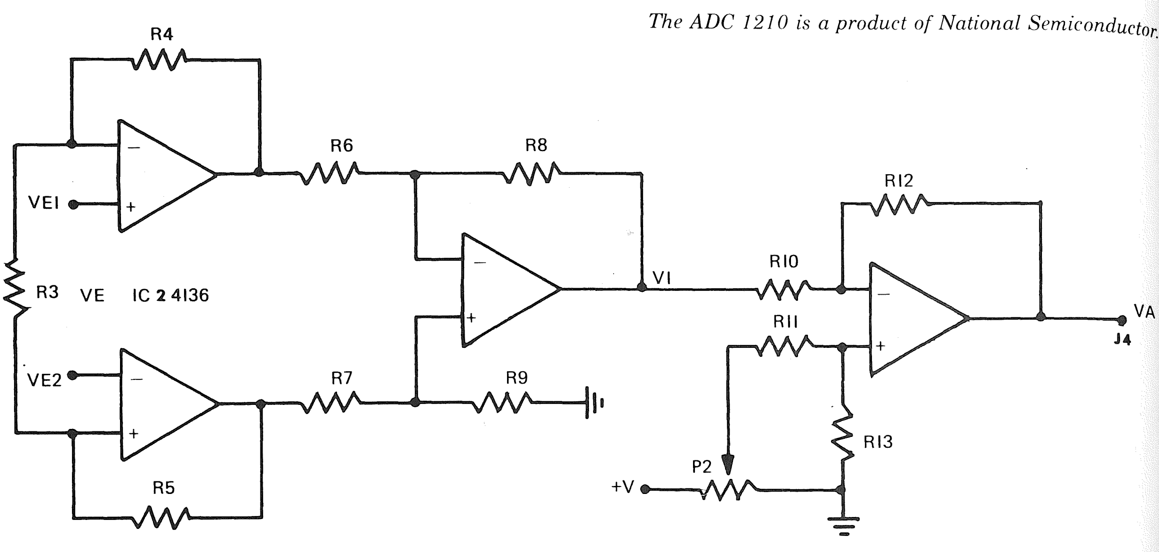


Figure 3.27 Instrumentation Amplifier Schematic

If you make R4 equal to R5, and R6 equal to R7, R8, and R9, the gain factor is: V = 1+2*R4/R3. The recommended values of R4 and R6 are 100,000 ohms. As we have a differential input, the voltage V1 is: V1 = (1+2*R4/R3)*(VE2-VE1). If you choose 100,000 ohms for R4 and 2000 ohms for R3, you will have a voltage gain of V equals 100. The first stage of the amplifier is also a differential amplifier. With R10 = R11 = R12 and R13 equal to 100,000 ohms, the voltage gain is one. The potentiometer (P2) is used to adjust the output voltage level (VA) to the input range of the ADC.

Finally, we will look at pins PB6 and PB7 of the VIA 6522, which are tied together. On PB7 we will create a square-wave signal with a period of 0.1 second from timer one. Timer two acts as a counter. It is set to 10 by the program and decremented every tenth of a second. When it reaches zero it is time to take a new measurement.

Now we will take a look at the program. This is divided into a BASIC program part (Figure 3.28) and a machine-language part (Figure 3.29).

Figure 3.28 BASIC ADC Input Program

```
10   REM  *************************
12   REM  * ANALOG INPUT WITH THE *
14   REM  * ADC 1210 .            *
16   REM  * RAGE 0 - 5.12 VOLTS   *
18   REM  * 1 MEASUREMENT/SECOND  *
20   REM  * STARTING WITH BUTTON 1*
30   REM  *************************
100 MSB =  - 15105:LSB =  - 15106
```

Listing Continued . . .

Continued Listing

```
105 FIN =  - 15107
106  DIM MW(500)
107 I = 0
110 INIT =  - 15360:START =  - 15248:MEASURE =  - 15223:OFF =  - 15200
112  CALL INIT
115  PRINT "START MEASUREMENT BY KEYPRESS"
120  CALL START: GOSUB 1000:
130  CALL MEASURE: GOSUB 1000
140  IF I > 3 THEN  CALL OFF: GOSUB 1500
145  PRINT A
150  GOTO 130
160  END
1000 A =  PEEK (MSB) * 256 +  PEEK (LSB):
1010 A = A * 1.2942E - 03
1020 MW(I) = A:I = I + 1
1030  RETURN
1500  IF  PEEK (FIN) > 127 THEN  RETURN
1505  HGR : HCOLOR= 3
1506  HPLOT 1,159 TO 250,159
1507 Y = 25
1510  FOR K = 0 TO I - 1
1520  HPLOT K,159 - Y * MW(K)
1530  PRINT K;"      ";A
1540  NEXT K
1550  END
```

Figure 3.29 Machine-language Version

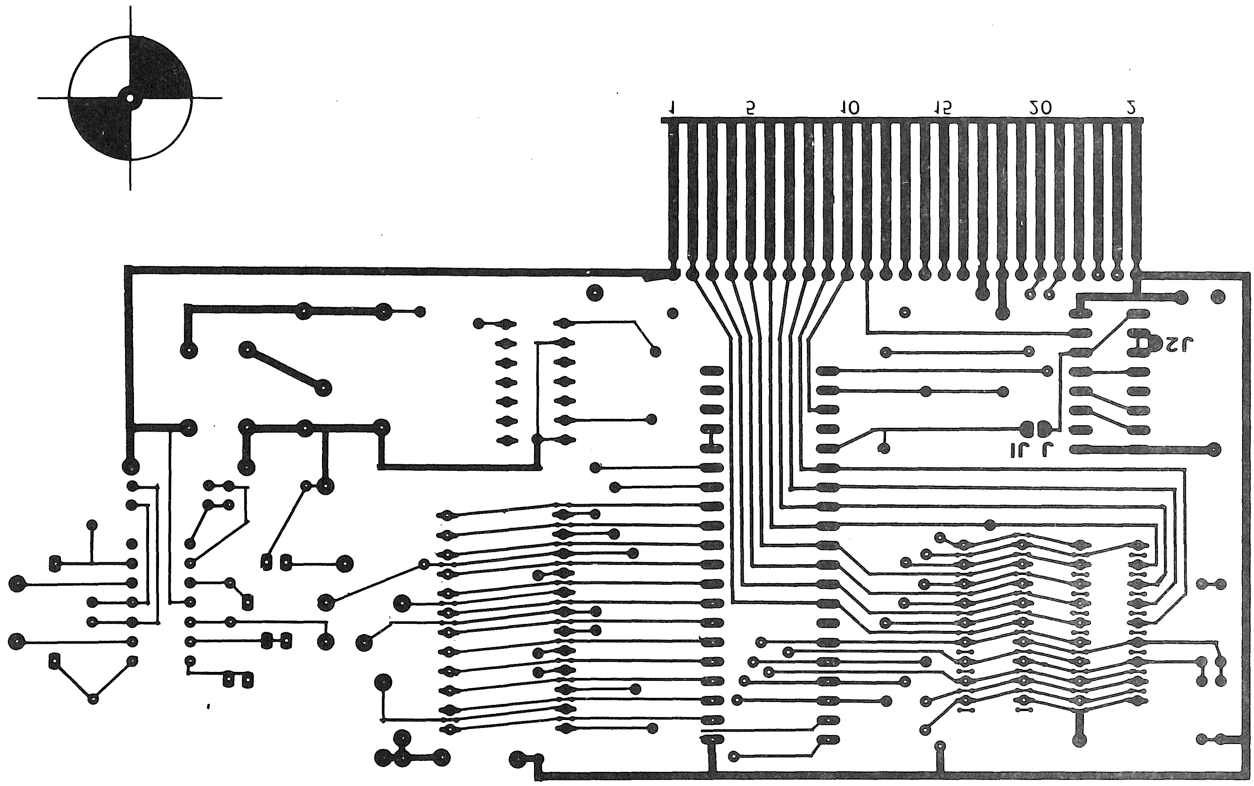
```
]CALL-151

*C400LLLLL

C400-   A9 20       LDA   #$20      INIT
C402-   8D C2 C0    STA   $C0C2
C405-   8D C0 C0    STA   $C0C0
C408-   A9 E0       LDA   #$E0
C40A-   8D CB C0    STA   $C0CB
C40D-   A9 0A       LDA   #$0A
C40F-   8D C8 C0    STA   $C0C8
C412-   A9 00       LDA   #$00
C414-   8D C9 C0    STA   $C0C9
C417-   60          RTS
C418-   EA          NOP
C419-   EA          NOP
C41A-   AD C0 C0    LDA   $C0C0
C41D-   29 10       AND   #$10
C41F-   D0 F9       BNE   $C41A
C421-   AD C0 C0    LDA   $C0C0
C424-   29 0F       AND   #$0F
C426-   49 0F       EOR   #$0F
C428-   8D FF C4    STA   $C4FF
C42B-   AD C1 C0    LDA   $C0C1
C42E-   49 FF       EOR   #$FF
C430-   8D FE C4    STA   $C4FE
C433-   60          RTS
C434-   20 00 C4    JSR   $C400     __________
C437-   20 70 C4    JSR   $C470
```

Listing Continued . . .

Continued Listing

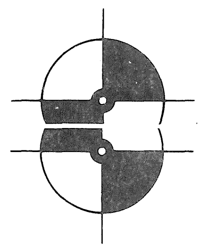
```
C43A-   AD FF C4    LDA   $C4FF
C43D-   20 DA FD    JSR   $FDDA
C440-   AD FE C4    LDA   $C4FE
C443-   20 DA FD    JSR   $FDDA
C446-   20 62 FC    JSR   $FC62      Not Used
C449-   20 89 C4    JSR   $C489
C44C-   18          CLC
C44D-   90 EB       BCC   $C43A
C44F-   20 62 FC    JSR   $FC62
C452-   CA          DEX
C453-   D0 EB       BNE   $C440
C455-   4C 59 FF    JMP   $FF59
C458-   F7          ???
C459-   F7          ???
C45A-   F7          ???
C45B-   FF          ???
C45C-   7D FB FA    ADC   $FAFB,X
C45F-   FF          ???
C460-   A9 00       LDA   #$00
C462-   8D C0 C0    STA   $C0C0      __________
C465-   A0 05       LDY   #$05
C467-   88          DEY
C468-   D0 FD       BNE   $C467
C46A-   A9 20       LDA   #$20
C46C-   8D C0 C0    STA   $C0C0
C46F-   60          RTS
C470-   AD 62 C0    LDA   $C062      INIT
C473-   30 FB       BMI   $C470
C475-   A9 4E       LDA   #$4E
C477-   8D C4 C0    STA   $C0C4
C47A-   A9 C7       LDA   #$C7
C47C-   8D C5 C0    STA   $C0C5
C47F-   20 60 C4    JSR   $C460
C482-   20 1A C4    JSR   $C41A
C485-   60          RTS
C486-   EA          NOP
C487-   EA          NOP
C488-   EA          NOP
C489-   AD C8 C0    LDA   $C0C8      MEASURE
C48C-   D0 FB       BNE   $C489
C48E-   A9 0A       LDA   #$0A
C490-   8D C8 C0    STA   $C0C8
C493-   A9 00       LDA   #$00
C495-   8D C9 C0    STA   $C0C9
C498-   20 60 C4    JSR   $C460
C49B-   20 1A C4    JSR   $C41A
C49E-   60          RTS
C49F-   EA          NOP
C4A0-   AD 62 C0    LDA   $C062      OFF
C4A3-   8D FD C4    STA   $C4FD
C4A6-   60          RTS
C4A7-   00          BRK
C4A8-   8D CB C0    STA   $C0CB
C4AB-   60          RTS
```

Listing Continued . .

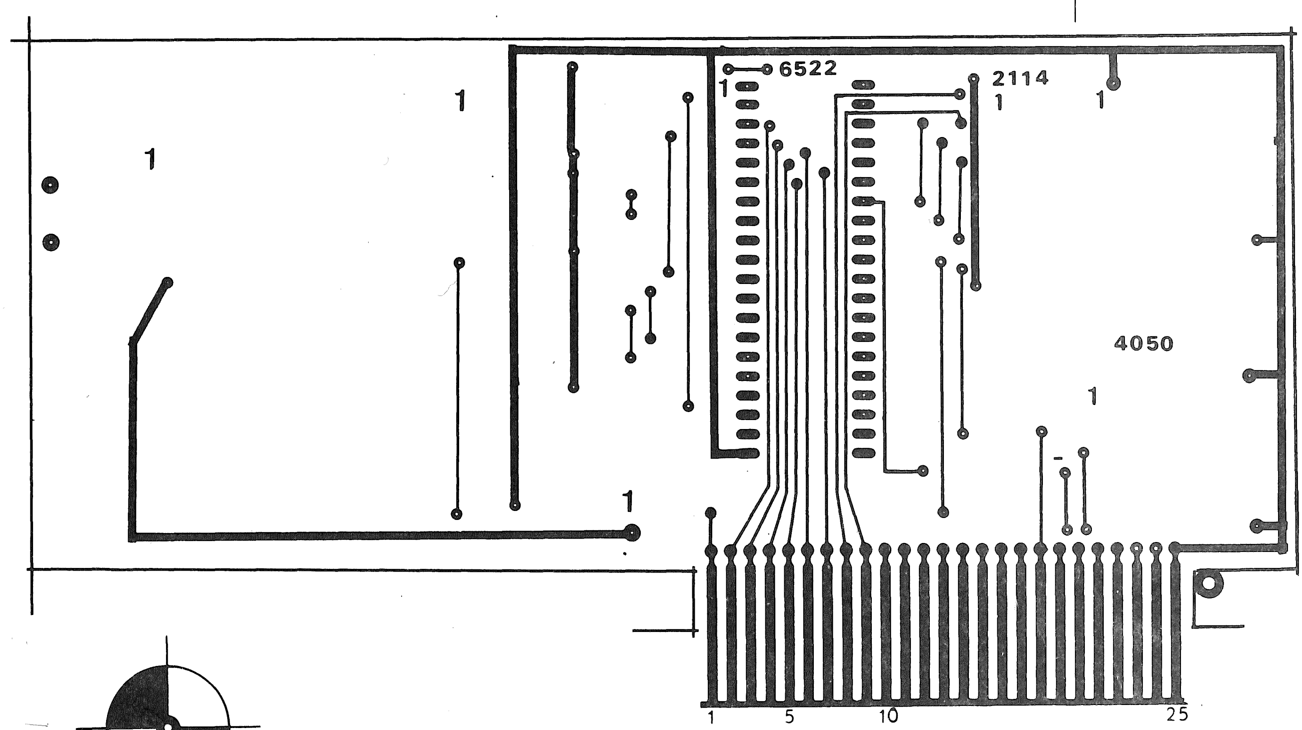


(Bottom)

Printed Circuit Board



(Top)



 "Is written on a blank page to avoid confusion" is written on a blank page to avoid confusion . . .

Continued Listing

```
C4AC-    FF            ???
C4AD-    BF            ???
C4AE-    FB            ???
C4AF-    BF            ???
C4B0-    DD A4 FF      CMP
]CALL-151

*C400.C433

C400- A9 20 8D C2 C0 8D C0 C0
C408- A9 E0 8D CB C0 A9 0A 8D
C410- C8 C0 A9 00 8D C9 C0 60
C418- EA EA AD C0 C0 29 10 D0
C420- F9 AD C0 C0 29 0F 49 0F
C428- 8D FF C4 AD C1 C0 49 FF
C430- 8D FE C4 60
*
 C460.C4A6

C460- A9 00 8D C0 C0 A0 05 88
C468- D0 FD A9 20 8D C0 C0 60
C470- AD 62 C0 30 FB A9 4E 8D
C478- C4 C0 A9 C7 8D C5 C0 20
C480- 60 C4 20 1A C4 60 EA EA
C488- EA AD C8 C0 D0 FB A9 0A
C490- 8D C8 C0 A9 00 8D C9 C0
C498- 20 60 C4 20 1A C4 60 EA
C4A0- AD 62 C0 8D FD C4 60
*
```

The data of the ADC 1210 is transferred to the BASIC program via the memory locations **C4FF** (MSB) and **C4FE** (LSB). Another memory location, **C4FD** is used as a flag to stop the measurement.

To start a measurement, we use the push button P0 on the game I/O connecter of the Apple. By pressing this button the timers are set and the first measurement is taken. The value is stored in array MW(I). This array will contain all measurements. As they are taken they will be stored in the next higher sub-array. To figure the exact voltage that you have just measured, it is necessary to multiply it by a scale factor which is: A = Vref/4096 (= 0.00125 volts with a reference voltage of 5.12 volts).

The storing and calculating in the BASIC program takes about 4 tenths to 6 tenths of a second. The rest of the time the program waits in the subroutine MEASURE for the rest of the second to elapse.

For this application, the I/O board was redesigned and made into a printed circuit instead of using the prototyping area.

The following Figures 3.30 and 3.31 show the layout of the board and of the parts:

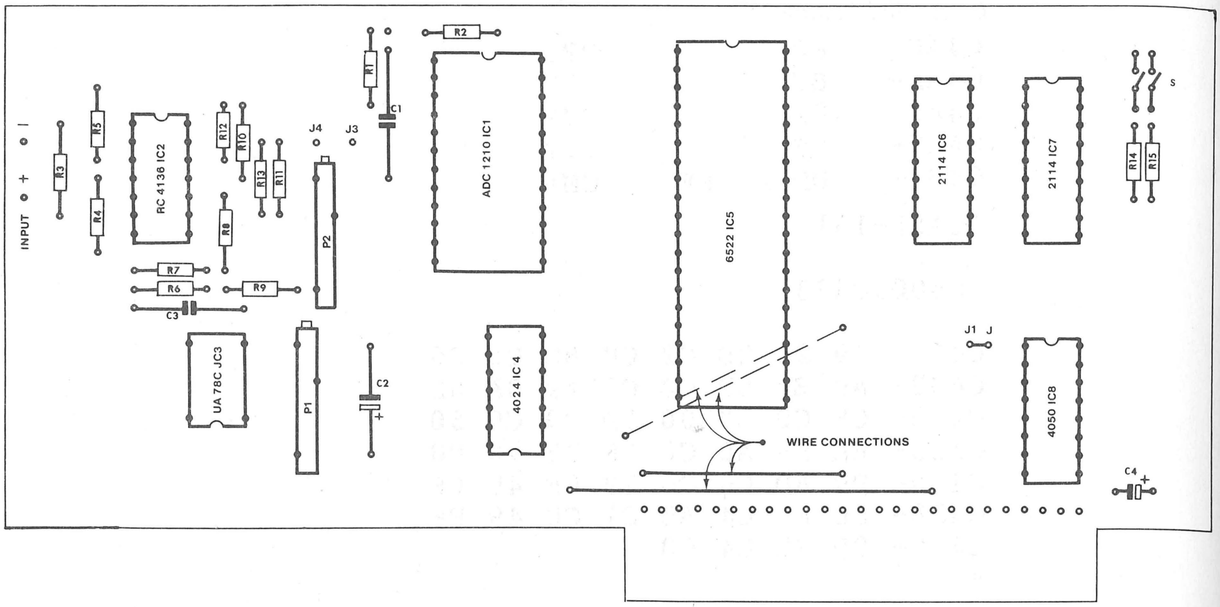
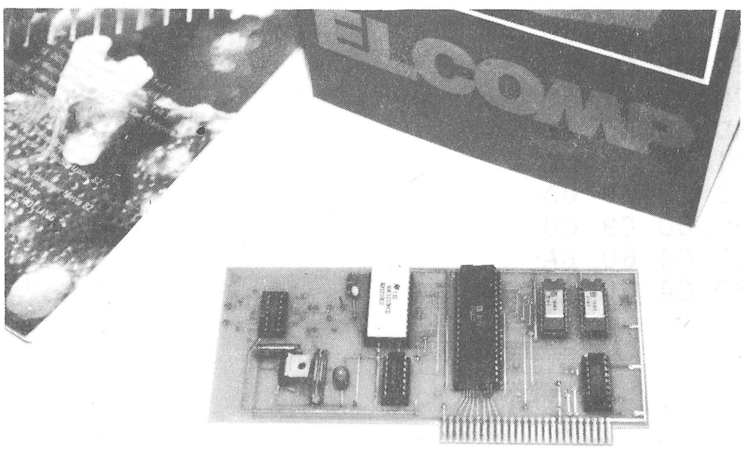


Figure 3.30 Layout of the 1210 Board



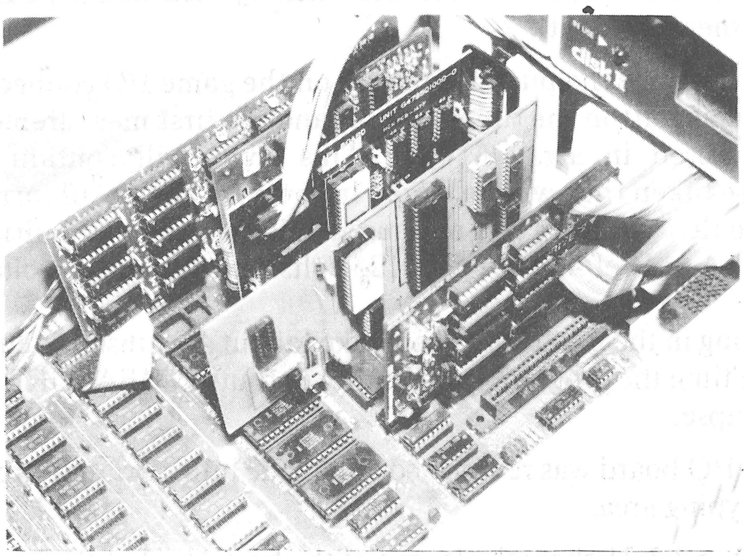


Figure 3.31 Parts

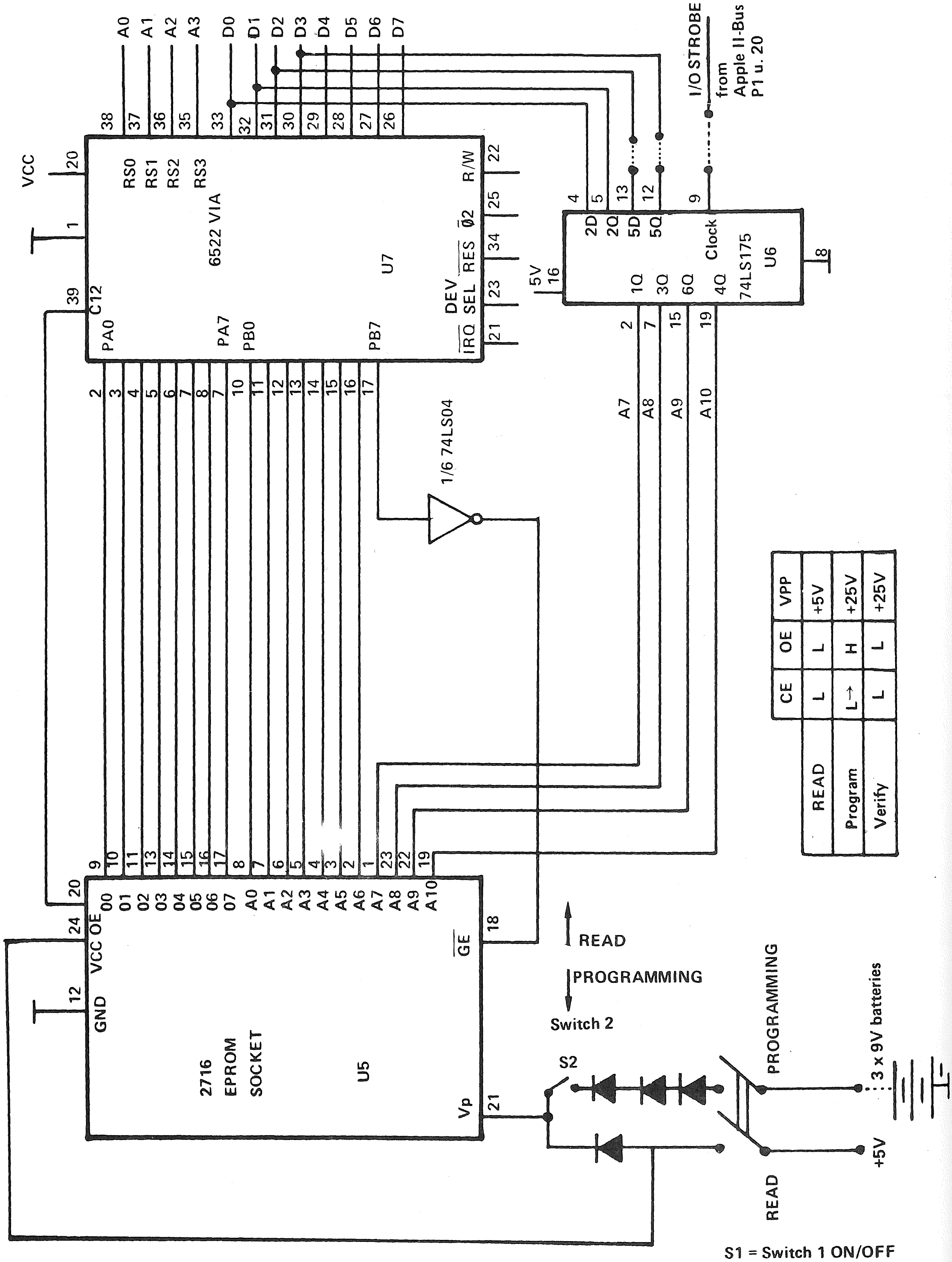
R1	200k	
R2	1k	
R3 – R13	(See Text)	
R14, R15	4.7k	
C1	100 pF Ceramic	
C2	10 μF/35 V	
C3	0.1 μF	
C4	10 μF/35 V Tantal	
P1, P2	5k Trimmer	
IC1	ADC 1210	N.S.
IC2	RC 4136	T.I.
IC3	UA 78G	Fairchild
IC4	4024	
IC5	6522	
IC6, IC7	2114	
IC8	4050	Motorola
S	2 Pole Dual In-line Switch	

4

An Eprom Burner For The Apple Computer.

Why do you need an Eprom burner? The first major advantage, if you are into hardware at all, is that there comes a time when you realize how nice it would be to be able to put routines that are used most often in a nice safe spot (which the Eprom would allow you to do). If you decide to get into hardware development or special applications and control systems, you will be able to use your Apple computer with the 6522 and the Eprom burner circuitry to actually create your own microprocessor boards for specific applications. In this chapter we will deal with how to construct an Eprom burner circuit and tie it into your Apple computer system, allowing you to experiment with hardware development and system control applications. To build the Eprom burner we will use a card very similar to the 6522 I/O board described earlier. For this project, you won't need the additional RAM's that were on the original board. Because of the complexity of the circuits in this new project, we aren't going to start off by trying to modify the old board, but we will start anew with the overall schematic of the whole board, showing you how to construct the new circuits and add the extra components that will be necessary on the prototype side of the board. In this project, the prototype area of the board will be converted into an actual printed circuit board in order to make it permanent and reliable. This project, like all of the projects described so far, can be used in any open slot of the Apple computer. However, this chapter was written with the idea in mind that it would be placed in slot 4. If you wish to use the software and the board in another slot, you will have to modify the addresses in the program to point to the addresses of the other slots.

For this project you will also need a 25-volt supply voltage for the full burning of the Eprom, and this can be accomplished by tying together three 9-volt batteries in series or building your own DC power supply. If you are going to use three 9 volt batteries in series, you will get 27 volts, but since the Eprom burner requires 25 ± 0.5 volts, it will be necessary to put three or four diodes in series. It would be advisable to check with a meter to insure that you do have the actual voltage you need. On the far left hand side of the board, in the prototype area, there are already spaces for 5 diodes, so it would be easy to simply put diodes there and hook them up until the voltage is proper for what you need.



	CE	OE	VPP
READ	L	L	+5V
Program	L→	H	+25V
Verify	L	L	+25V

Figure 4.1 An Eprom Burner

The application described in this chapter will only work using the 2716 Eprom that has a single 5 volt power supply. The programming of the Eproms is performed utilizing the 6522 versatile interface adapter and a 74LS175 quad flip-flop. In order for data to be transferred to the Eprom that you wish to burn in, it is necessary to first make that data available to the CPU, which will then transfer it over the data lines to the 6522. With the software being used it is assumed that the 6522 VIA board (I/O board) will be in slot 4. The 6522 will store that information until the appropriate time and then transfer it to the Eprom through Port A. In order to address the Eprom so that it knows information is coming, you must use the 7 least significant bits of Port B plus the 4 outputs of the 74LS175 quad flip-flop. This allows you to address the 2K where the Eprom believes it is residing at the moment. We will explain more about Bit 8 of Port B a little later. For now, just think of bit 8 as a pulse that will be made to go high for the required length of time to burn the information in.

As we can only use 7 lines of Port B for addressing the Eprom, the remaining 4 address lines are provided by a 74LS175 quad latch. The 8 lower bits of the address are first stored in Port B and in memory location LACL (see Fig. 4.5, subroutine EOUT). The higher address bits are stored in memory location LACH. Next, LACL is rotated left one time. The Bit 8 of that location is shifted into the carry bit. With a rotate left of LACH, the carry bit becomes the lowest address bit in this location. The following instruction creates a strobe pulse which stores the 4 remaining bits in the quad latch.

Now that we have the address stored in the 7 least significant bits of Port B and the four outputs of the flip-flops in the 74LS175, we need to get the data that we want to store in the address that we programmed. At this time the data we wish to transfer will be transferred to Port A of the 6522 by a store instruction. From this point on, the actual address is available at the Eprom pins (input pins), and the data is now also available to the Eprom from Port A. Transfer of the information concerning the address we want to burn in and the data we want burned in the memory location within the Eprom is accomplished by a pulse of specific duration (very close to 50 milliseconds). This is passed to the chip through the most significant bit of Port B to pin 18 of the Eprom socket. During the entire Eprom burning-in process, the voltage applied to pin 21 must be held at a constant 25 volts to insure a stable burn-in.

To make the appropriate voltages available when needed, there are two switches on the far left-hand side of the prototype area. The reason for the two switches is safety. The bottom switch provides the 25-volt burn-in voltage, while the top switch is a safeguard to insure that you cannot remove the 5-volt operating voltage from the chip while the 25 volts is applied. When both switches are down, nothing happens. When the top switch is in the up position, it applies the 5-volt operating voltage to the chip in order to allow you to read or write. When the bottom switch is down, this is the read position, enabling you to read from the Eprom. When the bottom switch is up, provided the top switch is also up, you will obtain the 25 volts necessary to burn in the information you wish to the Eprom. The top switch is a double-pole double-throw. The bottom one is a single-pole double-throw switch.

In the schematic, the pin labeled CE is the chip-enable pin. In order to read from the Eprom, the CE pin must be low, and the pin labeled OE must also be low. The chip must be supplied with the 5-volt operating voltage at the same time that CE and OE are low in order to read. The programming voltage comes in through pin 21 by having both S1 and S2 in the up position.

One thing to note is that it is possible to verify what you are burning in. In order to verify, the CE and OE pins must be low, and the 25 volts used for burning in must be applied to the chip. You must not run a verifying cycle for very long, or you may damage the chip. However, if you use the software supplied, everything will be taken care of. It is only when you experiment with the software that these things become important.

Using the Eprom Burner

Once the board has been fully assembled according to the instructions given at the end of this chapter, it would be a good idea to inspect the board for solder bridges, little balls of solder, and bad places. Also examine it generally to be sure the chips are in the right orientation according to the diagram and that the jumpers are in place. Then you can put the Eprom burner into slot 4 of the Apple computer. The next step is to insure that both switches are in the down (or off) position. At this point you can place the Eprom into the Eprom socket without regard to whether the Apple is on or off as the switches (if they are both in the down position) isolate the Eprom socket from the rest of the computer, and it will not crash or reset. Unless you are sincerely interested in the operation of a burned-out Eprom, it is a very good idea to make sure that both switches are down and you have inserted the Eprom with pin 1 lined up with the 1 on the circuit board (the nose will point at the 6522 chip). If S2 (the bottom switch) is down and S1 (the top switch) is up, you can read the contents of the Eprom into the computer's RAM. The next step, performed by the software, is to read the Eprom into memory, or, in the case of a new Eprom, to see if it's fully erased. The software will do this for you automatically and signal you as to whether the chip is fully erased or the read was succesful.

Using the Software

First you have to enter the monitor by a CALL -151 from BASIC. Then you start this program from the Apple monitor by **800G**. You will get a prompt at the top of the screen indicating what you should do. By typing the first character of any of the three words that appear at the top of the screen you will initiate that function. If you type an 'R' at this point, the entire contents of the Eprom will be read into memory locations **4000 - 47FF**. If you only want to read in one section on the Eprom, you must start the program by **803G**. If you do this, first change the contents of several memory locations in memory so that the program will be able to find the parameters for reading or burning that section of the Eprom.

The following table shows the addresses to place the starting location you wish to read from, the ending location you wish to read from, the starting address you wish the information to be written to, and the ending address it will be written to.

10	SAEPL	Starting Address EPROM Low
11	SAEPH	Starting Address EPROM High
12	EAEPL	Ending Addreee EPROM Low
13	EAEPH	Ending Address EPROM High
14	SAPL	Starting Address Program Low
15	SAPH	Starting Address Program High
16	EAPL	Ending Address Program Low
17	EAPH	Ending Address Program High
Default Values:	10 = 00 11 = 00 12 = FF 13 = 7F	14 = 00 15 = 40 16 = FF 17 = 47

Figure 4.2 Table of the Addresses

One of the nice features of this table is that you can select any free memory locations anywhere in RAM to read the information into or to store the information to be burned into the Eprom. The physical addresses for the Eprom are always in the range of **0000** to **07FF**. Because these addresses are stored in the 6522 and the 74LS175 as described above, they don't correspond with the appropriate addresses in the computer. Because of the way they are stored they are not always the same addresses you would get if you did a PEEK to memory locations **0000** through **07FF**. You need to run the program that will enable the 6522 board in order to read the actual Eprom information at those addresses.

For example, if you want to read the physical memory locations **05** to **15** of the Eprom, you must set the starting address as follows: SAEPL = **05** and SAEPH = **00**. The ending address is: EAEPL = **15**, EAEPH = **00**.

Now you have to decide where the data from these memory locations should be placed. If you want them in **2005** to **2015**, you have to set SAPL = **05** and SAEPH = **20**. It is not necessary to set the ending addresses (EAPL and EAPH) because the program stops reading in memory location **0015**. Once all of the memory locations have been initialized with the values you want, start the program by going to address **803**.

Testing an Eprom.

In order to be sure that you are going to get a clean burn in a new Eprom, it's a good idea to test it before use to insure that it really is empty or erased. All contents of every memory location within the Eprom must be **FF** in order to be burned. The software provided with this application will test the entire Eprom and assure that every byte within the Eprom is actually **FF**. If it finds one that is not, it will signal you with an error message: EPROM NOT ERASED. If you wish to test just one section of the Eprom, you can use the same procedure as just described by setting the starting and ending addresses and using **803G**. If the program finds the Eprom completely erased and usable, it will give you the message: EPROM ERASED.

Programming the Eprom.

If you hit a B for burning the Eprom, the programming procedure will start. Before hitting B to start the actual burning-in process, you must be sure that both switches are in the up position. Since it requires 50 milliseconds to program each addessable byte within the Eprom, the entire burn-in procedure will take approximately 100 seconds. After each byte is burned into the Eprom, the software will do an automatic VERIFY of that byte to assure that it is there. If it finds while verifying that the byte in memory does not match the byte that was just read from the Eprom, it will generate an error message: EPROM NOT PROGRAMMED. If all goes well and every byte can be verified the message EPROM PROGRAMMED will appear at the end of the burning process. As described above, a program start of **800G** will burn in the entire 2K of the Eprom.

We can set the addresses to burn only a part of the Eprom in the same way we set the addresses for reading part of the Eprom. It is possible to burn only one single address of the Eprom. For example, if we want the program at addresses **40GF** to **4137** in the Eprom starting address **187**, we have to set the addresses as follows: SAPL = **GF**, SAPH = **40**, EAPL = **37**, EAPH = **41**, SAEPL = **87**, and SAEPH = **01**. It is not necessary to set the ending addresses (EAEPL and EAEPH), because the program will stop burning at address **4137**.

The following is a short summary of the steps required to perform the functions:

1. Insert the board into slot 4, insuring that the computer has been turned off.
2. Turn the computer on.
3. Read in the program that is going to be doing the work.
4. Insure that both switches on the board are in the down position.
5. Insert the Eprom in the Eprom circuit on the board, insuring that the nose points to the 6522 chip, and pin 1 of the Eprom is on top of pin 1 printed on the circuit board.
6. Read into memory the program you want to burn into the Eprom.
7. Flip the top switch (S1) into the up position.
8. Either go to memory location **800** to program the entire Eprom, or store the appropriate numbers in the memory locations and use **803G** for programming a part of the Eprom.

9. Be sure to test the Eprom to make sure it is completely erased and ready to burn-in.
10. Flip S2 (the bottom switch) to the up position.
11. Start the burning-in by either **800G** or **803G.**
12. Upon completion of the burn-in, turn S1 and S2 down and remove the Eprom.

Assembling the Eprom Burner Board.

The first step is to assemble the right hand side of the board in the same manner as we have for the previous projects, such as the 6522 I/O board, but in this project it won't be necessary to mount the additional RAM as was done on the other boards. The next step is to mount all required sockets and solder them in. Install the textool zero insertion-force socket, with the handle pointing at the 6522 chip.

Now you must provide a 25-volt power supply or use three 9-volt batteries in series. In the latter case, it will be necessary to install at least three diodes on the left hand side of the printed circuit board.

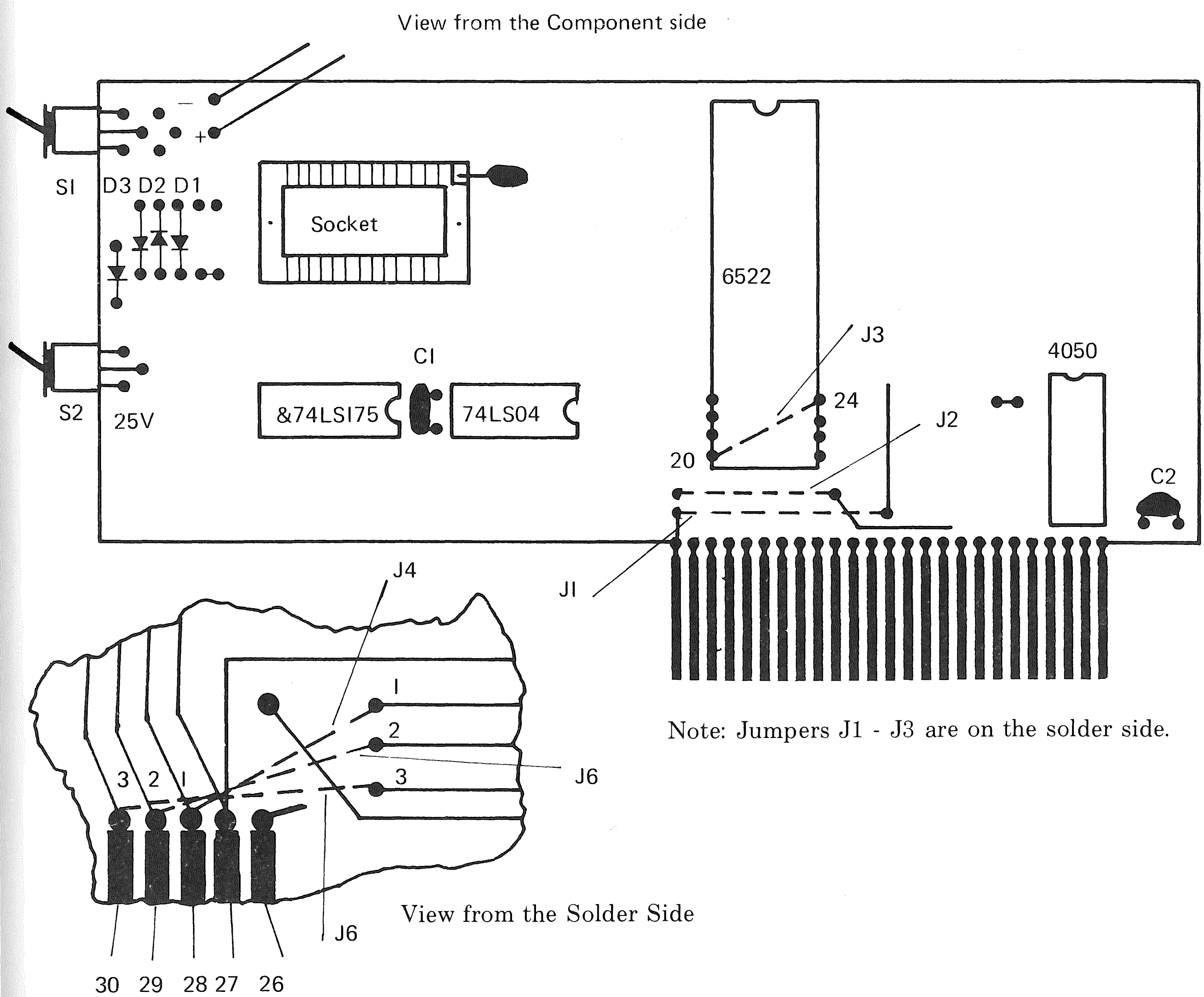


Figure 4.3 Parts Layout

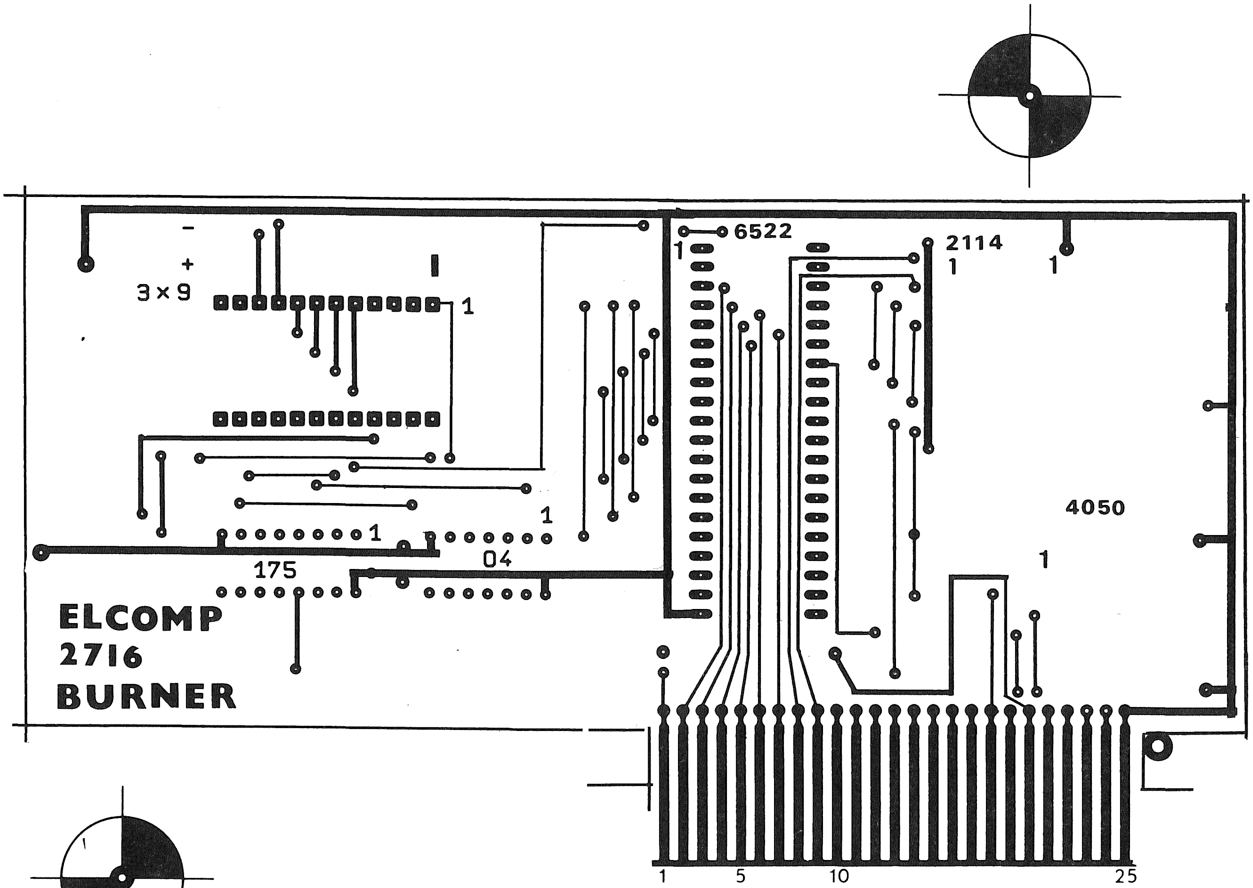
Install the two switches, S1 and S2, making sure that you get the double-pole double-throw at the top and the single-pole double-throw at the bottom. The bottom set of contacts on S1 will be wired to the first three holes in triangular array closest to the left-hand side of the board. The top three leads of S1 will be put in the second three holes. If the leads don't reach, you can attach short jumpers to make the connections. Then connect your 25-volt power supply (or the three batteries in series) to the two holes marked on the schematic with + and - near switch 1 at the top of the printed circuit board. Attach the jumpers on both sides of the board. Be sure they are in the right position; then check them again. There are two jumpers to be installed on the front (or top) side of the board: J1 and J2.

Now turn the board over, and on the back (or bottom side) of the board install J3, 4, 5 and 6. Pay particular attention to getting these in the right place. Now you need to mount two capacitors, C1 and C2. C1 goes between the two IC's, 74LS175 and 74LS04. C2 goes in the right-hand bottom corner of the 6522 VIA board. Now you can plug in all of the IC'S into their respective sockets, making sure that the nose goes in the same direction as shown on the schematic and that pin 1 printed on the circuit board lines up with pin 1 on the IC's as you plug them in.

Figure 4.4 Parts List for the Eprom Burner

Qty	Description
1	Capacitor tantal 10 μF/35V
1	Capacitor 100 nF
1	DPDT-Switch
1	SPDT-Switch
1	Diode 2N 4148
1	14 pin socket DIL
1	16 pin socket DIL
1	18 pin socket DIL
1	40 pin socket DIL
1	24 pin socket TEXTOOL
1	6522 (Rockwell)
1	4050 (Motorola)
1	74LS154
1	74LS04
1	PC-Board EPROM-BURNER
3	Diodes 2N 4148 see text

(Top)



(Bottom)

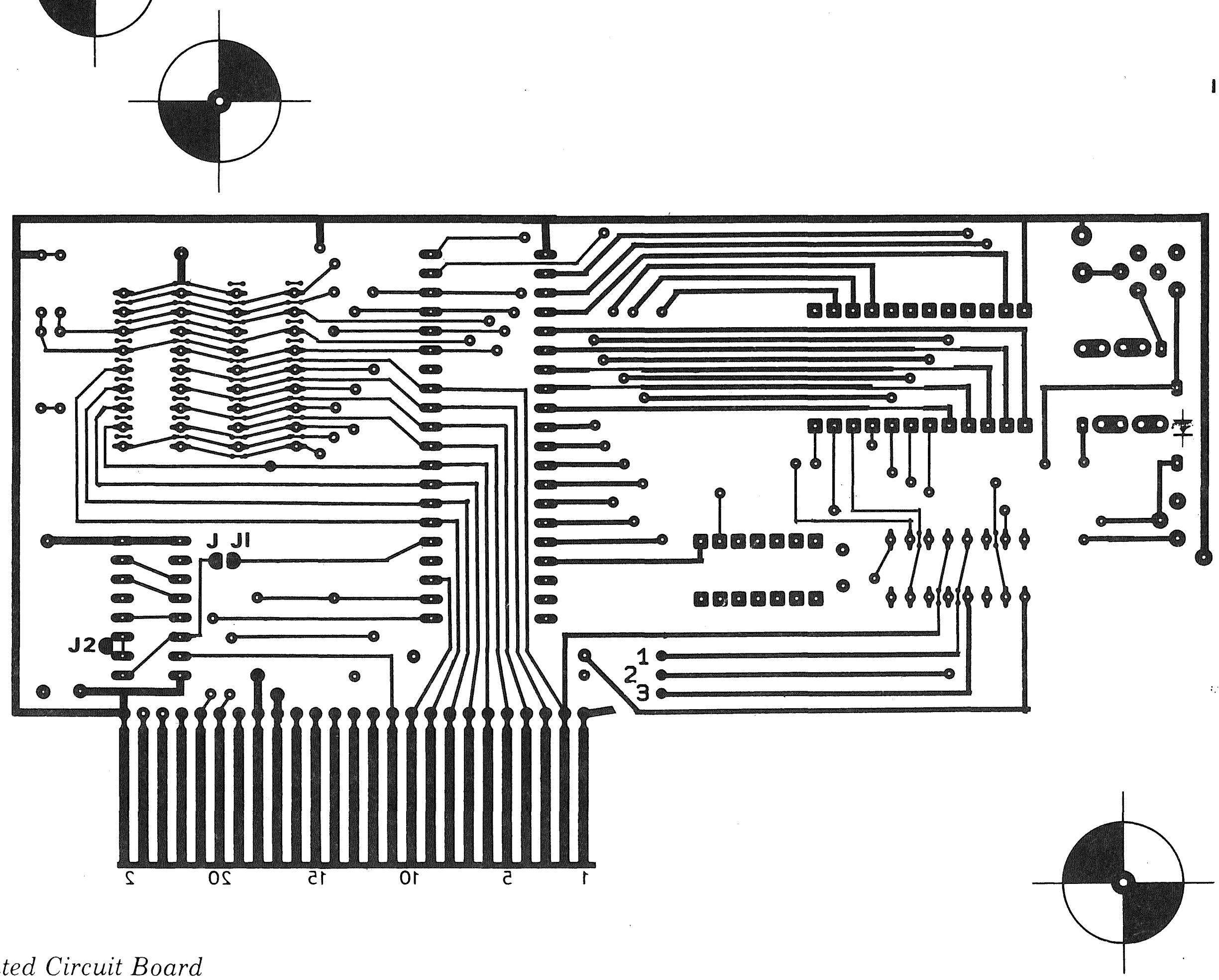


Figure 4.6 Printed Circuit Board

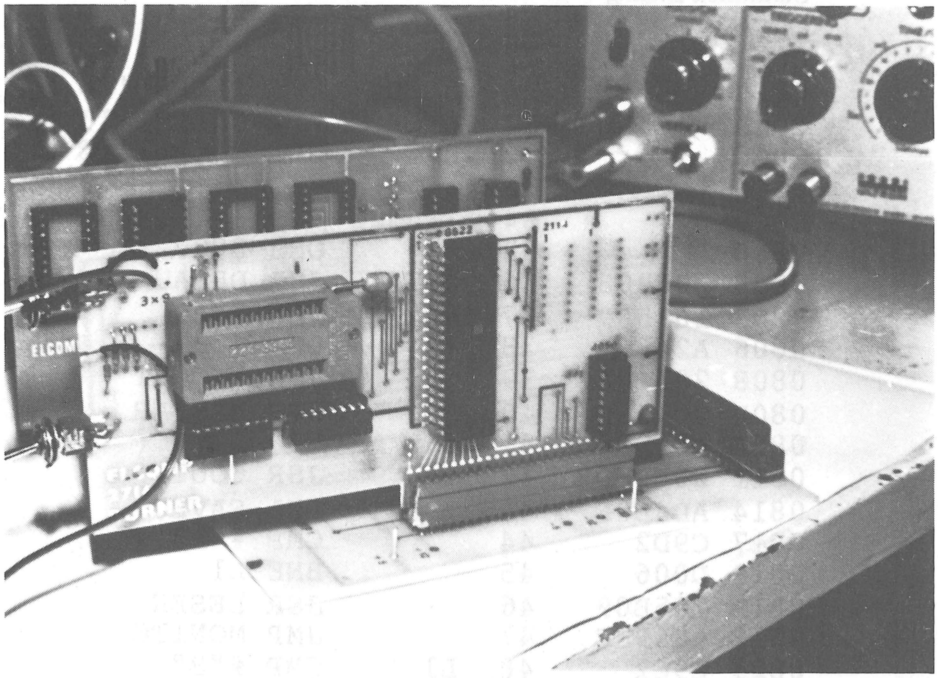


Photo of the Eprom Burner Board

In the following Figure (4.5) the program for burning the Eprom and the hex codes of this program are shown.

Figure 4.5 Eprom Program

```
0800                    DCM "PR#1"
C0C0                    ORG $C0C0
C0C0           TORB     EQU *
C0C0           TORA     EQU *+!1
C0C0           DDRB     EQU *+!2
C0C0           DDRA     EQU *+!3
C0C0           T1CL     EQU *+!4
C0C0           T1CH     EQU *+!5
C0C0           ACR      EQU *+!11
C0C0           PCR      EQU *+!12
C0C0           IFR      EQU *+!13
C0C0           ;
C0C0           ;
C0C0           STR      EQU $C800
C0C0           COUT     EQU $FDED
C0C0           RDCHAR   EQU $FD35
C0C0           HOME     EQU $FC58
C0C0           MONITO   EQU $FF59
C0C0           ;
C0C0           SAEPL    EPZ $10
C0C0           SAEPH    EPZ SAEPL+!1
C0C0           EAEPL    EPZ SAEPL+!2
C0C0           EAEPH    EPZ SAEPL+!3
C0C0           SAPL     EPZ SAEPL+!4
C0C0           SAPH     EPZ SAEPL+!5
C0C0           EAPL     EPZ SAEPL+!6
C0C0           EAPH     EPZ SAEPL+!7
```

Listing Continued . . .

Continued Listing

```
C0C0           LAL    EPZ SAEPL+!8
C0C0           LAH    EPZ SAEPL+!9
C0C0           LACL   EPZ SAEPL+!10
C0C0           LACH   EPZ SAEPL+!11
C0C0           HFZ    EPZ SAEPL+!12
C0C0           ;
C0C0           ;
0800                  ORG $800
0800 20C208    CSTART JSR DEFAU
0803 201509    WSTART JSR INIT
0806 A246             LDX #70
0808 203B08           JSR TXTOUT
080B 2035FD           JSR RDCHAR
080E 8D4D08           STA SAVEC
0811 20EDFD           JSR COUT
0814 AD4D08           LDA SAVEC
0817 C9D2             CMP #"R"
0819 D006             BNE L1
081B 205B09           JSR LESEN
081E 4C59FF           JMP MONITO
0821 C9C2      L1     CMP #"B"
0823 D006             BNE L2
0825 20CB09           JSR PROGRA
0828 4C59FF           JMP MONITO
082B C9D4      L2     CMP #"T"
082D D0D4             BNE WSTART
082F 207709           JSR PRUEFE
0832 A265             LDX #101
0834 203B08           JSR TXTOUT
0837 4C59FF           JMP MONITO
083A 00               BRK
083B           ;
083B           ;
083B 8D4B08    TXTOUT STA SAVEA
083E BD4E08    TXT1   LDA TEXT,X
0841 F007             BEQ FIN
0843 20EDFD           JSR COUT
0846 E8               INX
0847 18               CLC
0848 90F4             BCC TXT1
084A 60        FIN    RTS
084B 0000      SAVEA  HEX 0000
084D 00        SAVEC  HEX 00
084E           ;
084E           ;
084E 8D8D      TEXT   HEX 8D8D
0850 C5D0D2           ASC "EPROM NOT EREASED     "
0853 CFCDA0
0856 CECFD4
0859 A0C5D2
085C C5C1D3
085F C5C4A0
0862 A0A0A0
0865 008D             HEX 008D
0867 C5D0D2           ASC "EPROM NOT PROGRAMMED     "
086A CFCDA0
```

Listing Continued . . .

Continued Listing

```
086D CECFD4
0870 A0D0D2
0873 CFC7D2
0876 C1CDCD
0879 C5C4A0
087C A0A0A0
087F 008D      77            HEX 008D
0881 C5D0D2    78            ASC "EPROM PROGRAMMED  "
0884 CFCDA0
0887 D0D2CF
088A C7D2C1
088D CDCDC5
0890 C4A0A0
0893 008D      79            HEX 008D
0895 C2A9D5    80            ASC "B)URNING T)ESTING R)EADING   "
0898 D2CEC9
089B CEC7A0
089E D4A9C5
08A1 D3D4C9
08A4 CEC7A0
08A7 D2A9C5
08AA C1C4C9
08AD CEC7A0
08B0 A0A0
08B2 00        81            HEX 00
08B3 8D        82            HEX 8D
08B4 C5D0D2    83            ASC "EPROM EREASED"
08B7 CFCDA0
08BA C5D2C5
08BD C1D3C5
08C0 C4
08C1 00        84            HEX 00
08C2           85      ;
08C2           86      ;
08C2           87      ;
08C2 A900      88      DEFAU LDA #$00
08C4 8510      89            STA SAEPL
08C6 8511      90            STA SAEPH
08C8 8514      91            STA SAPL
08CA A9FF      92            LDA #$FF
08CC 8516      93            STA EAPL
08CE 8512      94            STA EAEPL
08D0 A907      95            LDA #$07
08D2 8513      96            STA EAEPH
08D4 A940      97            LDA #$40
08D6 8515      98            STA SAPH
08D8 A947      99            LDA #$47
08DA 8517      100           STA EAPH
08DC 60        101           RTS
08DD           102     ;
08DD           103     ;
08DD A518      104     EOUT  LDA LAL
08DF 8DC0C0    105           STA TORB
08E2 851A      106           STA LACL
08E4 A519      107           LDA LAH
08E6 851B      108           STA LACH
```

Listing Continued . . .

Continued Listing

```
08E8 261A               ROL LACL
08EA 261B               ROL LACH
08EC A51B               LDA LACH
08EE 8D00C8             STA STR
08F1 18                 CLC
08F2 60                 RTS
08F3          ;
08F3          ;
08F3 E618     NEXT      INC LAL
08F5 D002               BNE N1
08F7 E619               INC LAH
08F9 E610     N1        INC SAEPL
08FB D002               BNE N2
08FD E611               INC SAEPH
08FF A511     N2        LDA SAEPH
0901 C513               CMP EAEPH
0903 900C               BCC N3
0905 F002               BEQ N4
0907 B00B               BCS N5
0909 A510     N4        LDA SAEPL
090B C512               CMP EAEPL
090D F002               BEQ N3
090F B003               BCS N5
0911 20DD08   N3        JSR EOUT
0914 60       N5        RTS
0915          ;
0915          ;
0915 2058FC   INIT      JSR HOME
0918 A900               LDA #$00
091A 8DC3C0             STA DDRA
091D AA                 TAX
091E A8                 TAY
091F A97F               LDA #$7F
0921 8DC2C0             STA DDRB
0924 A980               LDA #$80
0926 8DCBC0             STA ACR               ;PB7 MONOFLOP
0929 60                 RTS
092A          ;
092A          ;
092A A510     START     LDA SAEPL
092C 8518               STA LAL
092E 851A               STA LACL
0930 8DC0C0             STA TORB
0933 A511               LDA SAEPH
0935 8519               STA LAH
0937 851B               STA LACH
0939 261A               ROL LACL
093B 261B               ROL LACH
093D A51B               LDA LACH
093F 8D00C8             STA STR
0942 60                 RTS
0943          ;
0943          ;
0943 C901     ERROR     CMP #$01
0945 D008               BNE E1
0947 A200               LDX #$00
```

Listing Continued . . .

Continued Listing

```
0949 203B08            JSR TXTOUT
094C 4C59FF            JMP MONITO
094F C902     E1       CMP #$02
0951 D005              BNE E2
0953 A218              LDX #24
0955 203B08            JSR TXTOUT
0958 4C59FF   E2       JMP MONITO
095B          ;
095B 201509   LESEN    JSR INIT
095E A90C              LDA #$0C
0960 8DCCC0            STA PCR             ;OE=L LESEN
0963 202A09            JSR START
0966 ADC1C0   LES1     LDA TORA
0969 9114              STA (SAPL),Y
096B E614              INC SAPL
096D D002              BNE LES2
096F E615              INC SAPH
0971 20F308   LES2     JSR NEXT
0974 90F0              BCC LES1
0976 60                RTS
0977          ;
0977 201509   PRUEFE   JSR INIT
097A A90C              LDA #$0C
097C 8DCCC0            STA PCR             ;OE=L LESEN
097F 202A09            JSR START
0982 ADC1C0   P1       LDA TORA
0985 C9FF              CMP #$FF
0987 F005              BEQ P2
0989 A901              LDA #$01
098B 4C4309            JMP ERROR
098E 20F308   P2       JSR NEXT
0991 90EF              BCC P1
0993 60                RTS
0994          ;
0994 A90E     MONOFL   LDA #$0E
0996 8DCCC0            STA PCR             ;OE=H PROGRAMMIEREN
0999 A950              LDA #$50
099B 8DC4C0            STA T1CL
099E A9C3              LDA #$C3
09A0 8DC5C0            STA T1CH
09A3 ADCDC0   MO1      LDA IFR
09A6 2940              AND #$40
09A8 F0F9              BEQ MO1
09AA A90C              LDA #$0C
09AC 8DCCC0            STA PCR             ;OE=L
09AF 60                RTS
09B0          ;
09B0          ;
09B0 A200     CHANGE   LDX #$00
09B2 A004              LDY #$04
09B4 B510     CA1      LDA $0010,X
09B6 851C              STA HFZ
09B8 B91000            LDA $0010,Y
09BB 9510              STA $0010,X
09BD A51C              LDA HFZ
09BF 991000            STA $0010,Y
```

Listing Continued . . .

Continued Listing

```
09C2 C8                   INY
09C3 E8                   INX
09C4 E004                 CPX #$04
09C6 D0EC                 BNE CA1
09C8 A000                 LDY #$00
09CA 60                   RTS
09CB               ;
09CB               ;
09CB 202A09        PROGRA JSR START
09CE 20B009               JSR CHANGE
09D1 A9FF          PR1    LDA #$FF
09D3 8DC3C0               STA DDRA
09D6 B110                 LDA (SAEPL),Y
09D8 8DC1C0               STA TORA
09DB AA                   TAX
09DC 209409               JSR MONOFL
09DF A900                 LDA #$00
09E1 8DC3C0               STA DDRA
09E4 8A                   TXA
09E5 CDC1C0               CMP TORA
09E8 F00B                 BEQ PR3
09EA A902                 LDA #$02
09EC 4C4309               JMP ERROR
09EF E610          PR2    INC SAEPL
09F1 D002                 BNE PR3
09F3 E611                 INC SAEPH
09F5 20F308        PR3    JSR NEXT
09F8 90D7                 BCC PR1
09FA A232                 LDX #50
09FC 4C3B08               JMP TXTOUT
09FF 60                   RTS
0A00               ;
                          END

          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

SAEPL  0010  SAEPH  0011  EAEPL  0012  EAEPH  0013  SAPL   0014  SAPH   0015
EAPL   0016  EAPH   0017  LAL    0018  LAH    0019  LACL   001A  LACH   001B
HFZ    001C

** ABSOLUTE VARABLES/LABELS

TORB   C0C0  TORA   C0C1  DDRB   C0C2  DDRA   C0C3  T1CL   C0C4
T1CH   C0C5  ACR    C0CB  PCR    C0CC  IFR    C0CD  STR    C800  COUT   FDED
RDCHAR FD35  HOME   FC58  MONITO FF59  CSTART 0800  WSTART 0803  L1     0821
L2     082B  TXTOUT 083B  TXT1   083E  FIN    084A  SAVEA  084B  SAVEC  084D
TEXT   084E  DEFAU  08C2  EOUT   08DD  NEXT   08F3  N1     08F9  N2     08FF
N4     0909  N3     0911  N5     0914  INIT   0915  START  092A  ERROR  0943
```

Listing Continued . . .

Continued Listing

```
E1      094F  E2      0958  LESEN   095B  LES1    0966  LES2    0971  PRUEFE 0977
P1      0982  P2      098E  MONOFL  0994  MO1     09A3  CHANGE  09B0  CA1    09B4
PROGRA  09CB  PR1     09D1  PR2     09EF  PR3     09F5

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0212

 BR

0800- 20 C2 08 20 15 09 A2 46
0808- 20 3B 08 20 35 FD 8D 4D
0810- 08 20 ED FD AD 4D 08 C9
0818- D2 D0 06 20 5B 09 4C 59
0820- FF C9 C2 D0 06 20 CB 09
0828- 4C 59 FF C9 D4 D0 D4 20
0830- 77 09 A2 65 20 3B 08 4C
0838- 59 FF 00 8D 4B 08 BD 4E
0840- 08 F0 07 20 ED FD E8 18
0848- 90 F4 60 00 00 00 8D 8D
0850- C5 D0 D2 CF CD A0 CE CF
0858- D4 A0 C5 D2 C5 C1 D3 C5
0860- C4 A0 A0 A0 A0 00 8D C5
0868- D0 D2 CF CD A0 CE CF D4
0870- A0 D0 D2 CF C7 D2 C1 CD
0878- CD C5 C4 A0 A0 A0 A0 00
0880- 8D C5 D0 D2 CF CD A0 D0
0888- D2 CF C7 D2 C1 CD CD C5
0890- C4 A0 A0 00 8D C2 A9 D5
0898- D2 CE C9 CE C7 A0 D4 A9
08A0- C5 D3 D4 C9 CE C7 A0 D2
08A8- A9 C5 C1 C4 C9 CE C7 A0
08B0- A0 A0 00 8D C5 D0 D2 CF
08B8- CD A0 C5 D2 C5 C1 D3 C5
08C0- C4 00 A9 00 85 10 85 11
08C8- 85 14 A9 FF 85 16 85 12
08D0- A9 07 85 13 A9 40 85 15
08D8- A9 47 85 17 60 A5 18 8D
08E0- C0 C0 85 1A A5 19 85 1B
08E8- 26 1A 26 1B A5 1B 8D 00
08F0- C8 18 60 E6 18 D0 02 E6
08F8- 19 E6 10 D0 02 E6 11 A5
0900- 11 C5 13 90 0C F0 02 B0
0908- 0B A5 10 C5 12 F0 02 B0
0910- 03 20 DD 08 60 20 58 FC
0918- A9 00 8D C3 C0 AA A8 A9
0920- 7F 8D C2 C0 A9 80 8D CB
0928- C0 60 A5 10 85 18 85 1A
0930- 8D C0 C0 A5 11 85 19 85
0938- 1B 26 1A 26 1B A5 1B 8D
0940- 00 C8 60 C9 01 D0 08 A2
0948- 00 20 3B 08 4C 59 FF C9
0950- 02 D0 05 A2 18 20 3B 08
0958- 4C 59 FF 20 15 09 A9 0C
0960- 8D CC C0 20 2A 09 AD C1
```

Listing Continued . . .

Continued Listing

```
0968- C0 91 14 E6 14 D0 02 E6
0970- 15 20 F3 08 90 F0 60 20
0978- 15 09 A9 0C 8D CC C0 20
0980- 2A 09 AD C1 C0 C9 FF F0
0988- 05 A9 01 4C 43 09 20 F3
0990- 08 90 EF 60 A9 0E 8D CC
0998- C0 A9 50 8D C4 C0 A9 C3
09A0- 8D C5 C0 AD CD C0 29 40
09A8- F0 F9 A9 0C 8D CC C0 60
09B0- A2 00 A0 04 B5 10 85 1C
09B8- B9 10 00 95 10 A5 1C 99
09C0- 10 00 C8 E8 E0 04 D0 EC
09C8- A0 00 60 20 2A 09 20 B0
09D0- 09 A9 FF 8D C3 C0 B1 10
09D8- 8D C1 C0 AA 20 94 09 A9
09E0- 00 8D C3 C0 8A CD C1 C0
09E8- F0 0B A9 02 4C 43 09 E6
09F0- 10 D0 02 E6 11 20 F3 08
09F8- 90 D7 A2 32 4C 3B 08 60
0A00- 12
*
```

5

Assembling an Eprom/RAM Board

If you have written and tested your machine-language program, and successfully burned your Eprom, it now contains the programs you need for your custom application. Now it's time to think about a place to plug in your 2716 Eprom. The Apple II computer, as supplied by the manufacturer, has no possible way to plug in more Eproms. Therefore, we are going to show you a special board which you can plug into one of the empty slots in the Apple. This board will hold up to four 2716 Eproms or Eprom compatible RAMS. A complete schematic is shown below.

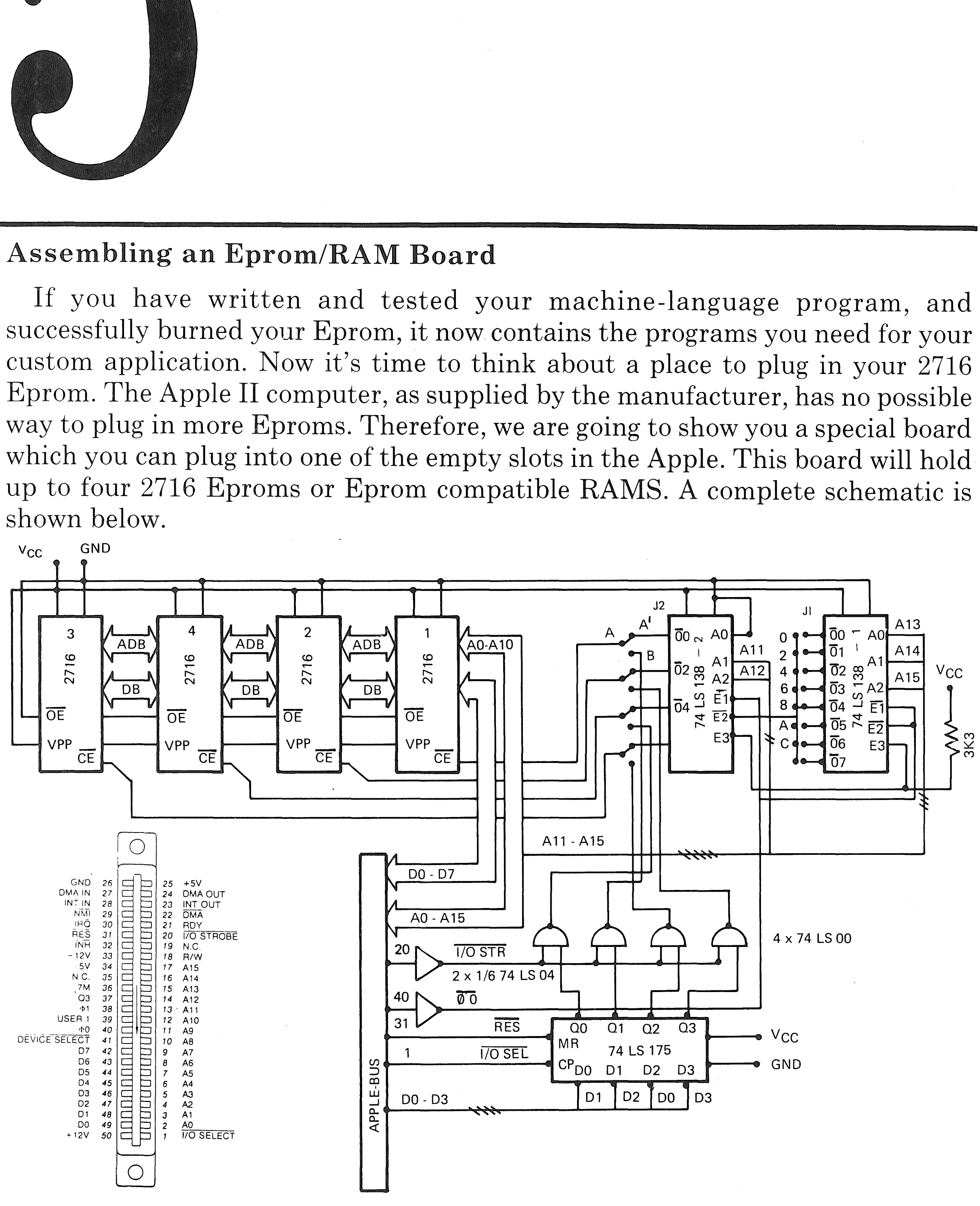


Figure 5.1 Schematic of the Eprom RAM Board

The BYTE WIDE concept recently introduced by Mostek has become very popular. BYTE WIDE allows you to use the same 24-pin sockets on your board to expand your Eprom capacity, or as an expansion of your RAM area. The Eprom compatible RAM's are available in 1K by 8-bit or 2K by 8-bit configurations. The Apple computer has a total of 8 slots, numbered from 0 to 7. Slot 0 is reserved for memory expansion, such as a language card or a ROM card. Slot 1 is reserved for a printer-driver card. Slots 2 through 5 are available for user expansion. These are the slots used by the applications described in this book. Slot 6 is usually used for the first of two disk controller boards. Slot 7 is used in Europe for an interface card for the PAL or the SECAM television systems. As you can see, there are actually only four empty slots in the Apple that you can use. There is a small limitation. You may only address one 2K byte Eprom per slot, and each socket always has the same address, **C800** to **CFFF**. This means you can only address 2K of Eprom or compatible RAM at a time. But our Eprom/RAM board allows you to use up to four 2K Eproms or RAMS. We use a specially developed bank-switching circuit to select one of these four Eproms and bring its contents into the range of **C800** to **CFFF**.

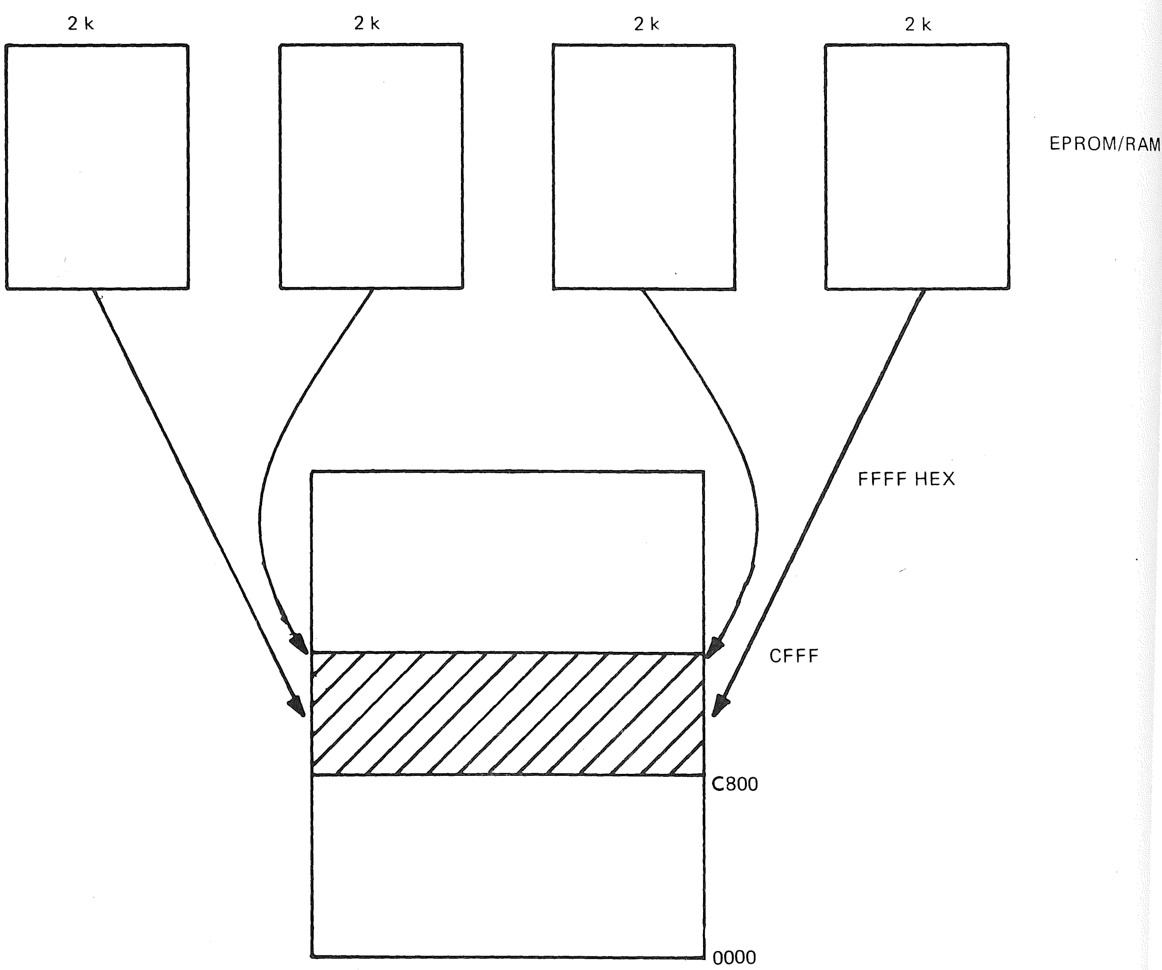


Figure 5.2 Memory Locations

Continuing from above, the Apple memory map shows you how the 2K bytes of Eprom could be brought in. Note that you can only have 2K byte at a time in the memory area **C800** to **CFFF**. If you want to use four Eprom/RAM boards in all four of the slots, 2 through 5, you can have up to 16 Eproms in your Apple; however, you can only use one 2K byte Eprom at a time. In addition to the Eprom compatible 2K RAM, there is also a 1K RAM available on the market, which has a 2716 compatible pin-out, such as the 4801 from Mostek. These are much cheaper than the 4802 (which has 2K of RAM). If you use a 4801 in the sockets of this Eprom/RAM board, you only have one half of the area **C800** to **CFFF** available. The 4801 1K RAM chips are 2K RAM chips in which one of the internal RAM areas is defective. If you use one of these chips, you have to determine which side is usable by writing and reading into the memory locations. If you want to use a 2716 Eprom and an Eprom compatible RAM together on the board, you will need to wire a jumper as shown in the following figure:

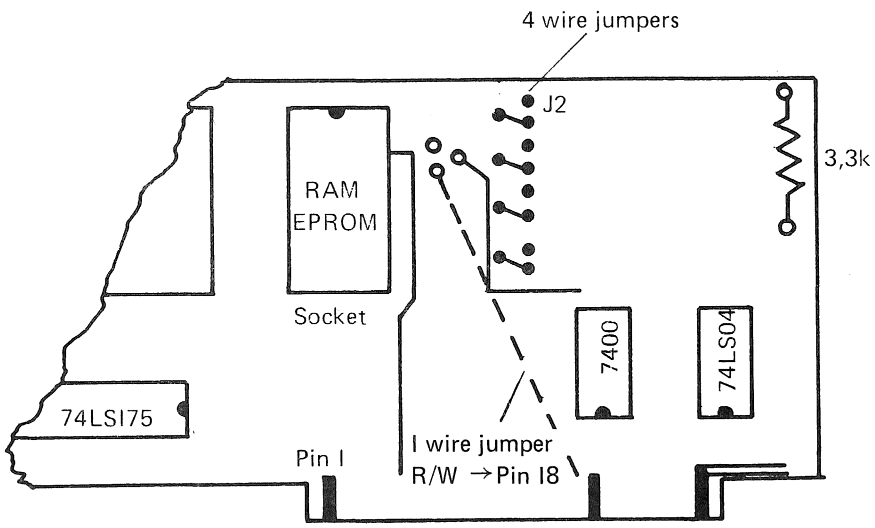


Figure 5.3 Parts Layout

Figure 5.4 also shows you how to place the components on your printed circuit board. The jumper wire is placed between pin 21 on the first Eprom/RAM socket and pin 18 of the 50 pin Apple connector. This jumper supplies the R/$\overline{W}$ signal to the RAM chip and allows reading and writing of that chip when activated. The Eprom, since it can only be read, doesn't care if pin 21 is high or low. So once the jumper has been installed, you can use RAM's or Eproms in that socket without worrying about connecting or disconnecting it anymore. To use the Eprom RAM board with the Apple II, we also have to install four small jumpers in the area marked J2 (see Figure 5.4). The Eprom/RAM has another unique feature. It allows you to first test your programs in RAM, then burn them directly into an Eprom for future use in those same memory locations.

SURVEY over the most common

24 PIN MEMORY EPROMS & RAMS

DEVICE	TYPE	MANUF	SIZE	PIN 18	PIN 19	PIN 20	PIN 21
4801	RAM	MOSTEK	1Kx8	CS*	NC	OE*	WE*
4118	RAM	MOSTEK	1Kx8	CS*	L*	OE*	WE*
4008	RAM	TI	1Kx8	CS*	AR	OE*	WE*
2716	EPROM	INTEL	2Kx8	CS*/ PROG	+12V	A10	–5V
2516	EPROM	TI	2Kx8	CS*/ PROG	A10	OE*	Vpp
3636	PROM	INTEL	2Kx8	CS3	CS2	CS1*/ PROG	A10
4802/ 4016	RAM	MOST/TI	2Kx8	CS*	A10	OE*	WE*
58725	RAM	MITSU- BISHI	2Kx8	CS	10	OE	WE

CS*/S* = Chip Select (Low)
OE* = Output ENable (Low)
WE* = Write Enable (Low)
PD = Power Down
PROG/(PE) = Program Enable
Vpp = 25V (Program Voltage)
L* = LATCH (LOW)

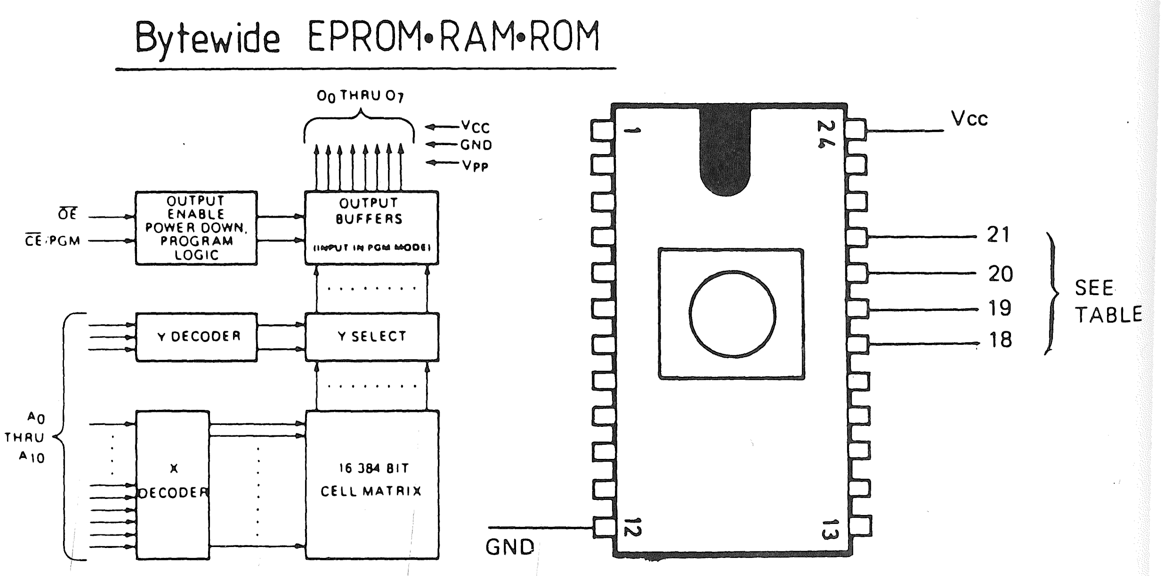
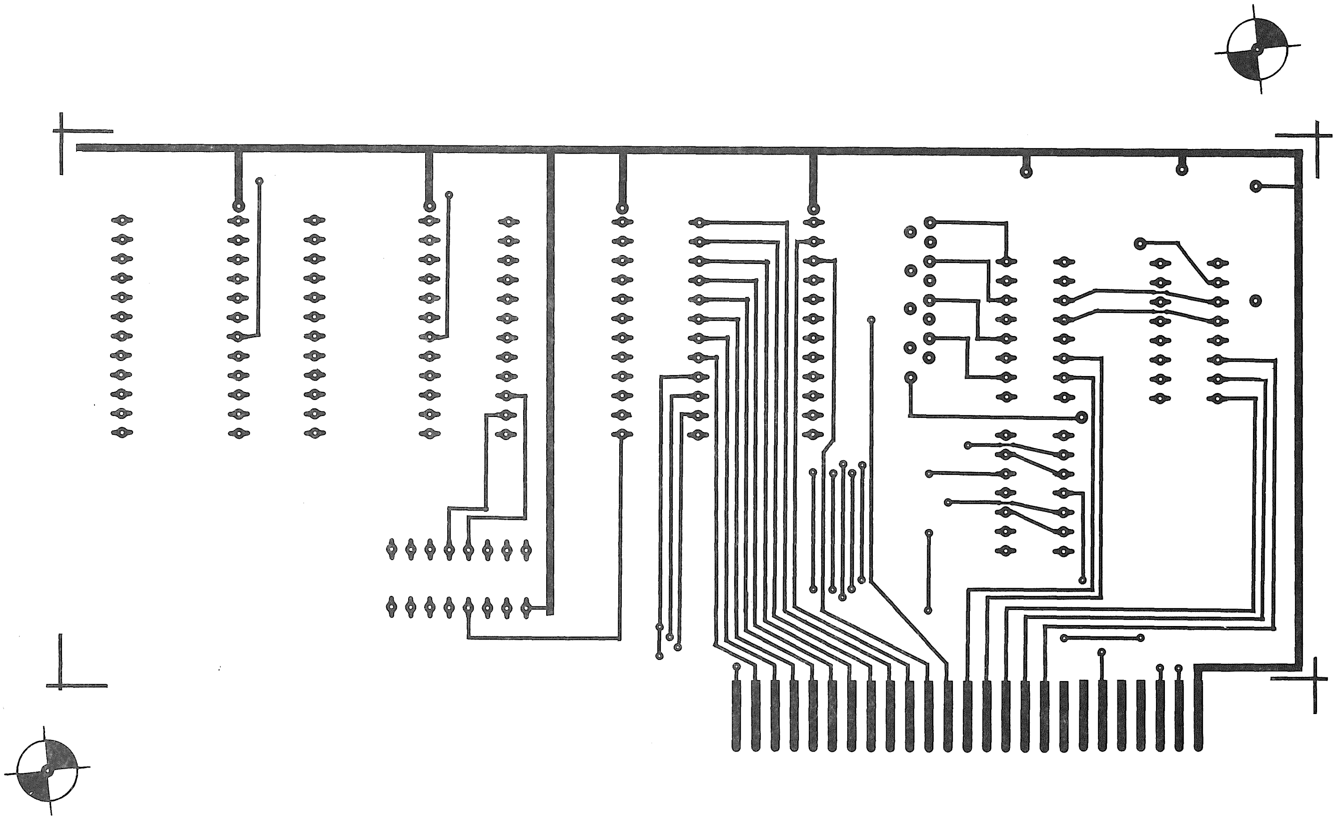


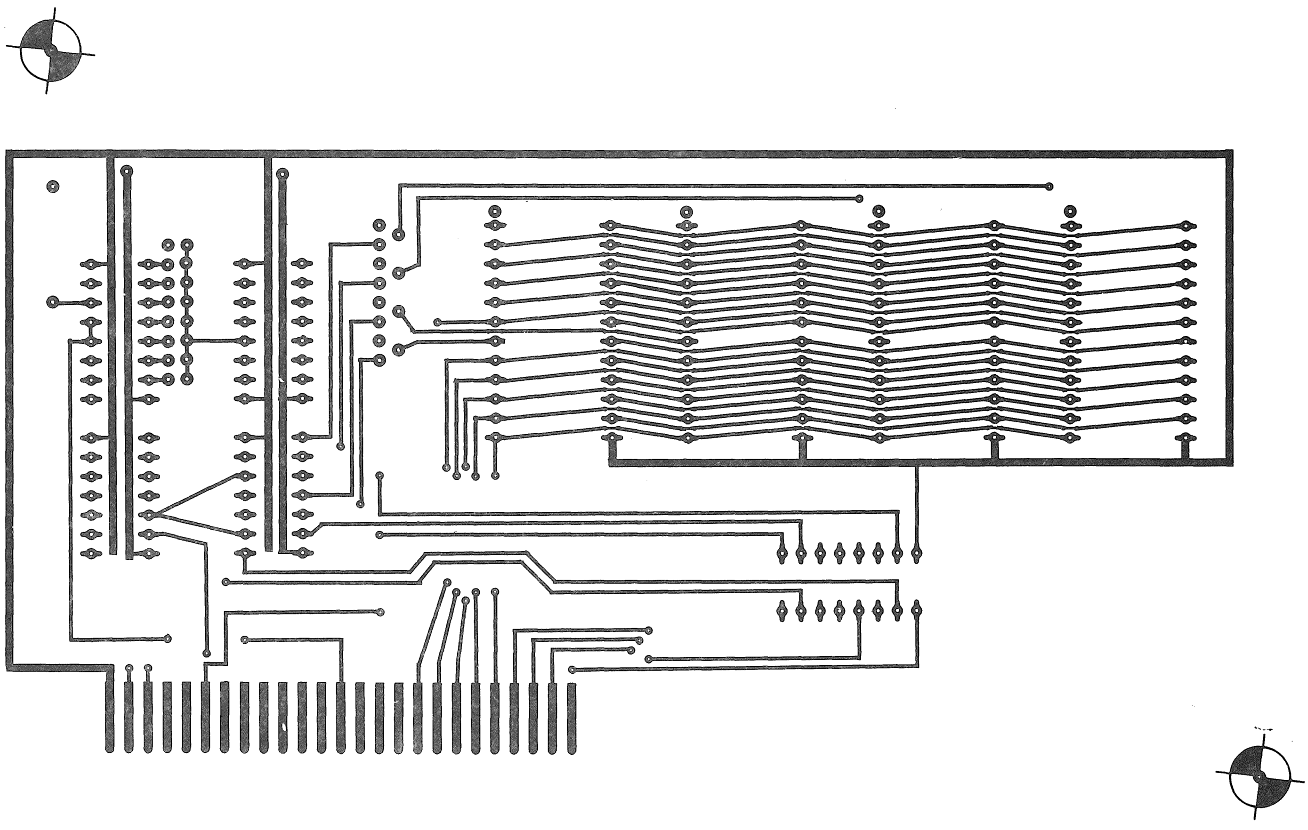
Figure 5.4 BYTE WIDE's Eprom RAM Board



(Top)

Printed Circuit Board

(Bottom)



 "Is written on a blank page to avoid confusion" is written on a blank page to avoid confusion . . .

Bank-Select Circuitry and Programming

Because of the fact that you can only use the area **C800** to **CFFF** for one Eprom at a time, a bank-switching circuit was developed to enable you to read the contents of any one of the four Eproms on the board. Through software, you can now select one of the Eproms by using the 74LS175 quad flip-flop. For example, if the Eprom RAM board is placed in slot two, you can select Eprom 1 on the board with the following machine-language instructions:

LDA # 01 and **STA $C200**.

After these instructions, the Eprom in socket 1 will be accesible in memory area **C800** to **CFFF**. If you want to select the Eprom in socket 2, enter instructions:

LDA #02 and **STA $C200**.

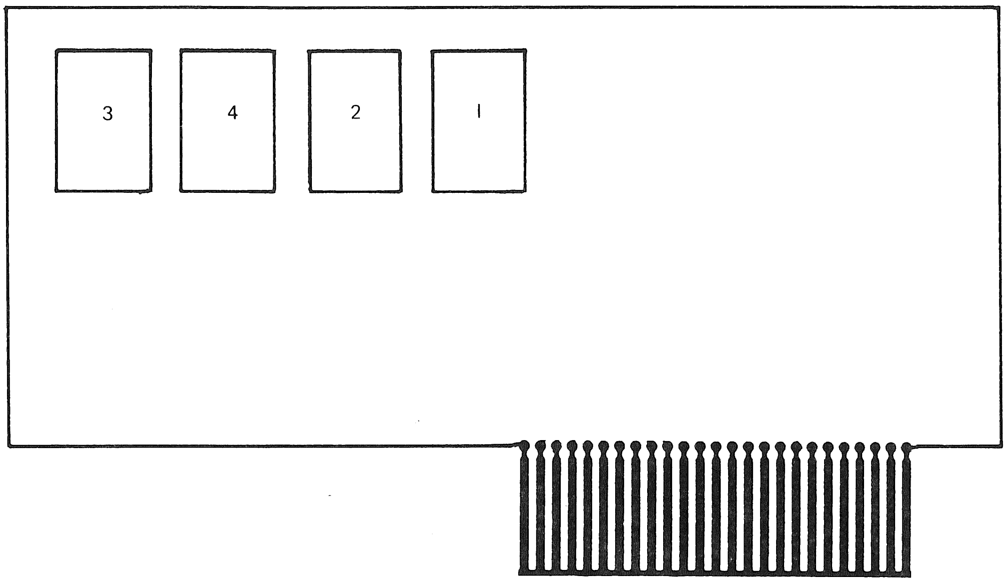


Figure 5.5 Socket Numbering

If you decide to plug your Eprom/RAM card into slot four, and you want to select the third socket, you must program **LDA #04** and **STA $C400**. In general, the Eproms can be selected as follows:

SLOT 2	SLOT 3	SLOT 4	SLOT 5	SELECT
LDA #01 STA $C200	LDA #01 STA $C300	LDA #01 STA $C400	LDA #01 STA $C500	EPROM 1
LDA #02 STA $C200	LDA #02 STA $C300	LDA #02 STA $C400	LDA #02 STA $500	EPROM 2
LDA #04 STA $C200	LDA #04 STA $C300	LDA #04 STA $C400	LDA #04 STA $C500	EPROM 3
LDA #08 STA $C200	LDA #08 STA $C300	LDA #08 STA $C400	LDA #08 STA $C500	EPROM 4

Figure 5.6 Selecting Eproms

The preceding table will make it easy to quickly look up instructions necessary to select any Eprom at any location. Since it is possible to have four boards, one in each slot with four Eproms on each board, it's possible to get a condition of jamming the data bus. To avoid this, a board must be shut off before turning on another board. The way to do this is to do a **LDA #00** and **STA $C200** to turn off the board in slot 2, for example. You could then select another board in another slot by loading A with the appropriate number of the Eprom you wish to access. You can also disconnect a board by pushing the reset button.

Note: if you need more than 2K, you can make up to 32K available by using a supervisor program to turn on one board and then select one Eprom to allow the 2K of instructions on the Eprom to execute. As long as the Eprom always comes back to the supervisor program, you can run through an entire 32K of machine-language or other higher level languages without having to access a disk drive or change your programming.

How to Assemble the Board

First solder all the sockets to the board for the integrated circuits. Then wire the necessary jumpers on the component side in the location marked J2 (See Figure 5.7).

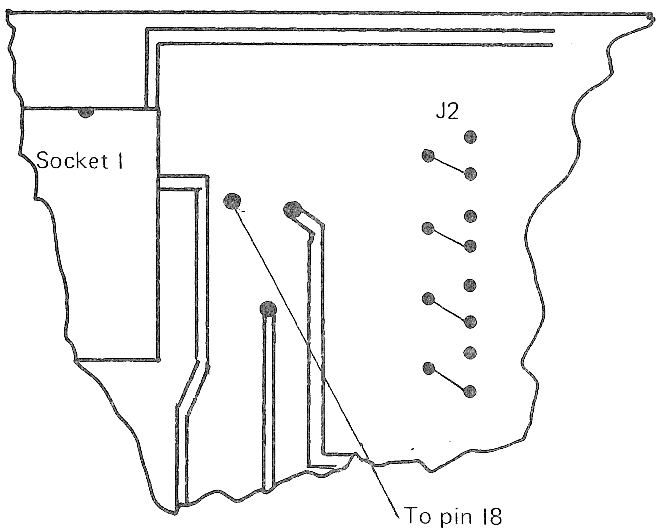


Figure 5.7 Board Assembly

If you also want to use RAM's, place a jumper wire from pin 21 of the EPROM/RAM board to pin 18 of the 50 pin Apple connector. It is also important to note that you must turn the board over to the solder side and cut the trace that leads from pin 24 to pin 21 of the first Eprom socket. Now place the integrated circuits in the appropriate sockets, making sure that pin 1 lines up with the pin 1 on the board and that their noses are all in the same orientation, as shown in Figure 5.4.

6

The Apple Slot Repeater

This chapter describes an Apple computer "slot repeater" project. This will allow you to have your Apple all closed up, yet access the slots within the machine. A perfect example of this would be using the 6522 I/O board while you are trying to design some hardware for the prototype area and don't want to keep looking into the computer or opening it all the time. Your machine can sit there intact, and you can do all the work outside, where there is better light and more freedom (for making measurements and designing your circuits, for example). In order to make the Apple slot repeater card work, it will have to be connected to slot 7 within the computer by a 40-connector cable, which allows you to connect the 40-pin cable coming out of the Apple to a 40 pin-connector socket (dual inline socket) mounted on the repeater board itself. Inside the Apple we recommend using the 50-pin experimenter board, which can be purchased from almost any Apple dealer. The experimenter board has a 50-pin edge-card which fits into the slot, and can be used to solder the wires from the 40-pin cable to the appropriate locations on the edge-card.

Figure 6.1 below shows the wiring sequence for the slot repeater.

Not every line on the Apple bus will be brought out to the repeater board. Among the lines that won't be brought out are the power supply lines, as we wouldn't want to draw too much power from the Apple power supply. The repeater board has pins available for hooking-up an external power supply. The printed circuit board provides the appropriate circuitry for wiring up the sockets required for this slot repeater board, and it has been designed so that the addresses will be the same as they are inside the Apple computer itself. This way the experimenter will find that any experiments he tries will behave the same when he plugs them into the Apple as they do on the repeater board.

The decoding of the addresses mentioned above is performed in the circuit described in the schematic below.

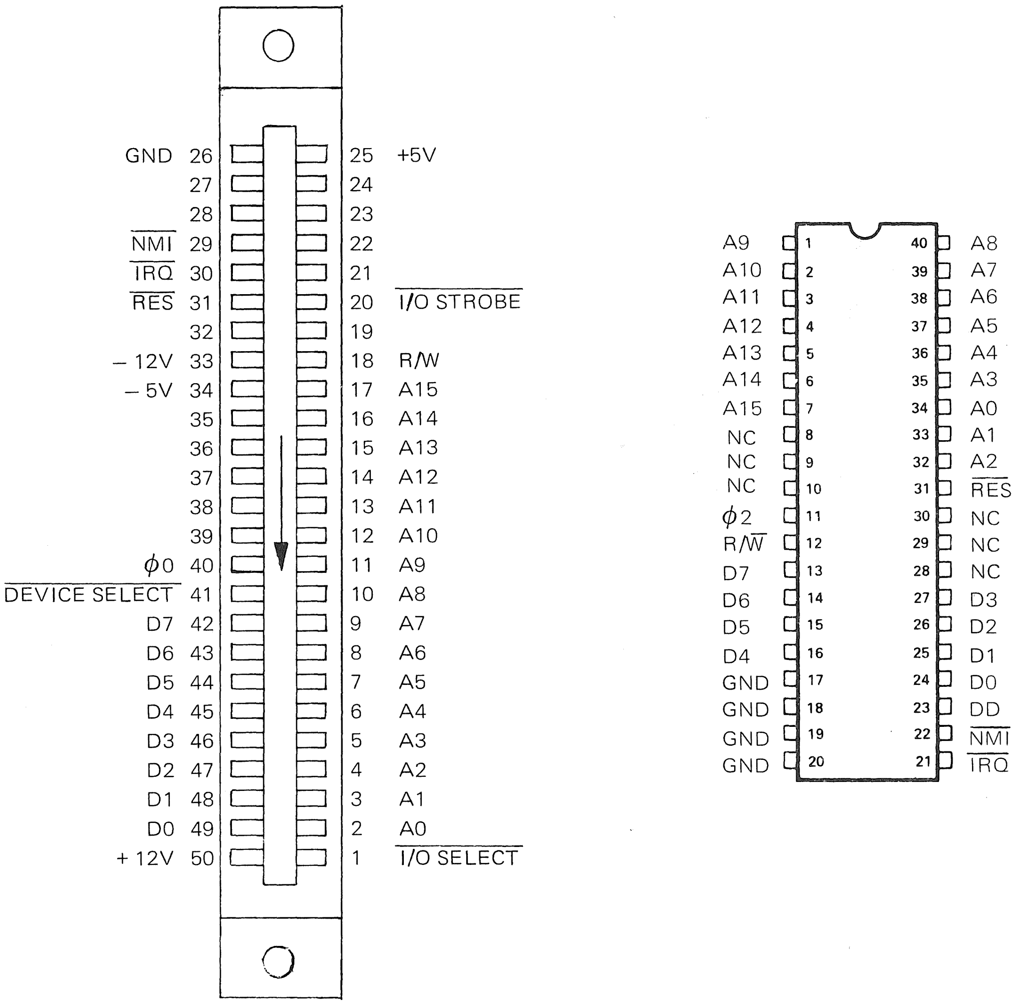


Figure 6.1 Connecting the Repeater Board

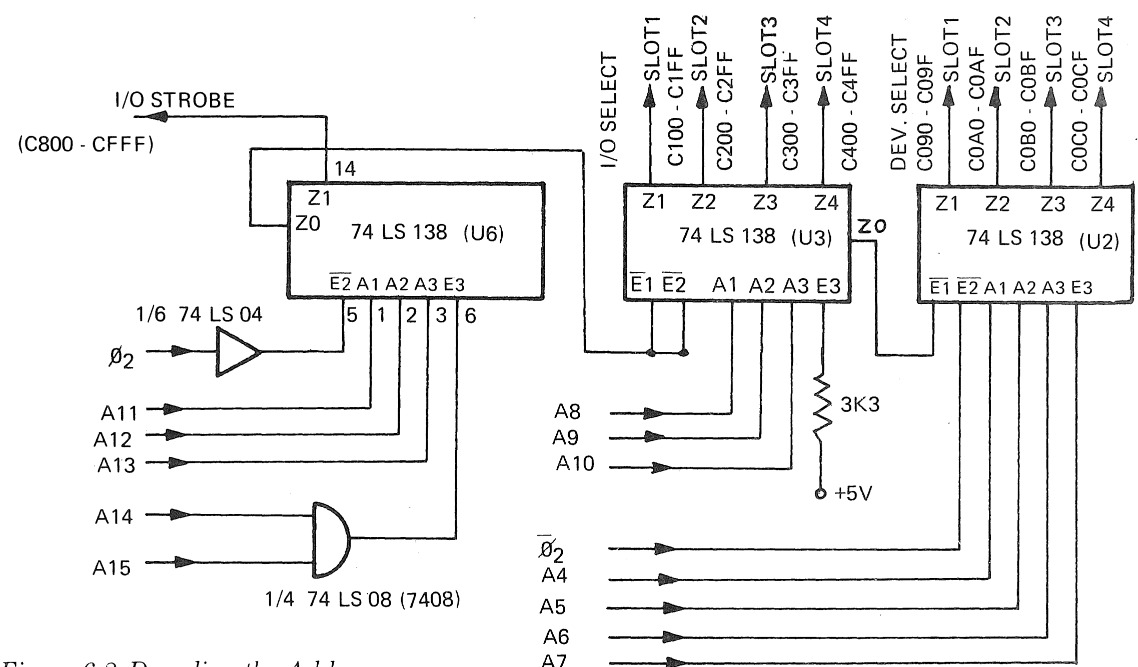


Figure 6.2 Decoding the Addresses

Address lines A11 through A15 are decoded by the 74LS138. This chip generates the select signal and the I/O strobe signal. Pin Z0 enables, with an active low signal, a second 74LS138 which decodes the address lines A8 through A10 and generates the I/O select lines for slots 1 through 4. These are, for example, the addresses **C100** to **C1FF** for the first slot. From this chip a third 74LS138 is enabled from Z0 of the second chip. It decodes the address lines A4 to A7. This creates a device select signal for the first four slots. For example, for slot one it would be **C090** through **C09F**.

The following table will show how to look up all the addresses of the device select, I/O select and the I/O strobe.

Slot	I/O SELECT	DEVICE SELECT	I/O-STROBE
2	C200 – C2FF	C0A0 – C0AF	C800 – CFFF
3	C300 – C3FF	C0B0 – C0BF	C800 – CFFF
4	C400 – C4FF	C0C0 – C0CF	C800 – CFFF
5	C500 – C5FF	C0D0 – C0DF	C800 – CFFF

Figure 6.3

On the memory repeater board, in the upper right-hand corner, there is a place to put in an S44 dual-inline female plug. However, this is for use by other 6502 computers and cannot be used in conjuction with the Apple. Just to the left of that area is a small prototype area for experimenting, or for changes you might want to make with your slot repeater board.

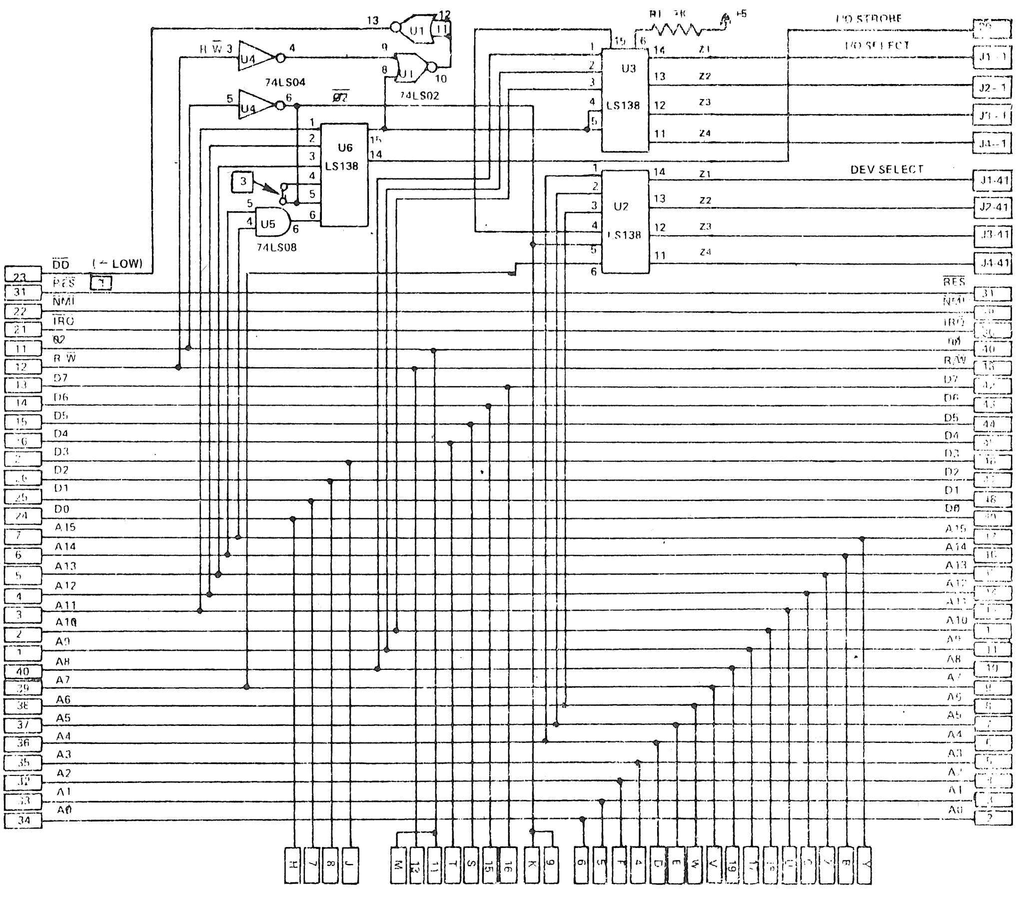


Figure 6.4 The Complete Schematic for the Slot Repeater

How to Assemble the Board

The first step is to mount all the female connectors and the sockets that will be required for the IC's used in this project. Then we connect the pins to the power supply. Next we can put on the capacitor C1, resistor R1, and the 50-pin female connector. The last step is to insert the IC's, making sure that they are lined up in the same manner as they are shown in the schematic (Figure 6.5).

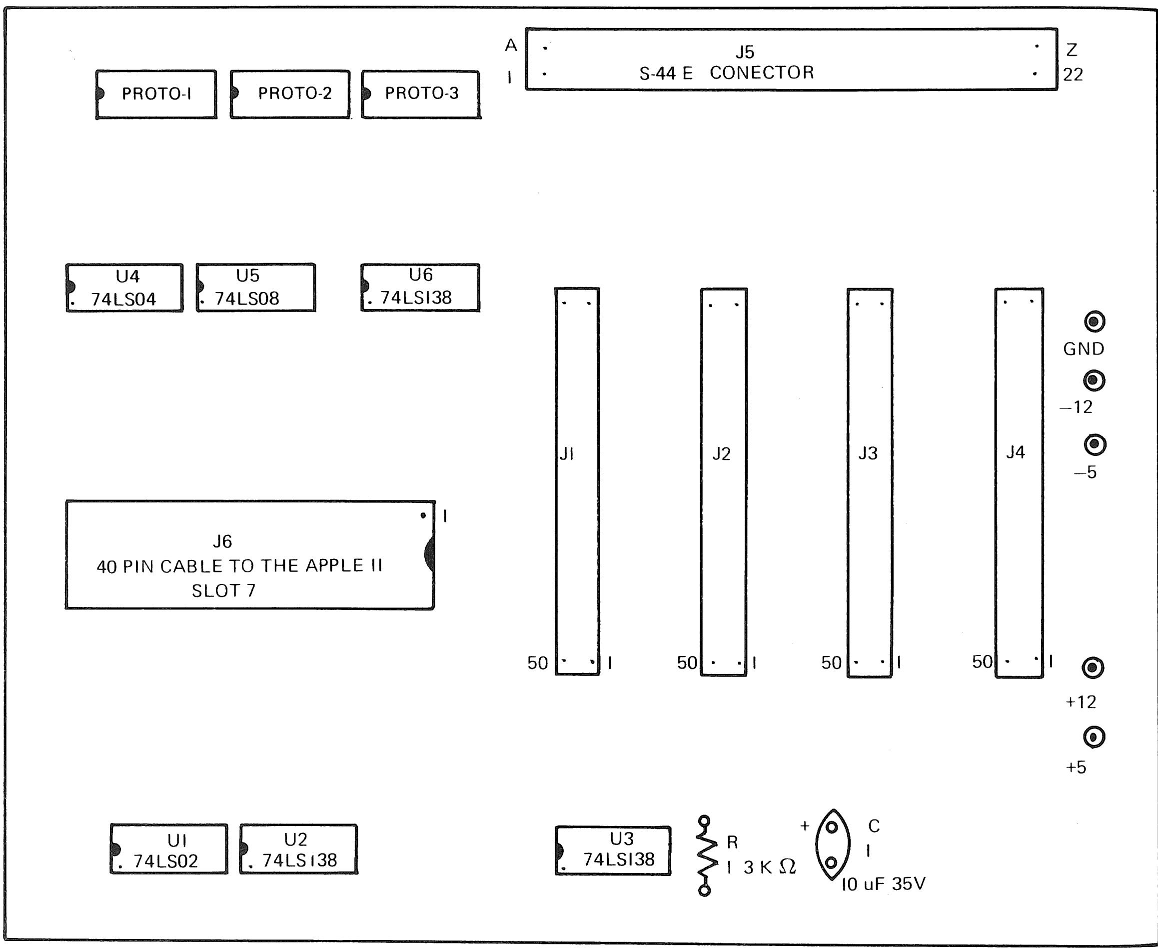


Figure 6.5 Parts Layout

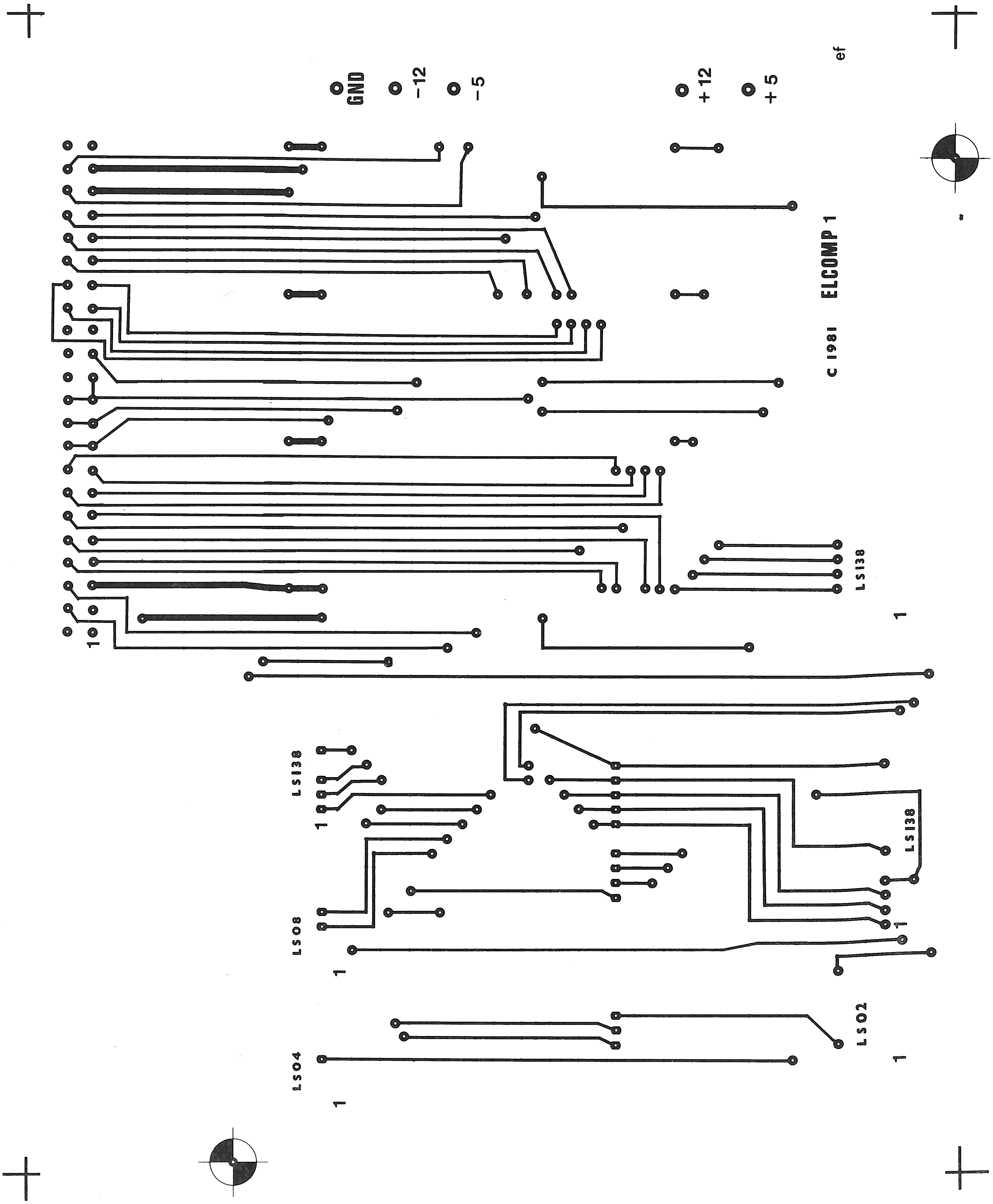


Figure 6.6a Top of the Printed Circuit Board

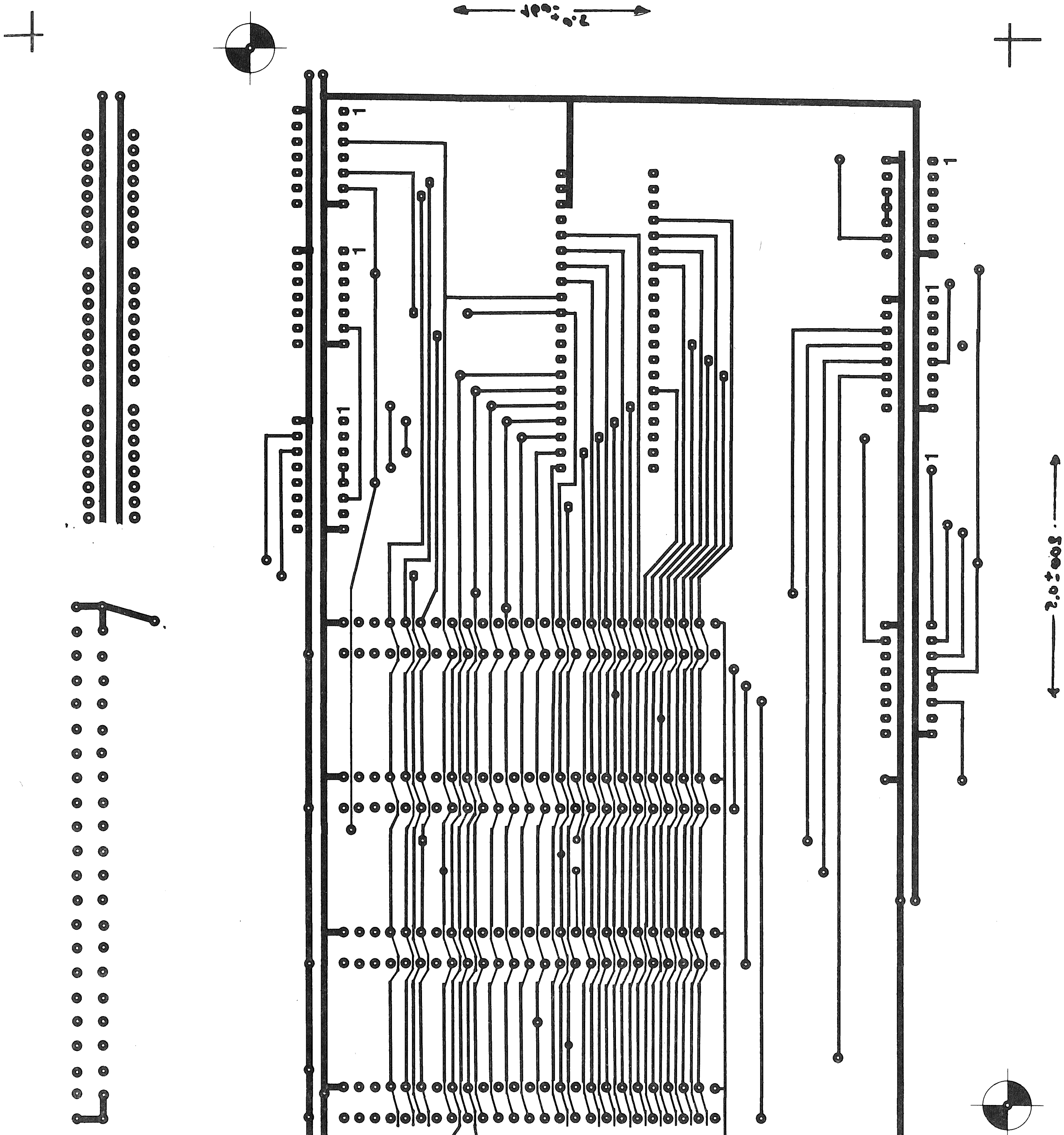


Figure 6.6b Bottom of the Printed Circuit Board

 "Is written on a blank page to avoid confusion" is written on a blank page to avoid confusion . . .

Here is a photograph of a completed board to give you an idea of how it should look if you have assembled it properly.

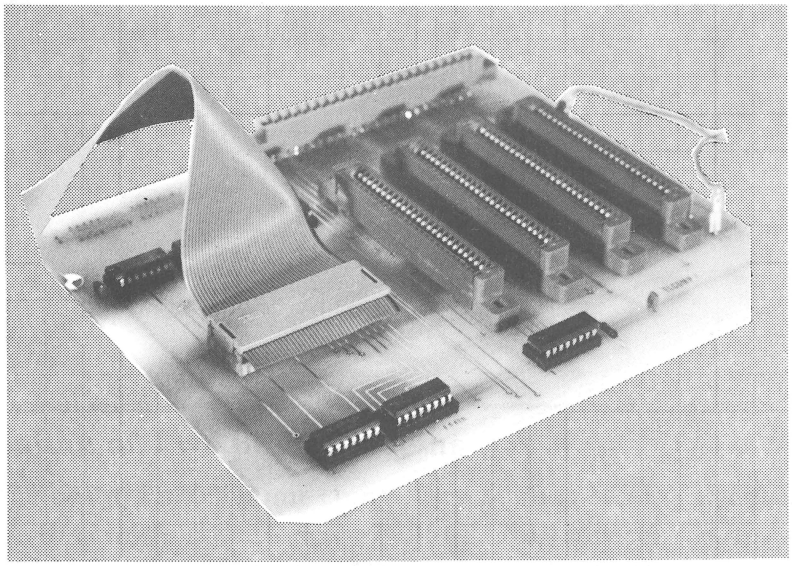


Figure 6.7 Photo of the Completed Board

Figure 6.8 Parts List

Qty	Description
3	14 pin DIL sockets
3	16 pin DIL sockets
1	40 pin DIL socket
1	capacitor 10µF/35V tantal
3	74LS138
1	74LS08
1	74LS02
1	74LS04
4	50 pin edge connectors (available from MOLEX)
1	Resistor 3 k / 0,25W

NOTES

7

The Coupling of Two 6502 Systems

Many of the better known home computers, such as KIM, SYM, AIM, ATARI, PET, APPLE, OHIO, and VIC 20 have a 6502 microprocessor for their CPU. It is sometimes useful to connect two of these systems together to exchange data. This makes the transfer of machine-language programs easier, too.

To define a common interface, we use the 6522 I/O card for each computer. The 6522 card plugs directly into the Apple bus, but to use it with other computers, you'll need the expansion board described in Chapter 6. Figure 7.1 shows the coupling of an Ohio Scientific CIP with an Apple computer, and Fig. 7.2 the program for data exchange.

Program Description

The program in Fig. 7.2 consists of two parts: SEND APPLE —> OHIO and RECEIVE APPLE <— OHIO.

The version shown is for the Apple II computer. Needless to say, the program for the Ohio is exactly the same except for the address of the monitor.

To clarify the use of this program, an example of data transfer from the Apple to the Ohio is presented. In the Apple, the starting address of the data (FROM) and the ending address (UNTIL) are set, and the program is started by **800G**. The Apple then waits in a loop until the Ohio is ready.

In the Ohio, set the address (TO) where the data is to be stored. Then the program is started by jumping to location **842**. The Ohio sends a 1 over PB0 to the Apple, indicating it's ready, which will begin the data transfer. At the end of the data transfer, the Apple jumps to the monitor. The Ohio doesn't know that the Apple has finished, so the receiving program has to be interrupted by pushing the break key of the Ohio.

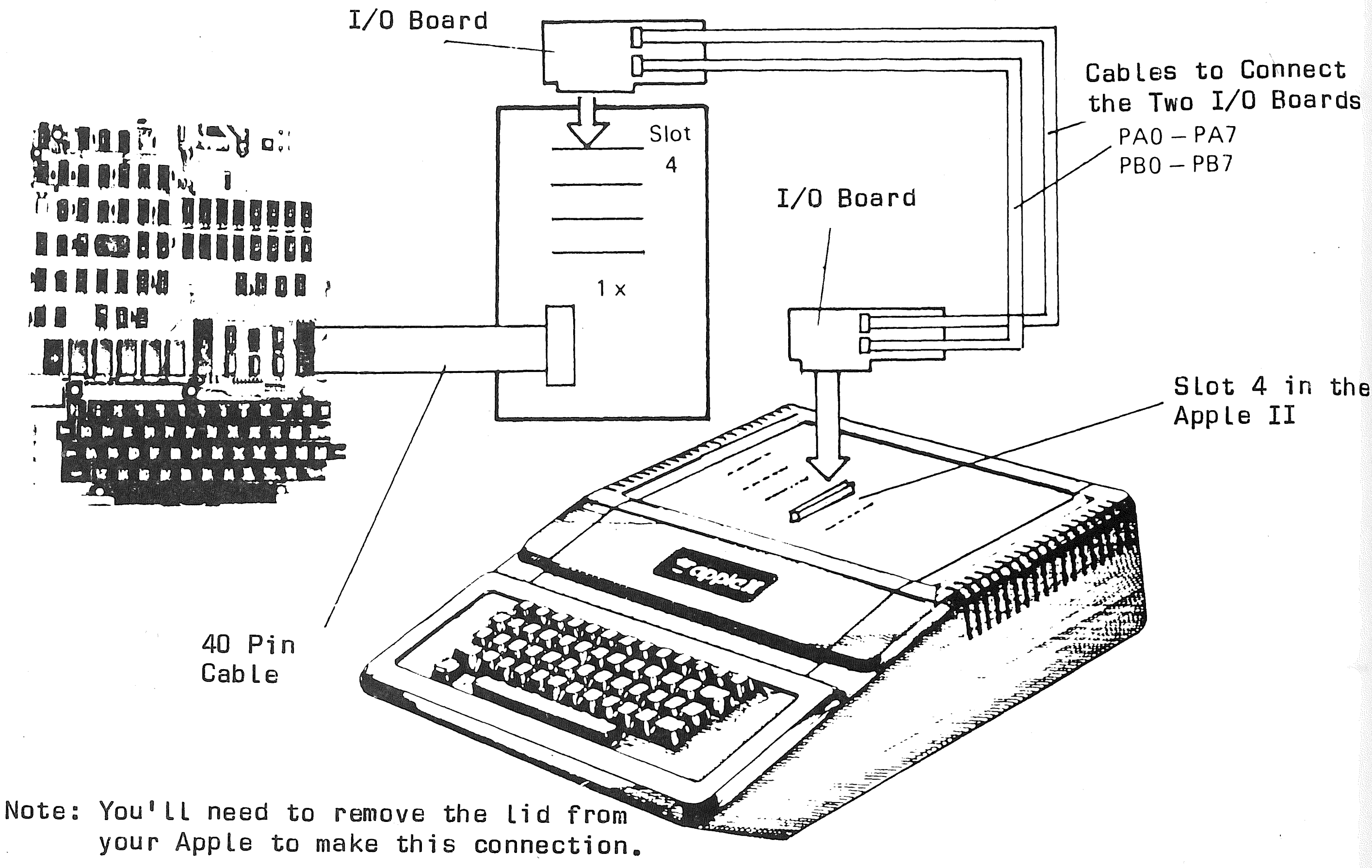


Figure 7.1 Block Diagram

Sending data from the Ohio to the Apple is done in the same manner, but now the Apple performs the receiving program while the Ohio performs the sending program.

This kind of data transfer program is very useful when you are developing programs for single-board computers like the SYM or KIM. The program can be developed and tested on the Apple with one of its powerful assemblers, then sent to a single-board computer without retyping the whole program.

Figure 7.2 Program Listing

```
0800                    DCM "PR#1"
0800         ;SEND APPLE-->OHIO
C0C0                    ORG $C0C0
C0C0         TORB       EQU *
C0C0         TORA       EQU *+!1
C0C0         DDRB       EQU *+!2
C0C0         DDRA       EQU *+!3
C0C0         MONITO     EQU $FF59
C0C0         ;
C0C0         VON        EPZ $10
C0C0         BIS        EPZ $12
C0C0         WOHIN      EPZ $14
C0C0         ;
```

Listing Continued . . .

Continued Listing

```
C0C0            ;
0800                    ORG $800
0800 A000               LDY #$00
0802 A9FF               LDA #$FF
0804 8DC3C0             STA DDRA
0807 ADC0C0     M       LDA TORB
080A 2901               AND #$01
080C D0F9               BNE M
080E B110       M00     LDA (VON),Y
0810 8DC1C0             STA TORA
0813 A980               LDA #$80
0815 8DC2C0             STA DDRB
0818 A900               LDA #$00
081A 8DC0C0             STA TORB
081D EA                 NOP
081E EA                 NOP
081F EA                 NOP
0820 A980               LDA #$80
0822 8DC0C0             STA TORB
0825 E610               INC VON
0827 D002               BNE M10
0829 E611               INC VON+1
082B A511       M10     LDA VON+1
082D C513               CMP BIS+1
082F 90D6               BCC M
0831 F002               BEQ M30
0833 B008               BCS FIN
0835 A510       M30     LDA VON
0837 C512               CMP BIS
0839 F0CC               BEQ M
083B 90CA               BCC M
083D A940       FIN     LDA #$40
083F 4C59FF             JMP MONITO
0842            ;
0842            ;
0842            ;RECIEVE APPLE<--OHIO
0842            ;
0842 A000               LDY #$00
0844 A901               LDA #$01
0846 8DC2C0             STA DDRB
0849 A900               LDA #$00
084B 8DC0C0             STA TORB
084E EA                 NOP
084F EA                 NOP
0850 EA                 NOP
0851 ADC0C0     M1      LDA TORB
0854 2940               AND #$40
0856 F003               BEQ M0
0858 4C59FF             JMP MONITO
085B ADC0C0     M0      LDA TORB
085E 30FB               BMI M0
0860 A901               LDA #$01
0862 8DC0C0             STA TORB
0865 ADC1C0             LDA TORA
0868 9114               STA (WOHIN),Y
086A E614               INC WOHIN
```

Listing Continued . . .

Continued Listing

```
086C D002               BNE M2
086E E615               INC WOHIN+1
0870 A900         M2    LDA #$00
0872 8DC0C0             STA TORB
0875 F0DA               BEQ M1
0877              ;
0877              ;
                        END

        *************************
        *                       *
        * SYMBOL TABLE -- V 1.5 *
        *                       *
        *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

VON     0010  BIS     0012  WOHIN  0014

** ABSOLUTE VARABLES/LABELS

TORB    C0C0  TORA    C0C1  DDRB   C0C2
DDRA    C0C3  MONITO  FF59  M      0807  M00   080E  M10  082B  M30  0835
FIN     083D  M1      0851  M0     085B  M2    0870

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0092

0800- A0 00 A9 FF 8D C3 C0 AD
0808- C0 C0 29 01 D0 F9 B1 10
0810- 8D C1 C0 A9 80 8D C2 C0
0818- A9 00 8D C0 C0 EA EA EA
0820- A9 80 8D C0 C0 E6 10 D0
0828- 02 E6 11 A5 11 C5 13 90
0830- D6 F0 02 B0 08 A5 10 C5
0838- 12 F0 CC 90 CA A9 40 4C
0840- 59 FF A0 00 A9 01 8D C2
0848- C0 A9 00 8D C0 C0 EA EA
0850- EA AD C0 C0 29 40 F0 03
0858- 4C 59 FF AD C0 C0 30 FB
0860- A9 01 8D C0 C0 AD C1 C0
0868- 91 14 E6 14 D0 02 E6 15
0870- A9 00 8D C0 C0 F0 DA CD
*
```

8

Connecting Other Microprocessors to the 6502

In some cases it's very useful to connect circuits of other microprocessor families to the 6502 CPU to use their outstanding performance in an area where the 6502 is weak. For example, there is an 8212 output port in the 80/85 family which is very cheap and has a fanout capacity of 15mA, with a low input load current of 0.25mA. This chip can be used to drive LED's or power transistors. We will discuss the connection of this chip to the 6502, as well as the connection of two other chips: the 8253 (a programmable interval timer) and the 8255 (a programmable peripheral interface).

The 8212 8-bit I/O Port

The connection of the 8212 to the Apple bus is shown in Figure 8.1.

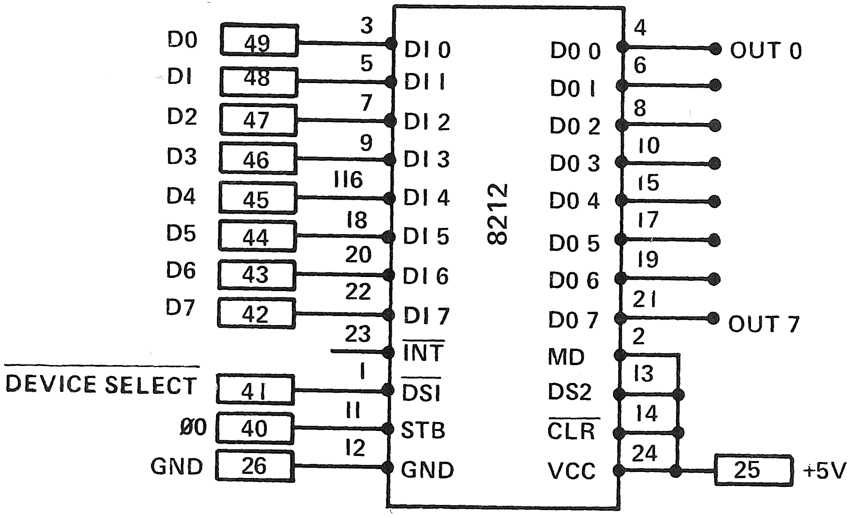


Figure 8.1 Apple Bus Connections

The 8 data lines, D0-D7, of the Apple bus are wired to the data input lines, D10-D17 of the 8212. The chip is selected by the DEV.SEL signal wired to the CSI input. The second chip select input (CS2) is not used and is therefore wired to +5V. The 8212 is used as an output device; therefore, the mode input (MD) is high to enable the output buffers. The clock-pulse for the STB input is the Phi 0 clock from the Apple bus. The output pins of the 8212 (DO0-DO7) are left open.

Outputting data from the Apple to the 8212 is very simple. The 8212 is placed on an experimenter board and put into slot 4. The DEV.SEL addresses are **C0C0** through **C0CF**. A store command to one of these addresses will bring the data to the coresponding output pin, DO0 through DO7.

For example:

LDA #SAA
STA $C0C0

sends the pattern 10101010 on the output pins. In this configuration, no input to the 6502 is possible. There are further restrictions in the use of the 8212. It acts only as an output device, and not like a memory location, as the 6522 does. It only accepts store commands. Other commands, like **INC $C0C0**, will not work with the 8212.

The 8253 Programmable Interval Timer

The 8253 is a programmable interval timer or counter. It consists of 3 independent 16-bit counters. This chip can solve most of the common problems in generating accurate time delays. The timer is set and started by software and can be read by the CPU or by a software interrupt. In the meantime, the CPU is free for other tasks. Figure 8.2 shows the pin configuration and the connection to the Apple bus. The chips of the 80/85 family have separate RD/WR signals, while the 6502 CPU has only one R/$\overline{W}$ signal. With 3 gates of 74LS00, the required RD/WR signals are created from the R/$\overline{W}$ and Phi 0 clock signal.

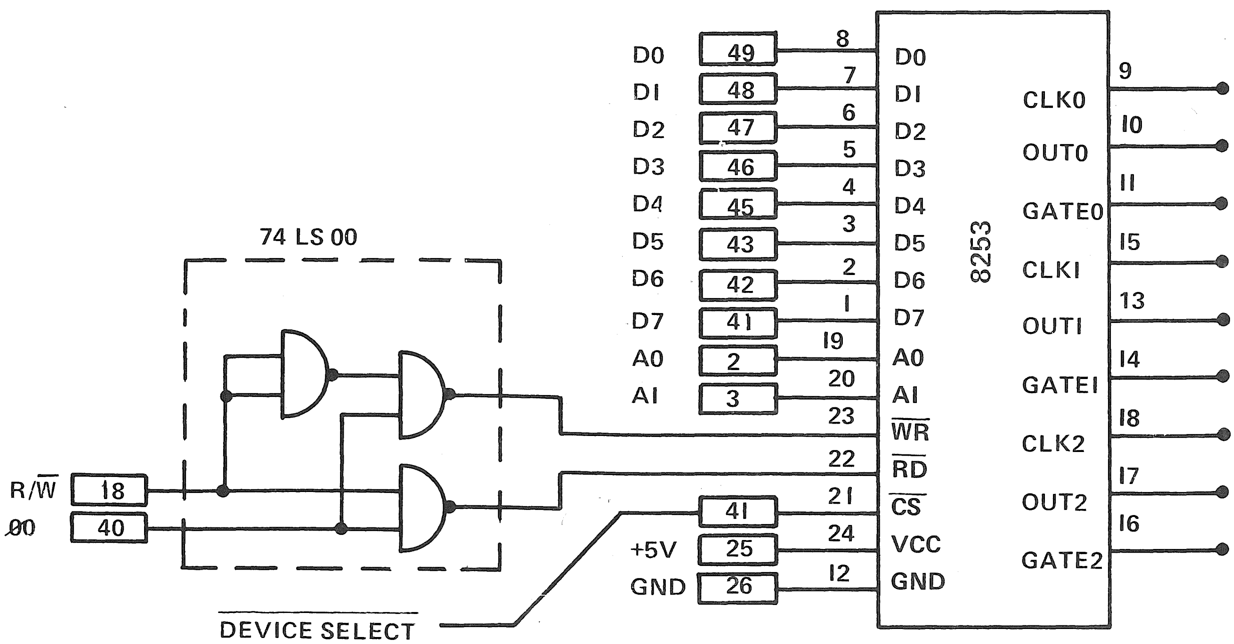


Figure 8.2 8253 Pin Connections

The three counters (0, 1 and 2) are identical in operation, so only one counter module will be discussed here. It consists of a 16-bit, pre-settable down counter, which can operate in either binary or BCD mode. The counter module has two inputs (a clock input and a gate input) and one output. It is controlled by a control word written to the control register. In Figure 8.3 the addressing of the counter and the control register is shown.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	AI	A0	FUNCTION
0	I	0	0	0	LOAD COUNTER 0
0	I	0	0	I	LOAD COUNTER I
0	I	0	I	0	LOAD COUNTER 2
0	I	0	I	I	WRITE CONTROL WORD
0	0	I	0	0	READ COUNTER 0
0	0	I	0	I	READ COUNTER I
0	0	I	I	0	READ COUNTER 2
I	X	X	X	X	DISABLE 3–STATE

Figure 8.3

The format of the control word is shown in Figure 8.4. The desired counter is selected with bits 6 and 7.

BIT	7	6	5	4	3	2	I	0
	SCI	SC0	RLI	RL0	M2	MI	M0	BCD

Figure 8.4 Control Word Format

Bit 7 = 0 Bit 6 = 0
Bit 7 = 0 Bit 6 = 1
Bit 7 = 1 Bit 6 = 0
Bit 7 = 1 Bit 6 = 1

; Select Counter 0
; Select Counter 1
; Select Counter 2
; Not Used

Bits 4 and 5 control the READ/WRITE operation of the counters.

Bit 5 = 0 Bit 4 = 0 Latched Reading
Bit 5 = 1 Bit 4 = 0 Read/Write MSB Only

Listing Continued . . .

Continued Listing

Bit 5 = 0	Bit 4 = 1	Read/Write LSB Only
Bit 5 = 1	Bit 4 = 1	Read/Write (LSB First — Then MSB)

It's often necessary to read a counter on the fly, that is, reading the contents while the counter is still decrementing. To get stable results, a control word with both bit 4 and bit 5 equal to 0 is written to the control register. The contents of the selected counter (set by bit 7 and bit 6) are latched when the write to the control register is done, and can then be transferred into the computer by two consecutive Read operations.

For example:

```
LDA  #$40
STA CTRL
LDA Counter1
STA MEM
LDA Counter1
STA MEM+1

; Select counter 1, latched Read
; Store (A) in the Control Register
; Read LSB first
; Save LSB
; Read MSB Next
; Save MSB
```

After setting the control register to latched reading, there must be two read operations from the selected counter. If the control register is set for reading or writing only one byte (MSB or LSB), there is only one read or write allowed. Otherwise, if it is set for a two-byte read or write, there must be two read or write operations. In the first read/write the least significant byte is transferred to or from the counter/timer, and with the second, the most significant byte will be transferred.

The next three bits of the control word define the operation of the counter. There are 5 different modes.

Bit 3=0	Bit 2=0	Bit 1=0	Mode 0 Interrupt on Count Termination
Bit 3=0	Bit 2=0	Bit 1=1	Mode 1 Programmable One-Shot
Bit 3=X	Bit 2=1	Bit 1=0	Mode 2 Rate generator

Listing Continued . . .

Continued Listing

Bit 3=X	Bit 2=1	Bit 1=1	Mode 3 Square-wave rate generator
Bit 3=1	Bit 2=0	Bit 1=0	Mode 4 Software triggered strobe
Bit 3=1	Bit 2=0	Bit 1=1	Mode 5 Hardware triggered strobe

Mode 0: Interrupt on Count Termination

After setting and starting the counter, the output will be low. It will go high when the counter has reached zero; generating an interrupt. The counter will not stop; it will continue decrementing (or cycling) until it is reloaded. When you write the LSB to the counter, it will stop, and upon writing the MSB it will restart.

Mode 1: Programmable One-Shot

The duration of the one-shot pulse is determined by the value written to the counter. After the rising edge reaches the gate input, the output will go low and stay low until the counter reaches zero. A changing of the stored value during counting will not change the duration of the pulse, but the one-shot can be retriggered by a pulse at the gate input. The mono-flop then starts with the newly defined value.

Mode 2: Rate Generator

The counter acts as a divide-by-N counter. The output will go low for one clock period every time the counter reaches zero. Then the counter is automatically reloaded. Changing the counter value will not affect the present period, but the next period will be use the new value.

Mode 3: Square-wave Rate Generator

This mode is similar to Mode 2 except that the output will remain high for one half of the count duration and will go low for the second half. With even numbers, the counter is decremented by two until it reaches zero. Then the polarity of the output is changed; the counter is reloaded and decremented by two again. With uneven numbers (during the first half period), the counter is first decremented by one, then by two until zero is reached or passed. Then the counter is reloaded, the polarity of the output signal changed, and the counter is decremented first by 3, then by 2 untill zero is reached or passed.

Mode 4: Software Triggered Strobe

After setting the mode, the counter will remain high untill the counter is loaded. Then the counter will start counting down and the output will go low for one clock period at zero-crossing. Reloading the counter will not affect the present period, but will change the next period. A low signal at the gate input will stop the counter. A reload of the counter can be done at this time. After a rising edge reaches the gate input, the counter will start with this new value.

Mode 5: Hardware Triggered Strobe

The counter will start after a rising edge reaches the gate input. The output will go low for one clock period at zero-crossing. The counter is re-triggerable. The output will not go low until a full count after a rising edge at the gate input has occurred.

Additional Information on the 8253

Bit 0 of the control word defines whether a counter acts as a BCD or a binary counter.

Bit 0=0 ; Binary counter (max count: 2 to the 16th)
Bit 0=1 ; BCD counter (max count: 10 to the 4th)

Now we'll examine a demonstration program using counter 0 in the way shown in Figure 8.5.

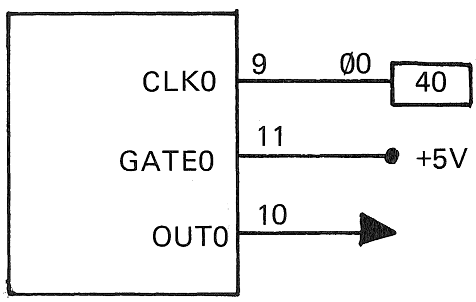


Figure 8.5

For the clock frequency we use the Phi 0 signal at pin 40 of the Apple bus. First we use it as a divide-by-N counter. The following little machine-language program starts the counter. We assume that the 8253 is mounted on an experimenter board plugged into slot 4 of the Apple bus.

The program in Figure 8.6 divides the clock frequency of the Apple by ten and produces negative pulses with a duration of 1 microsecond. Changing the values in memory locations **0806** and **080B** will change the dividing ratio.

The same program produces a square-wave generator when we change the control word to 00110110.

In either case the counting can only be stopped by switching off the computer. The counter can't be stopped by using the reset key.

Figure 8.6 Demo Program

```
0800                     DCM "PR#1"
0800             ;
0800             ;
0800             ;*****************************
0800             ;*                           *
0800             ;*   THE 8253 AS DIVIDE BY N  *
0800             ;*   COUNTER. N=10           *
0800             ;*                           *
0800             ;*****************************
0800             ;
0800             ;
0800             CTRL    EQU $C0C3
0800             COUNT0  EQU $C0C0
0800             ;
0800 A934        DIVIDE  LDA #%00110100      ;CONTROL WORD
0802 8DC3C0              STA CTRL
0805 A50A                LDA $0A
0807 8DC0C0              STA COUNT0          ;STORE LSB FIRST
080A A500                LDA $00
080C 8DC0C0              STA COUNT0          ;STORE MSB
080F 00                  BRK
0810             ;
                         END
```

The 8255 Programmable Peripheral Interface (PPI)

The PPI 8255 is a general purpose I/O device, designed for use with 80/85 microprocessors. But, like the 8112 or the 8253, it can easily be adapted to a 6502 CPU. The device has 24 I/O pins, divided into two groups of 12 I/O pins each. In group A there are 8 pins to Port A and 4 pins to Port B. Group B consists of 8 pins from Port C and 4 pins from Port B. These ports can be used in three different ways:

Mode 0 = Basic input/output
Mode 1 = Strobed input/output
Mode 2 = Bi-directional bus.

By writing a control word to the control register, the mode is set and the input/output definition made.

Figure 8.10 shows the connection of the PPI 8255 to the Apple bus. Instead of using the R/W signal, the RW and the WE signals for the 8255 are created with a 74LS00 NAND gate.

The reset signal of the PPI 8255 is active high; therefore, the RES signal from the 6502 CPU is inverted with the remaining gate of the 74LS04.

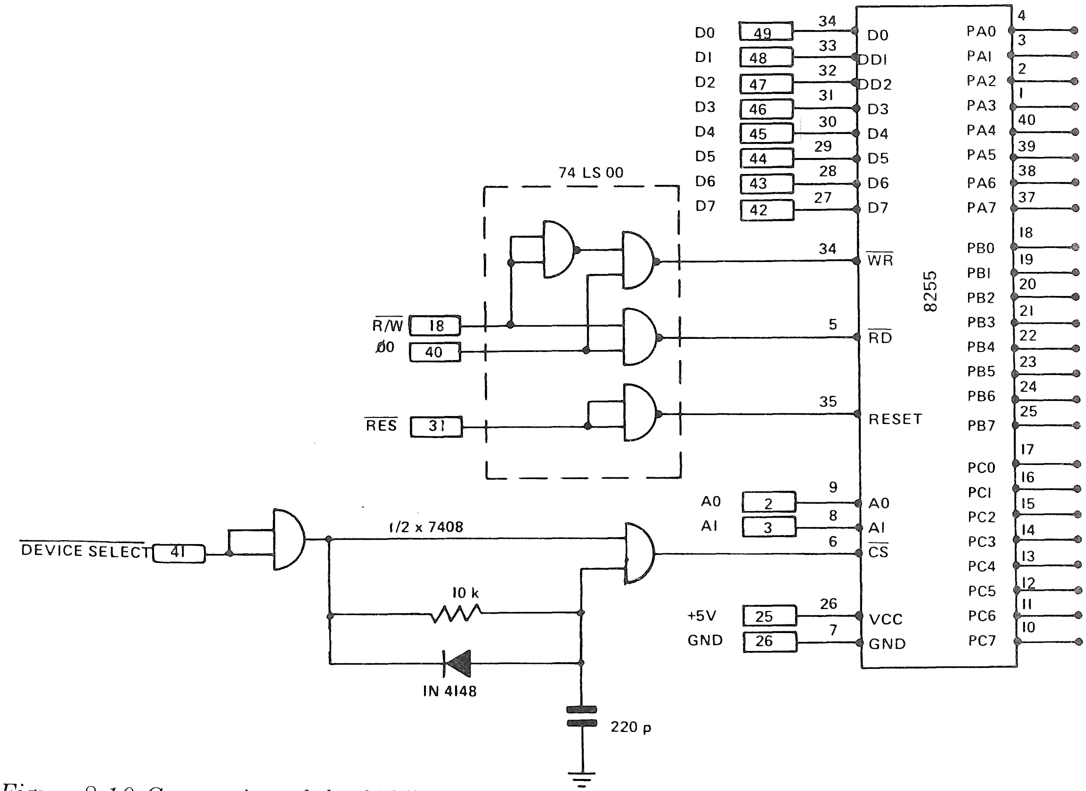


Figure 8.10 Connection of the 8255

There is also another problem. The DEVSEL signal from the APPLE bus has to be made about 50 to 100 nanoseconds longer. This is done with two AND gates and an R/C delay circuit. The diode discharges the capacitor rapidly at the negative pulse, but the positive pulse is delayed by the R/C circuit. This circuit is only necessary with APPLE computers, and not with other 6502 systems.

The addresses of the 3 ports and the control register are shown in Figure 8.11.

$\overline{CS}$	$\overline{WR}$	$\overline{RD}$	A0	A1	FUNCTION
0	1	0	0	0	READ PORT A
0	1	0	0	1	READ PORT B
0	1	0	1	0	READ PORT C
0	0	1	0	0	WRITE PORT A
0	0	1	0	1	WRITE PORT B
0	0	1	1	0	WRITE PORT C
0	0	1	1	1	WRITE CTRL-REG.
1	X	X	X	X	DATA BUS 3–STATE
0	1	0	1	1	ILLEGAL CONDITION
0	1	1	X	X	DATA BUS 3–STATE

Figure 8.11 Port Addressing

After defining an I/O pin as an output, a STORE command can be performed, or when a pin is defined as an input pin, a LOAD command can be performed. Figure 8.12 shows the control word for the mode and input/output definition. The control register is a write only register. A read command from the control register is illegal.

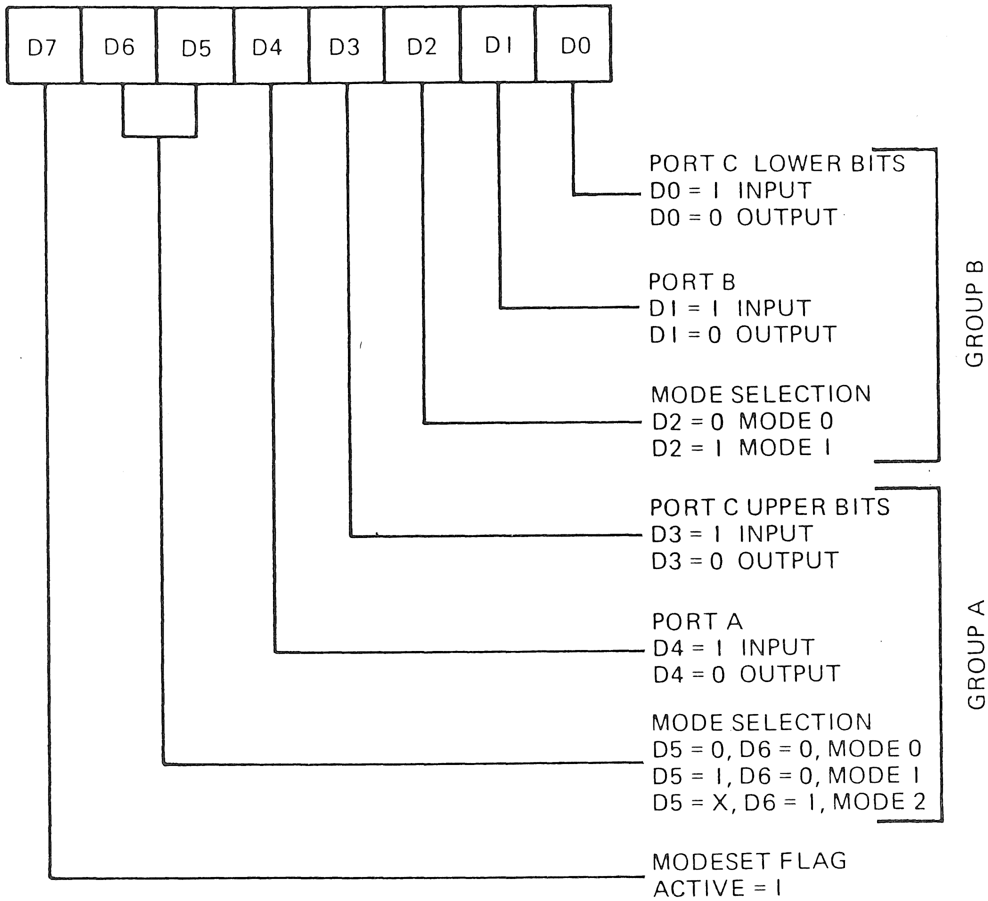


Figure 8.12 Control Word Format

Let's give an example with the PPI mounted on an experimenter board and plugged into slot 4. We will define Port A as an input, and Ports B and C as outputs in the basic input/output Mode 0. Next we have to write the control word (10010000 = **90**) to memory location **C0C3**.

```
LDA #$90
STA $C0C3
```

Now we can load input signals from Port A with the command **LDA $C0C0** and store output signals with **STA $C0C1** or **STX $C0C2** to Port B or Port C.

Figure 8.13 shows the 16 combinations of the control word for Mode 0. For example: The control word for setting all ports to outputs is 10000000 = **80**; or when ports B and C are inputs and Port A is output, then the control word is **8B**.

D7	D6	D5	D4	D3	D2	D1	D0	PORT A	PORT B	PORT CL	PORT CU	HEX
1	0	0	0	0	0	0	0	OUT	OUT	OUT	OUT	80
1	0	0	0	0	0	0	1	OUT	OUT	IN	OUT	81
1	0	0	0	0	0	1	0	OUT	IN	OUT	OUT	82
1	0	0	0	0	0	1	1	OUT	IN	IN	OUT	83
1	0	0	0	1	0	0	0	OUT	OUT	OUT	IN	88
1	0	0	0	1	0	0	1	OUT	OUT	IN	IN	89
1	0	0	0	1	0	1	0	OUT	IN	OUT	IN	8A
1	0	0	0	1	0	1	1	OUT	IN	IN	IN	8B
1	0	0	1	0	0	0	0	IN	OUT	OUT	OUT	90
1	0	0	1	0	0	0	1	IN	OUT	IN	OUT	91
1	0	0	1	0	0	1	0	IN	IN	OUT	OUT	92
1	0	0	1	0	0	1	1	IN	IN	IN	OUT	93
1	0	0	1	1	0	0	0	IN	OUT	OUT	IN	98
1	0	0	1	1	0	0	1	IN	OUT	IN	IN	99
1	0	0	1	1	0	1	0	IN	IN	OUT	IN	9A
1	0	0	1	1	0	1	1	IN	IN	IN	IN	9B

Figure 8.13 Control Word Combinations

Pressing the reset key will set all ports as input ports in Mode 0. Changing the mode of one register will also reset the other ports and the status flip-flops.

There is another feature which can be done with the control word. The output lines of Port C can be set or reset by a single output instruction. The control word for this feature is shown in Figure 8.14.

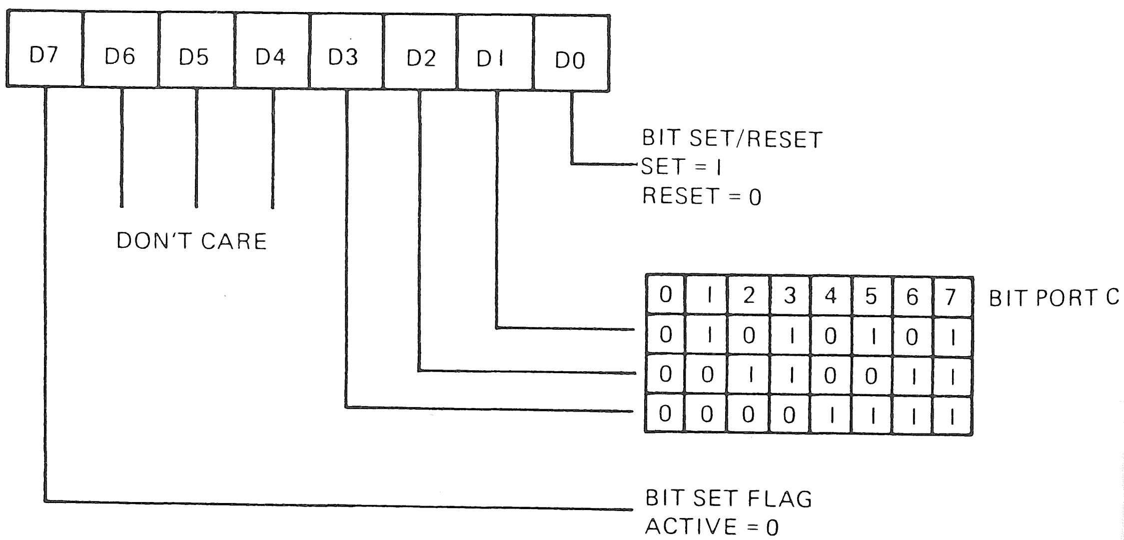


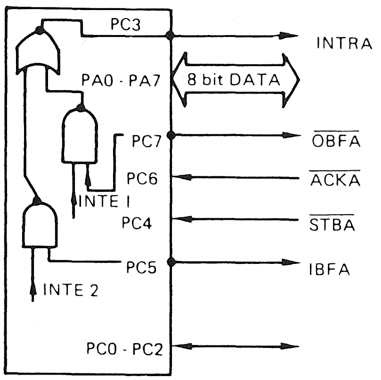
Figure 8.14 Another Control Word Format

The following two instructions set output line 7 of Port C to 1.

LDA %00001111
STA $C0C3

But this pin is only set to one when the port is set as an output port.

In Mode 1, Port A and Port B can be used as input or output ports. The four-bit ports, CL and CU, are used for control functions and for the status of the 8-bit ports. An input control signal definition for Port A is shown in Figure 8.15



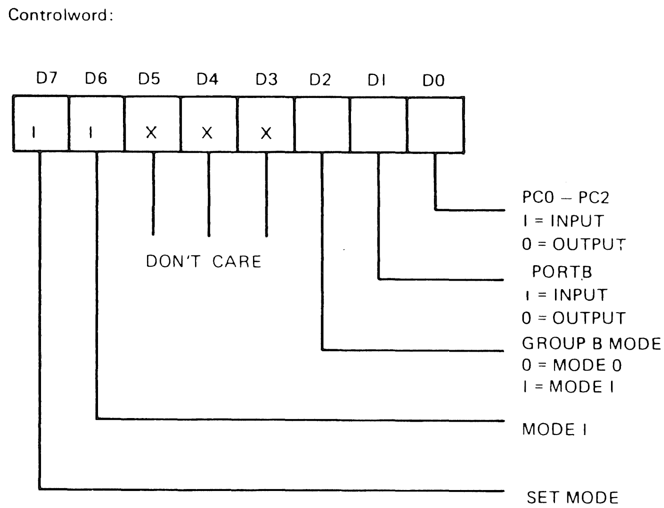


Figure 8.15 Control Signal Definition

For this case the control word is **B8**. A low input signal at STBA loads the data into the input latch of Port A. This is indicated to the CPU by a high going signal at IBF (Input Buffer Full). If a read command occurs at Port A, the IBFA signal is reset. Figure 8.16 is a little demonstration program. The IBFA (PB4) signal is connected to pin 6 (PC6). This pin is programmed as an input. PC4 is normally high.

Figure 8.16 Demo Program

```
0800                      DCM "PR#1"
0800             ;
0800             ;
0800             ;*****************************
0800             ;*                           *
0800             ;* DATAINPUT VIA PORTA       *
0800             ;* PPI 8255      MODE 1      *
0800             ;*                           *
0800             ;*****************************
0800             ;
0800             ;
0800             PORTA   EQU $C0C0
0800             PORTC   EQU $C0C2
0800             CTRL    EQU $C0C3
0800             MEM     EQU $1000
0800             ;
0800             ;
0800 A9B8        DATIN   LDA #$B8          ; MODE 1
0802 8DC3C0              STA CTRL
```

Listing Continued . . .

Continued Listing

```
0805 ADC2C0      M        LDA PORTC
0808 2940                 AND #%01000000
080A F0F9                 BEQ M
080C ADC0C0               LDA PORTA
080F 8D0010               STA MEM
0812 60                   RTS
0813             ;
                          END
```

After setting the mode, the program reads pin PC6 on Port C. As long as this pin is zero, the program stays in the loop. When data is stored in the output latches by a negative pulse on STBA (PC4), the program reads Port A and stores the contents in memory location MEM.

In this demonstration program the computer waits in a loop until a data ready signal is received. During this time the computer won't do anything else. To avoid this problem, use the interrupt technique. The PPI 8255 performs this with an internal INTE (Interrupt Enable) flip-flop.

The signal INTEA goes high when the following condition occurs: $\overline{STB} = 1$, AND IBFA = 1 AND INTEA = 1. The setting or resetting of the INTEA flip-flop is controlled by setting or resetting PC4 by program. This does not affect the STB pulse. Figure 8.17 is a listing of a program routine which enables the interrupting of the processor. The starting address of INTE must be stored in **03FB** and **03FC**.

Figure 8.17 Interrupt Demo Program

```
0800                          DCM "PR#1"
0800             ;
0800             ;
0800             ;****************************
0800             ;*                          *
0800             ;*   INTERUPTING THE 6502 BY *
0800             ;*   THE 8255 PPI            *
0800             ;*                          *
0800             ;****************************
0800             ;
0800             ;
0800             PORTA   EQU $C0C0
0800             CTRL    EQU $C0C3
0800             MEM     EQU $1000
0800             AWAY    EQU $2000
0800             ;
0800 A9B8        INT     LDA #$B8
0802 8DC3C0              STA CTRL            ;SET MODE1,PORTA INPUT
0805 A909                LDA #$09            ;SET BIT 4,PORT C
0807 8DC3C0              STA CTRL            ;ENABLE INTERUPT
080A 4C0020              JMP AWAY
080D             ;
080D             ;
080D ADC0C0      INTE    LDA PORTA
0810 8D0010              STA MEM
0813 40                  RTI
0814             ;
0814             ;
                         END
```

When this signal is used with the 6502 processor, the polarity of the interrupt must be changed because the 6502's interrupt signal is active low. First Mode 1 is set; then bit 4 of Port C is set, which enables the INTEA flip-flop. Then the program jumps to another routine. When an interrupt occurs, the program jumps to the address derived from the interrupt vector, labeled (INTE), loads the contents of Port A, and stores that value in memory location MEM. It then returns to its previous task by an RTI. Port B can be controlled in the same manner. The STB pin for Port B is PC2, the IBFB is PC1 and the interrupt signal INTRB is PC0. The INTEB flip-flop is controlled by setting or resetting PC2.

Figure 8.18 shows the Mode 1 output control signal definition. The $\overline{OBF}$ (Output Buffer Full) signal will go low when the CPU has performed a store instruction to Port A.

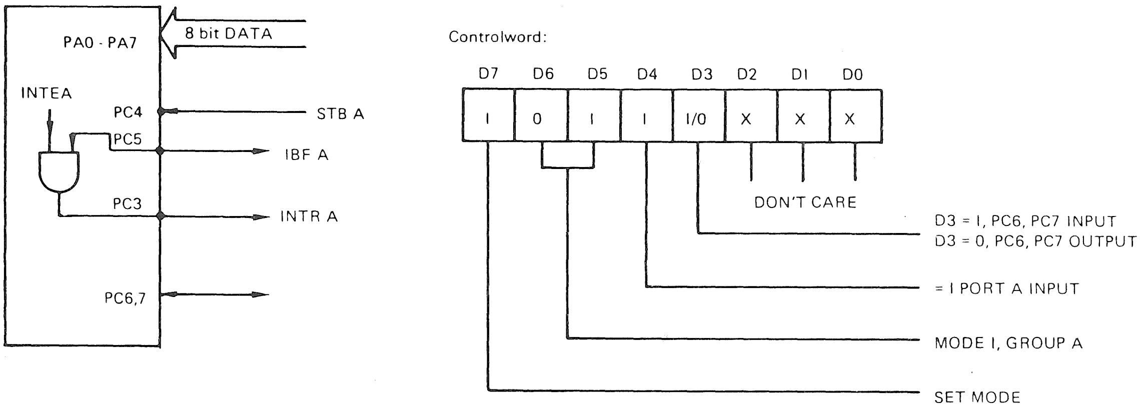


Figure 8.18 Output Signal Definition

$\overline{OBF}$ is reset by an acknowledge input. This is a low going pulse, which informs the CPU that the peripheral device has received the data. The interrupt request line INTRA goes high when the INTEA flip-flop is enabled and $\overline{OBF}$ = 1 and ACK = 1, thus indicating that the peripheral system has taken the data.

The interrupt enable flip-flop INTEA is controlled by setting or resetting PC6.

Figure 8.19 Data Output Program

```
0800            DCM "PR#1"
0800      ;
0800      ;
0800      ;*****************************
0800      ;*                           *
0800      ;*  DATA OUTPUT VIA PORTA    *
0800      ;*  PPI 8255     MODE 1      *
0800      ;*                           *
0800      ;*****************************
0800      ;
0800      ;
0800      PORTA   EQU $C0C0
0800      PORTC   EQU $C0C2
0800      CTRL    EQU $C0C3
0800      MEM     EQU $1000
```

Listing Continued . . .

Continued Listing

```
0800                ;
0800                ;
0800 A9A0           DATOUT LDA #$A0          ;MODE 1
0802 8DC3C0                STA CTRL
0805 AD0010                LDA MEM
0808 8DC0C0                STA PORTA
080B ADC2C0         M      LDA PORTC
080E 2920                  AND #%0100000
0810 D0F9                  BNE M
0812 60                    RTS
0813                ;
                           END
```

There are no restrictions for using both groups A and B. Group A (Port A and the upper part of Port C) can be programmed either as an input or an output. Likewise, group B (Port B and lower half of Port C) can be programmed as an input or an output, independent of the programming of Port A. Mode 2 combines the input control definition and the output control definition on Port A only. This port acts as a bi-directional input/output port controlled by bit 3 through bit 7 of Port C, as shown in Figure 8.20.

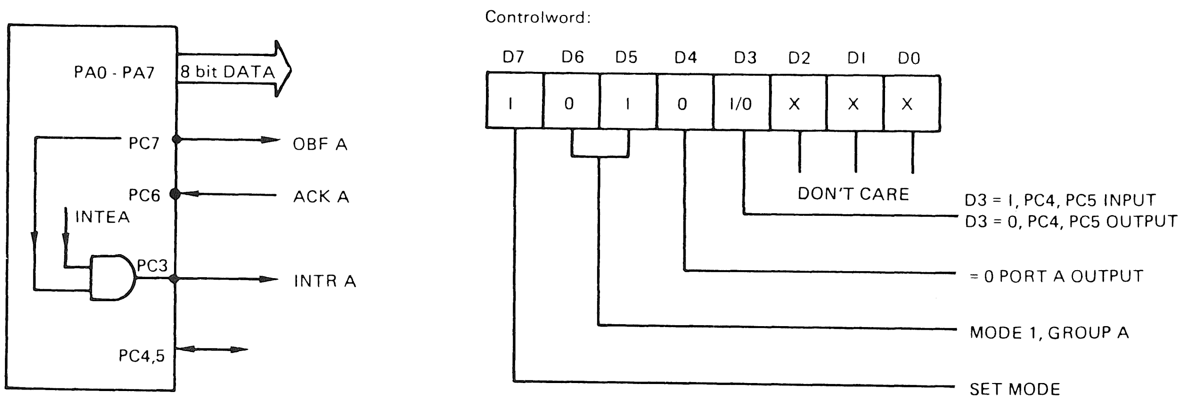


Figure 8.20 Control Definition

The input/output port is normally in tri-state. The data for input/output is strobed by the STBA or the ACKA signal. A low signal on the ACKA enables the tri-state buffers on Port A to send out data. A high signal on ACKA will put the buffer in a high impedance state.

A low signal on the STBA will load data into the input buffers. The IBFA and the OBFA signal act in the same manner as described in the input or output control definition. Likewise, the input interrupt (INTE2) is controlled by setting or resetting PC4, and the output interrupt (INTE1) is controlled by PC6.

The Peripheral Interface Adapter PIA 6821

The 6821 is a universal interface chip which provides two bi-directional ports, A and B; two control registers; and four interrupt lines, two of which are usable as peripheral output controls. Adapting to a 6502 system is very easy, because it has the same pin-out as the PIA 6520. The connection to the Apple bus is shown in Figure 8.21

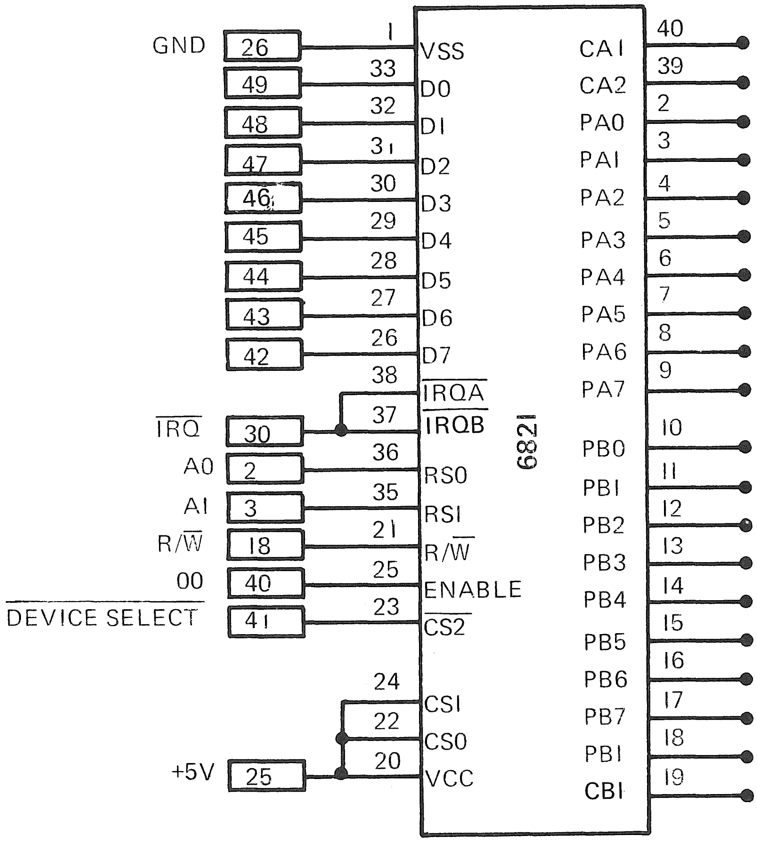


Figure 8.21 Pin Configuration of the 6821

The data direction for the two ports, A and B, is set by two data direction registers DDRA and DDRB, and controlled by the two control registers CRA and CRB. For these six registers there exist only two address lines, RS0 and RS1. Therefore, the ports and the data direction registers have the same address. Bit two in the control register determines which one of the two registers is accessed. Figure 8.23 shows the internal addressing of the 6821.

If bit two of the control register is a one, the port is accessed; if it is a zero, the data direction register is accessed. The following program in Figure 8.24 sets all lines of Port A to an output. For this program we assume that the 6821 is mounted on an experiment board and plugged into slot 4 of the Apple bus.

The next figure (8.25) shows the format of the control word in the two control registers CRA and CRB. Bits 0 through 5 can be set or reset by the CPU; bits 6 and 7 are read-only bits and are modified by external pulses at the CA1, CA2, CB1 and CB2 inputs. Bits 0 and 1 of CRA, or CRB, determine whether an interrupt occurs

		BIT 2 OF		SELECTED
RS0	RS1	CRA	CRB	REGISTER
0	0	1	—	PORTA
0	0	0	—	DDRA
0	1	—	—	CRA
1	0	—	1	PORTB
1	0	—	0	DDRB
1	0	—	—	CRB

Figure 8.23 Internal Addressing of the 6821

at IRQA, or IRQB, respectively, or signal a "no interrupt" condition. For example, if bit 0 is 1, and bit 1 is 0, a negative transition will set bit 7 to 0, causing an interrupt at IR. The four possibilities are shown in Figure 8.26 and are the same for both control registers.

Figure 8.24 Setting Port A to Output

```
0800                        DCM "PR#1"
0800                 ;
0800                 ;
0800                 ;****************************
0800                 ;*                          *
0800                 ;*   SETTING ALL PINS OF    *
0800                 ;*   PORT A FOR OUTPUT      *
0800                 ;*                          *
0800                 ;****************************
0800                 ;
0800                 ;
0800                 PORTA   EQU $C0C0
0800                 CRA     EQU $C0C1
0800                 ;
0800 A900            OUTPUT  LDA #$00          ;SELECT DATA
0802 8DC1C0                  STA CRA           ;DIRECTION REGISTER
0805 A9FF                    LDA #$FF          ; SELECT ALL PINS
0807 8DC0C0                  STA PORTA         ;FOR OUTPUT
080A A904                    LDA #$04          ;SELECT POTRA
080C 8DC1C0                  STA CRA
080F                 ;
080F A9AA                    LDA #$AA          ;BIT PATTERN
0811 8DC0C0                  STA PORTA         ;STORED IN PORTA
0814 60                      RTS
0815                 ;
                             END
```

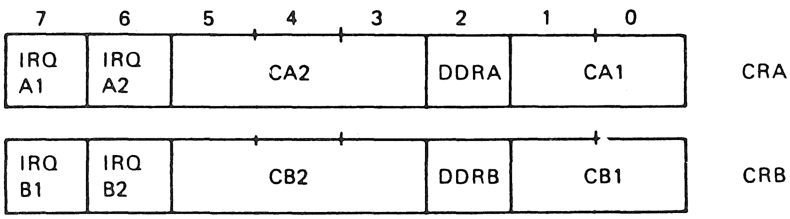


Figure 8.25 Control Word Format

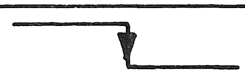
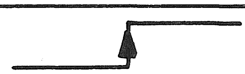
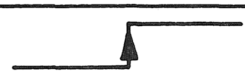
CRAI (CRBI)	CRA0 (CRB0)	INPUT AT CAI (CBI)	IRQA (IRQB)
0	0		NO INTERRUPT
0	I		BIT 7 = 0 INTERRUPT
I	0		NO INTERRUPT
I	I		BIT 7 = 0 INTERRUPT

Figure 8.26 Control Interrupt Modes

Bits 3, 4 and 5 control the interrupt lines, CA2 and CB2. If bit 5 is 0, both control registers perform the same function. Figure 8.27 shows the interrupt handling using CA2 or CB2, respectively. If CRA5 = 0, CRA4 = 1, and CRA3 = 0, a positive transition on CA2 will not cause an interrupt.

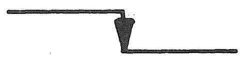
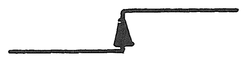
CRA5 (CRB5)	CRA4 (CRB4)	CRA3 (CRB3)	INPUT AT CA2 (CB2)	IRQA (IRQB)
0	0	0		NO INTERRUPT
0	0	I		BIT 6 = 0 INTERRUPT
0	I	0		NO INTERRUPT
0	I	I		BIT 6 = 0 INTERRUPT

Figure 8.27 Interrupt Handling

If bit 5 is set to 1, the control registers (CRA and CRB) have different functions. The control register (CRA) uses both input/output lines (CA1 and CA2) to perform handshaking while reading; the control register (CRB) performs handshaking while writing. In both cases, the lines CA2 and CB2 are outputs. The handshaking modes for reading are shown in Figure 8.28.

CRA5	CRA4	CRA3	MODE	FUNCTION
I	0	0	HANDSHAKE READ	CA2 = INTERRUPT ON CAI CA2 = 0 AFTER LOAD
I	0	I	PULSE	CA2 =  AFTER LOAD
I	I	0		CA2 = 0
I	I	I		CA 2 = I

Figure 8.28 Reading Handshaking Modes

With CRA3 = 0, CRA4 = 0, and CRA5 = 1, the output line is set to 1, and an interrupt occurs at CA1. CA2 is reset after a LOAD instruction. If CRA3 = 3, a pulse of one machine cycle is created after a load instruction.

CA2 can be set to zero with CRA4 = 1 and CRA3 = 0, or set to one with CRA4 = 1 and CRA3 = 1.

Figure 8.29 shows the handshaking modes while writing with CB2. With CRB5 = 1, CRB4 = 0, and CRB3 = 0, CB2 is set to zero after a store instruction.

CRB5	CRB4	CRB3	MODE	FUNCTION
I	0	0	HANDSHAKE WRITING	CB2 = 0 AFTER STORE CB2 = I AT INTERRUPT CBI
I	0	I	PULSE	CB2 = ‾___‾ AFTER STORE
I	I	0		CB2 = 0
I	I	I		CB2 = I

Figure 8.29 Writing Handshaking Modes

When an interrupt occurs on CB1, CA1 is set to 1. If CRB4 = 0 and CRB3 = 1, a pulse of one machine cycle is created after a store instruction. CB2 can be set to zero with CRB4 = 1 and CRB3 = 0, and CB2 can be set to one with CRB4 = 1 and CRB3 = 1.

The APPLE as a Logic Tester

The following program is an example of how the Apple could be used as a logic tester, or to demonstrate the operation of an integrated circuit.

In this example we'll be using the 74LS190. This is a decimal up and down counter that used parallel I/O. The wiring diagram for connecting it to the 6821 is shown in Fig. 8.30.

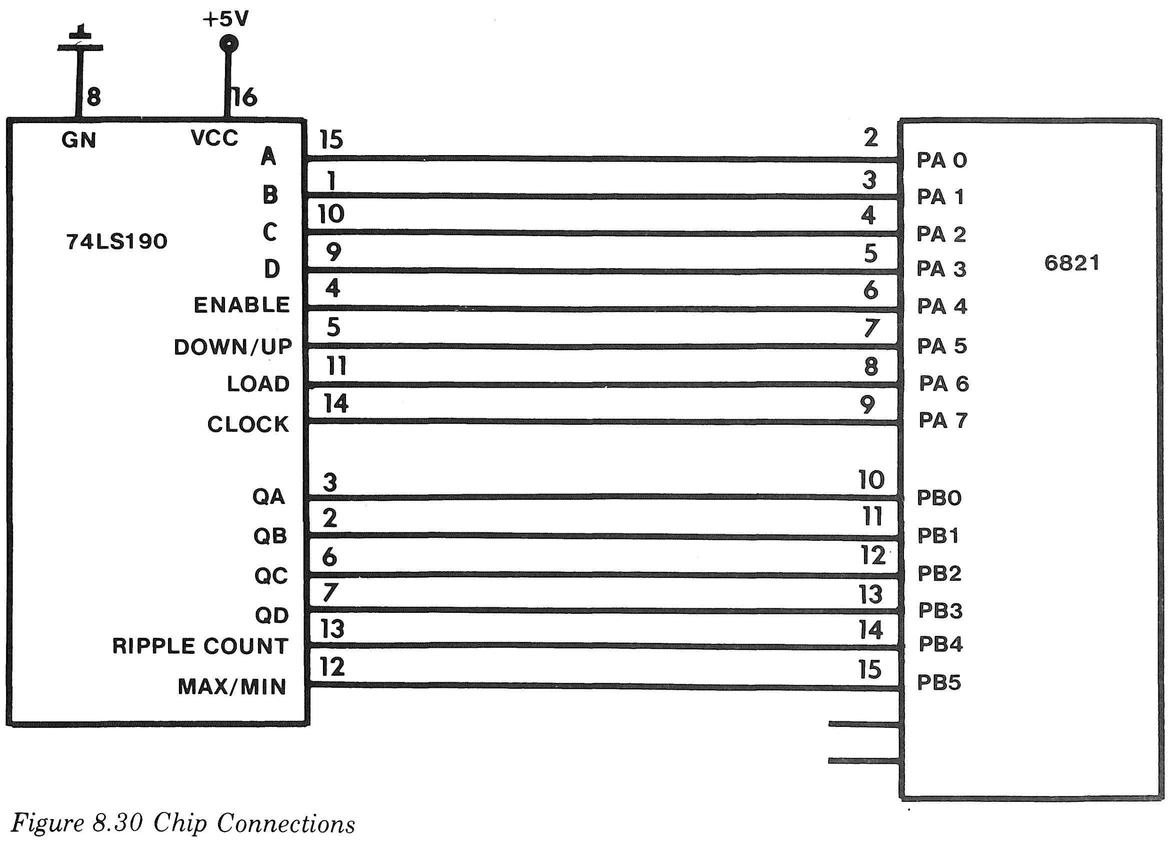


Figure 8.30 Chip Connections

All input pins of the 74LS190 are connected to Port A of the 6821, and the output pins are connected to Port B. The connections from the 6821 to the Apple bus were shown earlier in Fig. 8.21.

The program in Fig. 8.31 demonstrates the behavior (logic) of the counter. It uses the following subroutines:

INIT – initializes the 6821. Port B is set to the input mode, and Port A is set to the output mode. A bit pattern (11010000) is stored in Port A. For the 74LS190 this means that LOAD, CLOCK and ENABLE inputs are high, the data inputs A, B, C, D and the DOWN/UP input are low.

STATE – reads the output pins of the 74LS190 and displays them on the screen. For a low output an (L) will be displayed, and for a high output an (H) will be displayed. The outputs QA, QB, QC, QD, RC and MAX/MIN are displayed from left to right.

CLOCK – creates one clock pulse and LOAD creates one load pulse for the counter. These are negative-going pulses.

BSET – sets the input pins A, B, C and D. After a LOAD command, this state is transferred to the counter.

USER – is the main program entry point. First the message (ENTER:) is printed on the screen. If you enter an (E), the computer responds with (E=) and you may input a value for (E). Entering an (L) enables the 74LS190, entering (H) disables it, and the state of the output pins are shown. Pressing (C) causes the counter to count one pulse. (U) sets the counter to the UP mode, and (D) sets the counter to DOWN. With (S), the parallel inputs may be set. The computer responds with (ABCD=) and you can enter a combination of (H)'s and (L)'s. Once all four bits have been entered, the pattern is transferred to the counter by typing (L). Hitting any other key causes the program to jump to the machine-language monitor. This program is very specialized, but other programs for testing and demonstrating digital circuits can be written to adapt this circuit to your application.

Figure 8.31 Demo Program

```
0800              DCM "PR#1"
0800        ;
0800        DUMMY   EQU $1000
0800        OUTCH   EQU $FDED
0800        RDCHR   EQU $FD35
0800        HOME    EQU $FC58
0800        CR      EQU $FD8E
0800        BEEP    EQU $FF3A
0800        ;
0800        PORTA   EQU $C0C0          ; SLOT 4
0800        PORTB   EQU PORTA+2
0800        CTRLA   EQU PORTA+1
0800        CTRLB   EQU PORTA+3
0800        ;
0800                ORG $800
```

Listing Continued . . .

Continued Listing

```
0800               ;
0800 4CBB08               JMP USER
0803 BA            TXTOUT TSX             ; OUTPUT OF TEXT
0804 E8                   INX             ; TEXT MUST FOLLOW THE
0805 BD0001               LDA $100,X      ; SUBROUTINE CALL
0808 8D1B08               STA ADR+1       ; ENDING WITH HEX 00
080B E8                   INX
080C BD0001               LDA $100,X
080F 8D1C08               STA ADR+2
0812 EE1B08               INC ADR+1
0815 D003                 BNE ADR
0817 EE1C08               INC ADR+2
081A AD0010        ADR    LDA DUMMY
081D F00D                 BEQ M
081F 20EDFD               JSR OUTCH
0822 EE1B08               INC ADR+1
0825 D0F3                 BNE ADR
0827 EE1C08               INC ADR+2
082A D0EE                 BNE ADR
082C AD1C08        M      LDA ADR+2
082F 48                   PHA
0830 AD1B08               LDA ADR+1
0833 48                   PHA
0834 60                   RTS
0835               ;
0835 A900          INIT   LDA #00         ; SELECT DATA
0837 8DC1C0               STA CTRLA       ; DIRECTION REG
083A 8DC3C0               STA CTRLB
083D 8DC2C0               STA PORTB       ; PORT B INPUT
0840 A9FF                 LDA #$FF
0842 8DC0C0               STA PORTA       ; PORT A OUTPUT
0845 A904                 LDA #$04        ; SELECT
0847 8DC1C0               STA CTRLA
084A 8DC3C0               STA CTRLB
084D               ;
084D A9D0                 LDA #$D0
084F 8D7808               STA MASK
0852 8DC0C0               STA PORTA
0855 60                   RTS
0856               ;
0856 ADC2C0        STATE  LDA PORTB       ; OUTPUT THE STATE
0859 8D7708               STA ASAVE       ; OF THE 74190
085C A208                 LDX #$08        ; QA,QB,QC,QD,RC,MAX/MIN
085E 6E7708        S0     ROR ASAVE
0861 B004                 BCS S1
0863 A9CC                 LDA #"L"
0865 9002                 BCC S2
0867 A9C8          S1     LDA #"H"
0869 8E7608        S2     STX XSAVE
086C 20EDFD               JSR OUTCH
086F AE7608               LDX XSAVE
0872 CA                   DEX
0873 D0E9                 BNE S0
0875 60                   RTS
0876               ;
```

Listing Continued . . .

Continued Listing

```
0876            XSAVE   EQU *
0876            ASAVE   EQU *+1
0876            MASK    EQU *+2
087A                    DFS $4
087A            ;
087A AD7808     CLOCK   LDA MASK            ; CREATES ONE CLOCK PULSE
087D 297F               AND #$7F
087F 8DC0C0             STA PORTA
0882 0980               ORA #$80
0884 8DC0C0             STA PORTA
0887 60                 RTS
0888            ;
0888 AD7808     LOAD    LDA MASK            ; CREATES ONE LOAD PULSE
088B 29BF               AND #%10111111
088D 8DC0C0             STA PORTA
0890 0940               ORA #%01000000
0892 8DC0C0             STA PORTA
0895 60                 RTS
0896            ;
0896 A204       BSET    LDX #$04            ; SETS A,B,C,D
0898 8E7608             STX XSAVE
089B 2035FD             JSR RDCHR
089E 20EDFD             JSR OUTCH
08A1 AE7608             LDX XSAVE
08A4 C9C8               CMP #"H"
08A6 D003               BNE B1
08A8 38                 SEC
08A9 B001               BCS B2
08AB 18         B1      CLC
08AC 6E7908     B2      ROR MASK+1
08AF CA                 DEX
08B0 D0E6               BNE BSET+2
08B2 A204               LDX #$04
08B4 6E7908     B3      ROR MASK+1
08B7 CA                 DEX
08B8 D0FA               BNE B3
08BA 60                 RTS
08BB EA         USER    NOP
08BC            ;
08BC            ;
08BC 2058FC     IN      JSR HOME
08BF 200308             JSR TXTOUT
08C2 C5CED4             ASC "ENTER:"
08C5 C5D2BA
08C8 8D00               HEX 8D00
08CA 203508             JSR INIT
08CD 2035FD     IN0     JSR RDCHR
08D0 C9CC               CMP #"L"
08D2 D006               BNE IN1
08D4 208808             JSR LOAD
08D7 4C4709             JMP IN999
08DA C9C3       IN1     CMP #"C"
08DC D006               BNE IN2
08DE 207A08             JSR CLOCK
08E1 4C4709             JMP IN999
08E4 C9D3       IN2     CMP #"S"
```

Listing Continued . . .

Continued Listing

```
08E6 D01D                BNE IN3
08E8 200308              JSR TXTOUT
08EB C1C2C3              ASC "ABCD="
08EE C4BD
08F0 00                  HEX 00
08F1 A900                LDA #$00
08F3 8D7908              STA MASK+1
08F6 209608              JSR BSET
08F9 AD7808              LDA MASK
08FC 29F0                AND #%11110000
08FE 18                  CLC
08FF 6D7908              ADC MASK+1
0902 4C4709              JMP IN999
0905             ;
0905 C9D5        IN3     CMP #"U"
0907 D008                BNE IN4
0909 AD7808              LDA MASK
090C 29DF                AND #%11011111
090E 4C4709              JMP IN999
0911 C9C4        IN4     CMP #"D"
0913 D008                BNE IN5
0915 AD7808              LDA MASK
0918 0920                ORA #%00100000
091A 4C4709              JMP IN999
091D C9C5        IN5     CMP #"E"
091F D020                BNE IN6
0921 200308              JSR TXTOUT
0924 C5BD                ASC "E="
0926 00                  HEX 00
0927 2035FD              JSR RDCHR
092A 20EDFD              JSR OUTCH
092D C9C8                CMP #"H"
092F D008                BNE IN55
0931 A910                LDA #%00010000
0933 0D7808              ORA MASK
0936 4C4709              JMP IN999
0939 A9EF        IN55    LDA #%11101111
093B 2D7808              AND MASK
093E 4C4709              JMP IN999
0941 203AFF      IN6     JSR BEEP
0944 4C59FF              JMP $FF59
0947             ;
0947 8D7808      IN999   STA MASK
094A 8DC0C0              STA PORTA
094D 200308              JSR TXTOUT
0950 8D8D00              HEX 8D8D00
0953 205608              JSR STATE
0956 208EFD              JSR CR
0959 4CCD08              JMP IN0
                        END

***** END OF ASSEMBLY
```

Listing Continued . . .

Continued Listing

```
            *************************
            *                       *
            * SYMBOL TABLE -- V 1.5 *
            *                       *
            *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:


** ABSOLUTE VARABLES/LABELS

DUMMY  1000 OUTCH  FDED RDCHR FD35 HOME  FC58 CR     FD8E BEEP  FF3A
PORTA  C0C0 PORTB  C0C2 CTRLA C0C1 CTRLB C0C3 TXTOUT 0803 ADR   081A
M      082C INIT   0835 STATE 0856 S0    085E S1     0867 S2    0869
XSAVE  0876 ASAVE  0877 MASK  0878 CLOCK 087A LOAD   0888 BSET  0896
B1     08AB B2     08AC B3    08B4 USER  08BB IN     08BC IN0   08CD
IN1    08DA IN2    08E4 IN3   0905 IN4   0911 IN5    091D IN55  0939
IN6    0941 IN999  0947

            SYMBOL TABLE STARTING ADDRESS:6000
            SYMBOL TABLE LENGTH:0142


            !BR

            0800- 4C BB 08 BA E8 BD 00 01
            0808- 8D 1B 08 E8 BD 00 01 8D
            0810- 1C 08 EE 1B 08 D0 03 EE
            0818- 1C 08 AD 00 10 F0 0D 20
            0820- ED FD EE 1B 08 D0 F3 EE
            0828- 1C 08 D0 EE AD 1C 08 48
            0830- AD 1B 08 48 60 A9 00 8D
            0838- C1 C0 8D C3 C0 8D C2 C0
            0840- A9 FF 8D C0 C0 A9 04 8D
            0848- C1 C0 8D C3 C0 A9 D0 8D
            0850- 78 08 8D C0 C0 60 AD C2
            0858- C0 8D 77 08 A2 08 6E 77
            0860- 08 B0 04 A9 CC 90 02 A9
            0868- C8 8E 76 08 20 ED FD AE
            0870- 76 08 CA D0 E9 60 01 00
            0878- E0 01 AD 78 08 29 7F 8D
            0880- C0 C0 09 80 8D C0 C0 60
            0888- AD 78 08 29 BF 8D C0 C0
            0890- 09 40 8D C0 C0 60 A2 04
            0898- 8E 76 08 20 35 FD 20 ED
            08A0- FD AE 76 08 C9 C8 D0 03
            08A8- 38 B0 01 18 6E 79 08 CA
            08B0- D0 E6 A2 04 6E 79 08 CA
            08B8- D0 FA 60 EA 20 58 FC 20
            08C0- 03 08 C5 CE D4 C5 D2 BA
            08C8- 8D 00 20 35 08 20 35 FD
            08D0- C9 CC D0 06 20 88 08 4C
            08D8- 47 09 C9 C3 D0 06 20 7A
```

Listing Continued . . .

Continued Listing

```
08E0- 08 4C 47 09 C9 D3 D0 1D
08E8- 20 03 08 C1 C2 C3 C4 BD
08F0- 00 A9 00 8D 79 08 20 96
08F8- 08 AD 78 08 29 F0 18 6D
0900- 79 08 4C 47 09 C9 D5 D0
0908- 08 AD 78 08 29 DF 4C 47
0910- 09 C9 C4 D0 08 AD 78 08
0918- 09 20 4C 47 09 C9 C5 D0
0920- 20 20 03 08 C5 BD 00 20
0928- 35 FD 20 ED FD C9 C8 D0
0930- 08 A9 10 0D 78 08 4C 47
0938- 09 A9 EF 2D 78 08 4C 47
0940- 09 20 3A FF 4C 59 FF 8D
0948- 78 08 8D C0 C0 20 03 08
0950- 8D 8D 00 20 56 08 20 8E
0958- FD 4C CD 08 FF FF FF FF
*
```

NOTES

9

The Control of Step Motors

A step motor can be imagined as a mechanical digital to analog converter. The input is the number of pulses; the output is the same number of steps on a rotating shaft. The number of steps per revolution can vary. There are motors with 4 steps per revolution and a step angle of 90 degrees; others have up to 500 steps per revolution with a step angle of 0.72 degrees. Another characteristic of step motors is the maximum number of steps per second. This could be some 100 steps per second up to 10,000 steps per second, depending on the mechanical dimensions of the motor. The third characteristic we will mention here is the maximum number of steps per second with which the motor can start, called the starting frequency. This frequency depends on the number of steps per revolution and the moment of inertia which the motor must overcome. Once started, the step motor can reach higher frequencies by slowly varying the number of steps per second.

Step motors are used in a wide variety of applications, such as numerically controlled machines, digital plotters, medical equipment, and in various other cases where a rotating angle or linear length is controlled by a computer. In this chapter we will discuss some examples of controlling step motors with a computer program. It isn't important which motor is used. This depends on the mechanical environment. All these programs were tested with a step motor from SIGMA Instruments, having 200 steps per revolution, which gives a step angle of 1.8 degrees.

In Figures 9.1 through 9.4 the basic movement of a step motor is shown. We have two separately wound stators and a polarized rotor. With the switches (A and B) in the position shown, the rotor is in a stable position.

When we change switch B from 1 to 0, the position shown in Fig. 9.1 is no longer stable. Another stable position results, as shown in Figure 9.2.

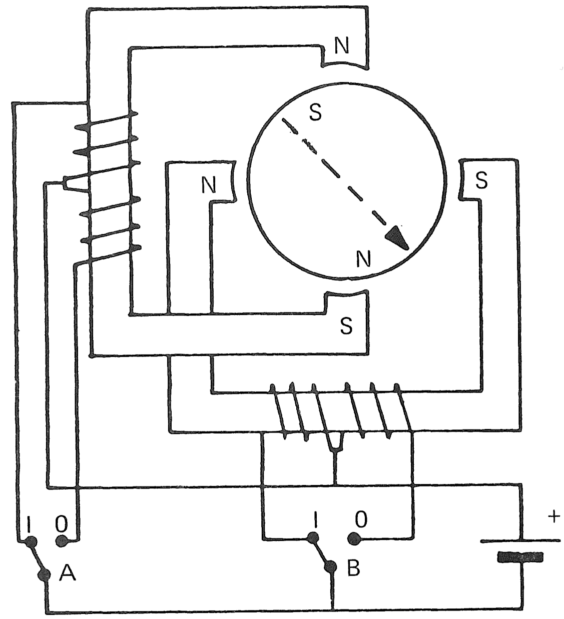


Figure 9.1 Step Motor Movement

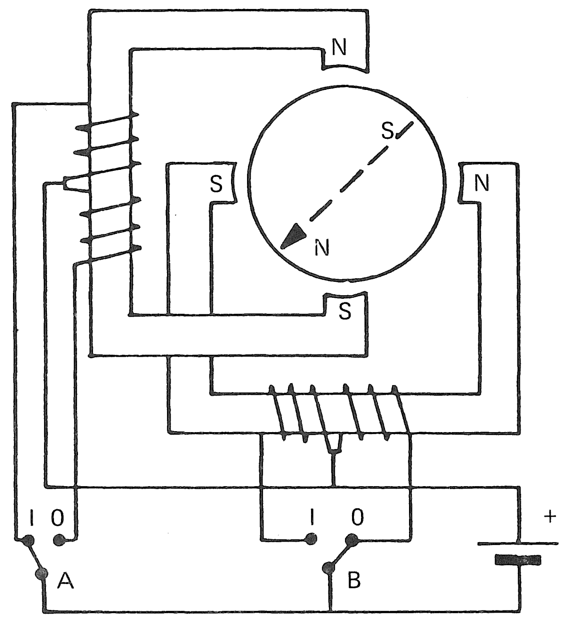


Figure 9.2 Step Motor Movement

The motor has turned one step to the right. When we change switch A from 1 to 0, the motor makes one more step to the right.

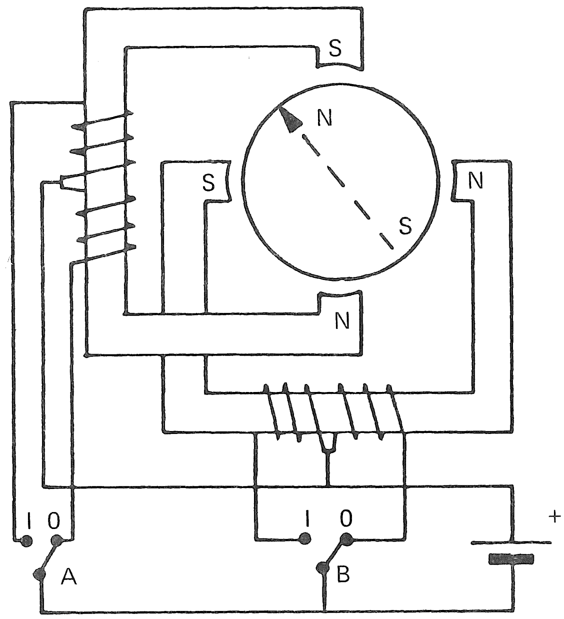


Figure 9.3 Step Motor Movement

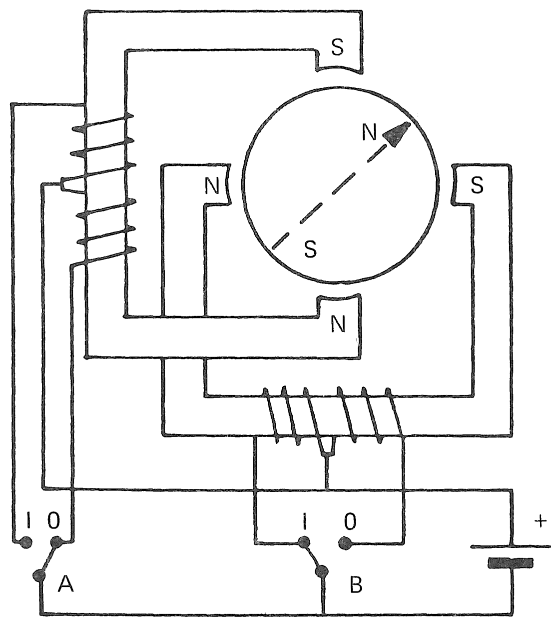


Figure 9.4 Step Motor Movement

If we now change switch B from 0 to 1, the step motor makes a third step to the right. When we then change switch A from 0 to 1, the motor reaches its starting position once again. This model represents a step motor with 4 steps per revolution and a step angle of 90 degrees. Figure 9.5 shows the timing diagram for right turns.

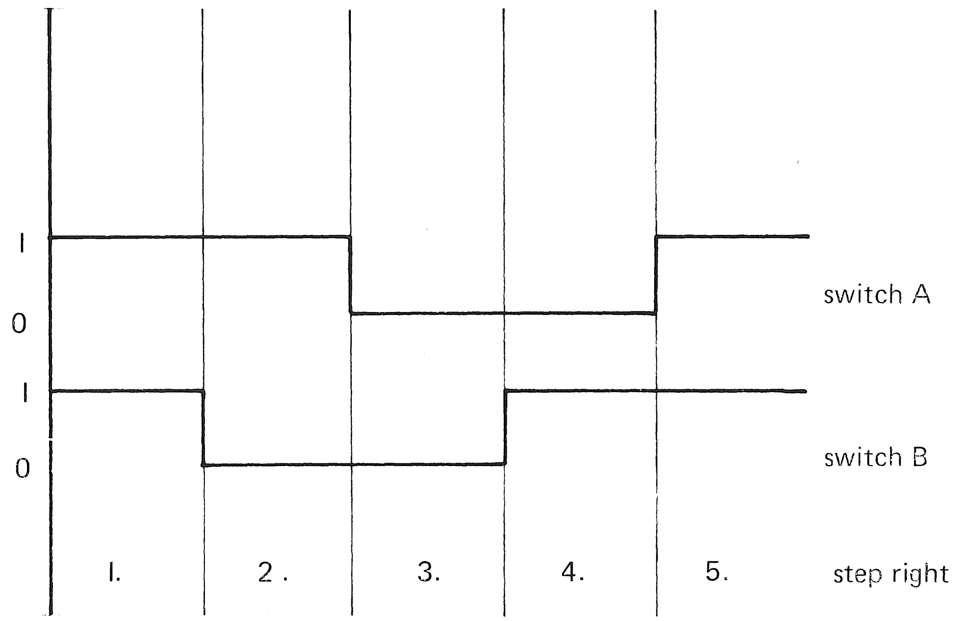


Figure 9.5 Right Turn Timing Diagram

To create left turns with our model, we first change switch A from 1 to 0, then switch B, and so on, as shown in Figure 9.6.

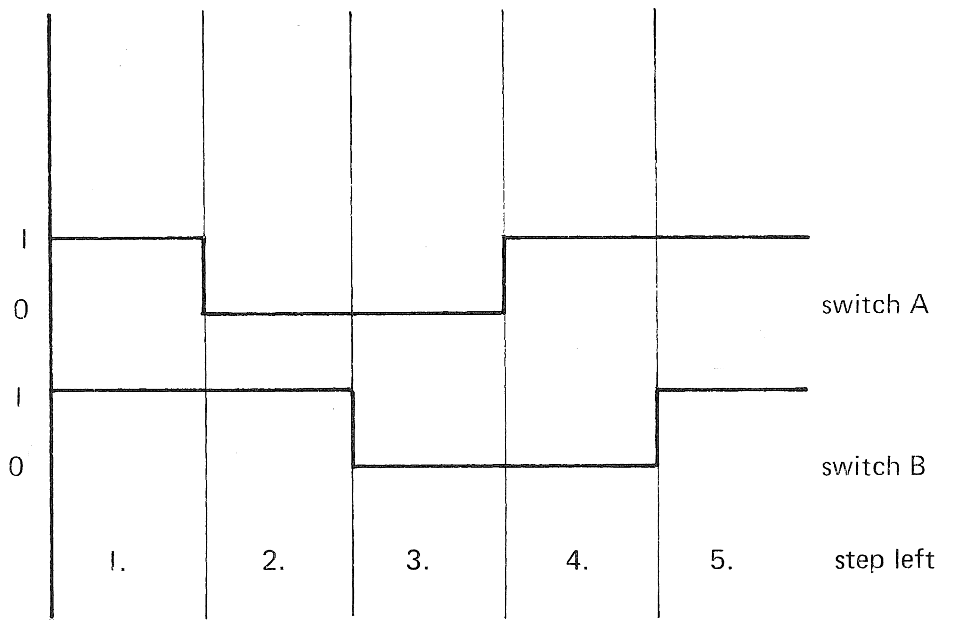


Figure 9.6 Left Turn Timing Diagram

Figure 9.7 shows the equipment for the experiments with a step motor. The I/O interface card is in slot 4 of the Apple bus. PB0 and PB1 are connected to the inputs of the power amplifier which drives the motor.

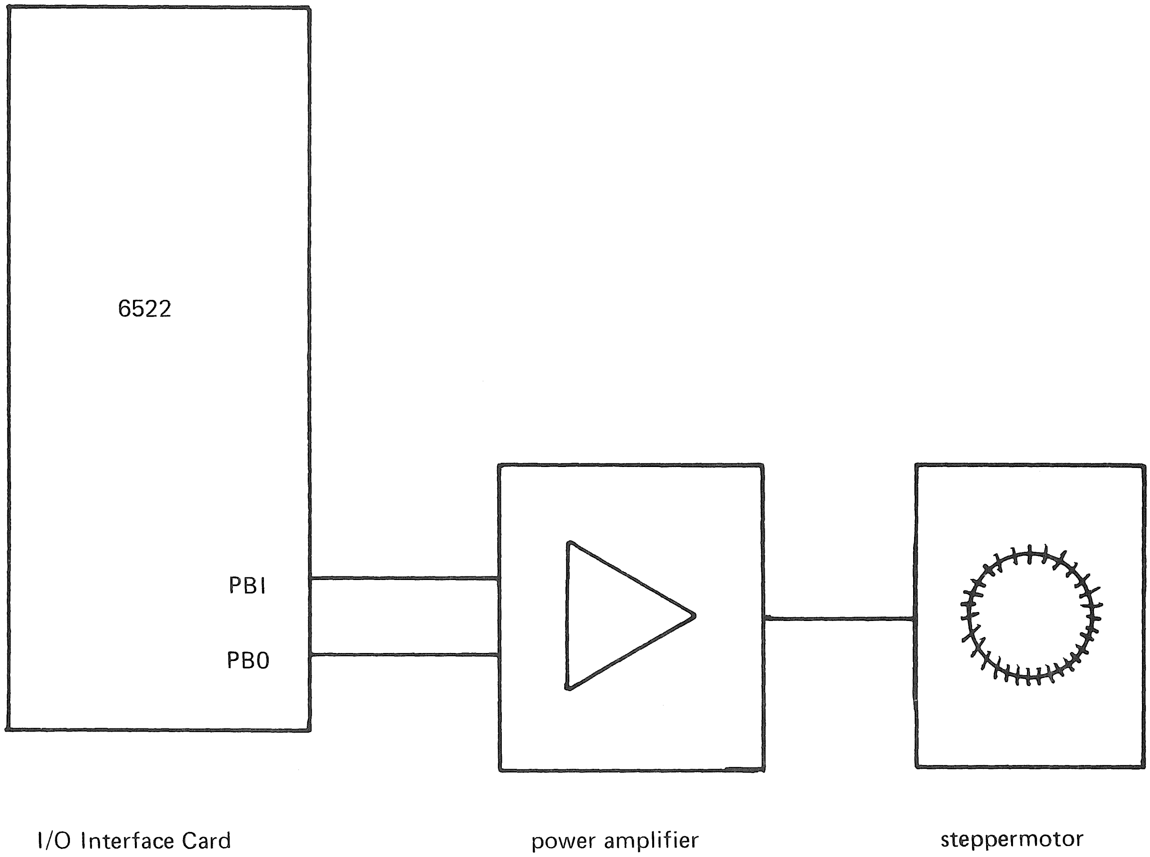


Figure 9.7 Block Diagram

In our first example, we used BASIC's POKE instructions. Figure 9.8 shows the program for a right turn; Figure 9.9 shows the program for a left turn.

Figure 9.8 Right Turn Program

```
 LIST

10  REM  STEPPERMOTOR GOES RIGHT
20  REM
30  REM
100 DDRB =  - 16190:PB =  - 16192
200  POKE DDRB,3
210  POKE PB,3
215  GOSUB 300
220  POKE PB,1
225  GOSUB 300
230  POKE PB,0
235  GOSUB 300
240  POKE PB,2
245  GOSUB 300
250  GOTO 210
300  FOR I = 1 TO 100: NEXT I: RETURN
```

Between the single steps there is a delay loop on line 300. Changing the ending value of the loop changes the speed of the motor.

Figure 9.9 Left Turn Program

```
 LIST

10   REM   STEPPERMOTOR GOES LEFT
20   REM
30   REM
100 DDRB =  - 16190:PB =  - 16192
200  POKE DDRB,3
210  POKE PB,3
215  GOSUB 300
220  POKE PB,2
225  GOSUB 300
230  POKE PB,0
235  GOSUB 300
240  POKE PB,1
245  GOSUB 300
250  GOTO 210
300  FOR I = 1 TO 100: NEXT I: RETURN
```

Figure 9.10 Stepping Program

```
 LIST

10   REM  CHOOSING RIGHT OR LEFT TURN
20   REM  AND NUMBER OF STEPS
30   REM
40   REM
100 INIT =  - 15360
110 RIGHT =  - 15349:LEFT =  - 15340
120 ST =  - 15331
150 BEEB$ = ""
200  CALL INIT
210  INPUT "R)IGHT,L)EFT,E)ND: ";A$
220  IF A$ = "R" THEN  CALL RIGHT: GOTO 250
230  IF A$ = "L" THEN  CALL LEFT: GOTO 250
235  IF A$ = "E" THEN  END
240  PRINT BEEP$: GOTO 210
250  INPUT "NUMBER OF STEPS:";N
300  FOR I = 1 TO N
310  CALL ST
320  NEXT I
330  PRINT : GOTO 210
```

Figure 9.11 Machine-language Version

```
PR#1

0800                    DCM "PR#1"
0800             ;
0800             ;
0800             ;*****************************
0800             ;*                           *
0800             ;* MACHINE-ROUTINES FOR      *
0800             ;* CONTROLLING STEPPER MOTORS*
0800             ;*                           *
0800             ;*****************************
0800             ;
0800             ;
0800             DDRB   EQU $C0C2
0800             PORTB  EQU $C0C0
0800             ACR    EQU $C0CB
0800             S1     EPZ $1E
0800             ;
C400                    ORG $C400
C400             ;
C400 A983        INIT   LDA #$83
C402 8DC2C0             STA DDRB
C405 A900               LDA #$00
C407 8DCBC0             STA ACR
C40A 60                 RTS
C40B             ;
C40B             ;
C40B A902        RIGHT  LDA #$02
C40D 851E               STA S1
C40F A900               LDA #$00
C411 851F               STA S1+1
C413 60                 RTS
C414             ;
C414             ;
C414 A901        LEFT   LDA #$01
C416 851E               STA S1
C418 A900               LDA #$00
C41A 851F               STA S1+1
C41C 60                 RTS
C41D             ;
C41D             ;
C41D A51E        STEP   LDA S1
C41F 4903               EOR #$03
C421 851E               STA S1
C423 451F               EOR S1+1
C425 8DC0C0             STA PORTB
C428 851F               STA S1+1
C42A 60                 RTS
C42B             ;
C42B             ;
C42B A51E        CHANGE LDA S1
C42D 451F               EOR S1+1
C42F 8DC0C0             STA PORTB
C432 A51E               LDA S1
C434 4903               EOR #$03
```

Listing Continued . . .

Continued Listing

```
C436 851E      54          STA S1
C438 60        55          RTS
C439           56  ;
C439           57  ;
              58  FIN     END

         ************************
         *                      *
         * SYMBOL TABLE -- V 1.5 *
         *                      *
         ************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

S1     001E

** ABSOLUTE VARABLES/LABELS

DDRB   C0C2  PORTB  C0C0  ACR    C0CB  INIT   C400  RIGHT  C40B
LEFT   C414  STEP   C41D  CHANGE C42B  FIN    C439

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0062
```

In this program, machine-language is used for setting the starting conditions for right or left turns. The timing sequences are also generated by machine-language. This routine is called STEP and is shown in Figure 9.11

This tricky program is explained in Figure 9.12.

Figure 9.12

	ACCU	S1	S1 + 1	PORT B	
	X X	0 1	0 0	0 0	Starting condition left
LDA SI	0 1				
EOR #03	1 0				
STA SI		1 0			
EOR SI+I	1 0				
STA PORT B				1 0	1. Step
STA SI+I			1 0		
LDA SI	1 0				
EOR #03	0 1				
STA SI		0 1			
EOR SI+I	1 1				
STA PORT B				1 1	2. Step
STA SI+I			1 1		
LDA SI	0 1				
EOR #03	1 0				
STA SI		1 0			

Continued Listing

```
EOR SI+I      0 I
STA PORT B                             0 I        3. Step
STA SI+I                        0 I

LDA SI        I 0
EOR #03       0 I
STA SI                 0 I
EOR SI+I      0 0
STA PORT B                             0 0        4. Step
STA SI+I                        0 0
```

The starting condition is set for a left turn. The first four steps are demonstrated. The sequence of steps is the same as shown in Figure 9.6 with switch A equal to PB1, switch B equal to PBO, and starting with step 3 of the diagram.

The next BASIC program makes the step motor continuously perform the same movement. The following sequence is programmed: 200 steps to the left, wait, 100 steps to the right with the same speed, and then 100 steps to the right with a slow speed.

Figure 9.13 Continuous Movement Program

```
 LIST

100 INIT =  - 15360
110 RIGHT =  - 15349:LEFT =  - 15340
120 ST =  - 15331
130 CHANGE =  - 15317
200  CALL INIT
210  CALL RIGHT
220  FOR I = 1 TO 200: CALL ST: FOR K = 1 TO 5: NEXT K: NEXT I
225  GOSUB 300: REM  WAIT
230  CALL CHANGE
240  FOR I = 1 TO 100: CALL ST: FOR K = 1 TO 5: NEXT K: NEXT I
250  FOR I = 1 TO 100: CALL ST
260  FOR J = 1 TO 20: NEXT J
270  NEXT I
280  CALL CHANGE: GOTO 220
299  END
300  FOR J = 1 TO 2000: NEXT J: RETURN
```

The subroutine CHANGE is used for changing the direction of the step motor.

Now we will use another language, PASCAL, for the control of step motors. In this high-level language, we write the same machine-language routines as in BASIC, but this time they are prepared for linking to a PASCAL program.

The PASCAL program is shown in Figure 9.15.

Figure 9.14a Machine-language Subroutine

```
PAGE -   0
Current memory available:    8657
0000|
0000|                                 ;MAKRO POP
0000|                                 ;
0000|                                 ;
0000|                                 .MACRO POP
0000|                                 PLA
0000|                                 STA %1
0000|                                 PLA
0000|                                 STA %1+1
0000|                                 .ENDM
0000|                                 ;
0000|                                 ;
0000|                                 .MACRO PUL
0000|                                 LDA %1+1
0000|                                 PHA
0000|                                 LDA %1
0000|                                 PHA
0000|                                 .ENDM
0000|                                 ;
0000|                                 ;
0000| 0000                       RETURN   .EQU 0
0000| C0C2                       DDRB     .EQU 0C0C2
0000| C0C0                       TORB     .EQU 0C0C0
0000| C0CB                       ACR      .EQU 0C0CB
0000| C0C4                       T1L      .EQU 0C0C4
0000| C0C5                       T1H      .EQU 0C0C5
0000| 0013                       S1       .EQU 13
0000| C400                       ZAHL     .EQU 0C400
0000|
2 blocks for procedure code  8053 words left
```

Figure 9.14b Machine-language Subroutine

```
0000|                                      .PROC INIT
Current memory available:    8004
0000|
0000|                                      POP RETURN
0000| 68                     #    PLA
0001| 85 00                  #    STA RETURN
0003| 68                     #    PLA
0004| 85 01                  #    STA RETURN+1
0006| A9 83                               LDA #83
0008| 8D C2C0                             STA DDRB
000B| A9 C0                               LDA #0C0
000D| 8D CBC0                             STA ACR
0010| A9 00                               LDA #00
0012| 85 17                               STA S1+4
0014| 85 14                               STA S1+1
0016| A9 02                               LDA #02
0018| 85 13                               STA S1
001A|                                     PUL RETURN
001A| A5 01                  #    LDA RETURN+1
001C| 48                     #    PHA
001D| A5 00                  #    LDA RETURN
```

Listing Continued . . .

Continued Listing

```
001F| 48                    #   PHA
0020| 60                            RTS
0021|
```

Figure 9.14c Machine-language Subroutine

```
0000|                               .PROC RIGHT
Current memory available:    8004
0000|                               POP RETURN
0000| 68                    #   PLA
0001| 85 00                 #   STA RETURN
0003| 68                    #   PLA
0004| 85 01                 #   STA RETURN+1
0006| A5 17                         LDA S1+4
0008| F0**                          BEQ L
000A| A5 13                         LDA S1
000C| 49 03                         EOR #03
000E| 85 13                         STA S1
0010| A9 00                         LDA #00
0012| 85 17                         STA S1+4
0014|                        L      PUL RETURN
0008* 0A
0014| A5 01                 #   LDA RETURN+1
0016| 48                    #   PHA
0017| A5 00                 #   LDA RETURN
0019| 48                    #   PHA
001A| 60                            RTS
001B|
```

Figure 9.14d Machine-language Subroutine

```
0000|                               .PROC LEFT
Current memory available:    8004
0000|                               POP RETURN
0000| 68                    #   PLA
0001| 85 00                 #   STA RETURN
0003| 68                    #   PLA
0004| 85 01                 #   STA RETURN+1
0006| A5 17                         LDA S1+4
0008| D0**                          BNE LL
000A| A5 13                         LDA S1
000C| 49 03                         EOR #03
000E| 85 13                         STA S1
0010| 85 17                         STA S1+4
0012|                        LL     PUL RETURN
0008* 08
0012| A5 01                 #   LDA RETURN+1
0014| 48                    #   PHA
0015| A5 00                 #   LDA RETURN
0017| 48                    #   PHA
0018| 60                            RTS
0019|
0019|
```

Figure 9.14e Machine-language Subroutine

```
0000|                               .PROC STEP
Current memory available:    8004
0000|
0000|                               POP RETURN
```

Figure 9.14f Machine-language Subroutine

```
0000| 68                    #   PLA
0001| 85 00                 #   STA RETURN
0003| 68                    #   PLA
0004| 85 01                 #   STA RETURN+1
0006| A5 13                         LDA S1
0008| 49 03                         EOR #03
000A| 85 13                         STA S1
000C| 45 14                         EOR S1+1
000E| 8D C0C0                       STA TORB
0011| 85 14                         STA S1+1
0013|                               PUL RETURN
0013| A5 01                 #   LDA RETURN+1
0015| 48                    #   PHA
0016| A5 00                 #   LDA RETURN
0018| 48                    #   PHA
0019| 60                            RTS
001A|
```

Figure 9.14g Machine-language Subroutine

```
0000|                               .PROC WAIT,1
Current memory available:   8004
0000|                               POP RETURN
0000| 68                    #   PLA
0001| 85 00                 #   STA RETURN
0003| 68                    #   PLA
0004| 85 01                 #   STA RETURN+1
0006| 68                            PLA
0007| 8D 00C4                       STA ZAHL
000A| 68                            PLA
000B| 8D 01C4                       STA ZAHL+1
000E| CE 00C4                L1     DEC ZAHL
0011| D0FB                          BNE L1
0013| AD 01C4                       LDA ZAHL+1
0016| F0**                          BEQ L2
0018| CE 01C4                       DEC ZAHL+1
001B| 18                            CLC
001C| 90F0                          BCC L1
001E|                        L2     PUL RETURN
0016* 06
001E| A5 01                 #   LDA RETURN+1
0020| 48                    #   PHA
0021| A5 00                 #   LDA RETURN
0023| 48                    #   PHA
0024| 60                            RTS
0025|
0025|                               .END
```

Figure 9.15 PASCAL Control Program

```
PROGRAM SPEED;
USES APPLESTUFF;
VAR N,R,I,K,L,X,PD:INTEGER;
    CH:CHAR;
    DIR:BOOLEAN;

PROCEDURE INIT;
EXTERNAL;
```

Listing Continued . . .

Continued Listing

```
PROCEDURE STEP;
EXTERNAL;


PROCEDURE RIGHT;
EXTERNAL;

PROCEDURE LEFT;
EXTERNAL;

PROCEDURE WAIT( W:INTEGER);
EXTERNAL;


BEGIN
INIT;RIGHT;
WRITE('L=');READLN(L);

REPEAT
PD:=PADDLE(0);
   BEGIN
     STEP;
     WAIT(L+PD);
   END;
UNTIL KEYPRESS;
END.
```

With this program, the speed of the step motor is controlled by paddle 0 via the game connector. The basic speed is read from the keyboard (READLN(L)), and changes are made by changing the value of paddle 0. We declare all machine-language routines, which we will use as external routines. We give them the same names as shown in the assembler routines in Figure 9.14. The procedure WAIT will pass one parameter from the main program to the machine-language routines. This is the sum of two integers (L and PD) and is equal to the delay time between two steps of the motor. This parameter of the main program is passed to the machine subroutine via the stack. The number of parameters has to be given after declaring the name of the procedure.

The declaration .PROC WAIT,1 means that one 16-bit number is passed to the routine. After POPping the RETURN address from the stack, the 16-bit number is POPped from the stack (low-order byte first) and is stored in memory locations ZAHL and ZAHL + 1.

The next program, TIM, uses timer 1 of the 6522 as a square-wave generator in the free-running mode. The frequency of the square-wave is determined by a number written to the timer latches. The conditions for the timers are set in such a manner that when the number 200 is written to the timer, a square-wave of 200 cycles per second is created. The timimg sequence for the step motor shown in Figures 9.5 and 9.6, created previously by software, is now done by the hardware.

The program asks for a starting frequency. If you input the number 200, a 200 steps per revolution motor will perform exactly one revolution per second. When you enter a new speed of rotation, the step motor will not reach this speed by a linear function, but by the function given in Figure 9.17.

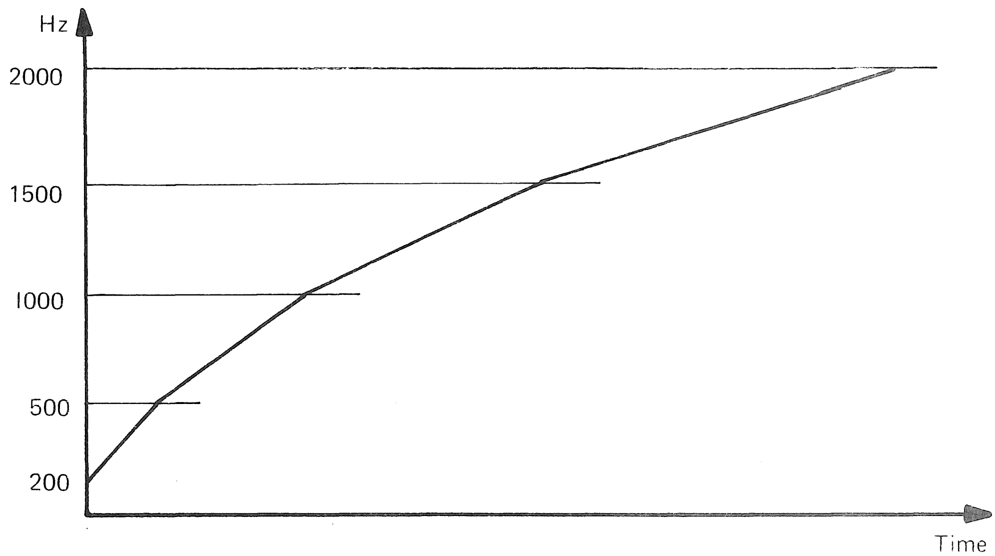


Figure 9.17 Step Motor Acceleration

Figure 9.18 Machine-language Subroutine

```
PAGE -    0
Current memory available:    8657
0000|
0000|                            ;MAKRO POP
0000|                            ;
0000|                            ;
0000|                            .MACRO POP
0000|                            PLA
0000|                            STA %1
0000|                            PLA
0000|                            STA %1+1
0000|                            .ENDM
0000|                            ;
0000|                            ;
0000|                            ;
0000|                            .MACRO PUL
0000|                            LDA %1+1
0000|                            PHA
0000|                            LDA %1
0000|                            PHA
0000|                            .ENDM
0000|                            ;
0000|                            ;
```

Listing Continued . . .

Continued Listing

```
0000|
0000| 0000                    RETURN   .EQU 0
0000| C0C2                    DDRB     .EQU 0C0C2
0000| C0C0                    TORB     .EQU 0C0C0
0000| C0CB                    ACR      .EQU 0C0CB
0000| C0C4                    T1L      .EQU 0C0C4
0000| C0C5                    T1H      .EQU 0C0C5
0000| C0C3                    DDRA     .EQU 0C0C3
0000| C0C1                    TORA     .EQU 0C0C1
0000| C0C6                    T1LL     .EQU 0C0C6
0000| C0C7                    T1HL     .EQU 0C0C7
0000| 0010                    H        .EQU 10
0000|
2 blocks for procedure code  8029 words left

0000|                                  .PROC TIMEINIT
Current memory available:    7980
0000|                                  POP RETURN
0000| 68                     #    PLA
0001| 85 00                  #    STA RETURN
0003| 68                     #    PLA
0004| 85 01                  #    STA RETURN+1
0006| A9 03                            LDA #03
0008| 8D C3C0                          STA DDRA
000B| A9 10                            LDA #10
000D| 8D C2C0                          STA DDRB
0010| A9 C0                            LDA #0C0
0012| 8D CBC0                          STA ACR
0015| A9 00                            LDA #00
0017| 8D C1C0                          STA TORA
001A| A9 03                            LDA #03
```

Listing Continued . . .

Continued Listing

```
001C| 8D C1C0                          STA TORA
001F|                                  PUL RETURN
001F| A5 01                  #    LDA RETURN+1
0021| 48                     #    PHA
0022| A5 00                  #    LDA RETURN
0024| 48                     #    PHA
0025| 60                               RTS
0026|
0026|
```

Listing Continued . . .

Continued Listing

```
0000|                                   .PROC SETTIMER,1
Current memory available:    7980
0000|                                   POP RETURN
0000| 68                    #    PLA
0001| 85 00                 #    STA RETURN
0003| 68                    #    PLA
0004| 85 01                 #    STA RETURN+1
0006| 68                                PLA
0007| 8D C4C0                           STA T1L
000A| 68                                PLA
000B| 8D C5C0                           STA T1H
000E|                                   PUL RETURN
000E| A5 01                 #    LDA RETURN+1
0010| 48                    #    PHA
0011| A5 00                 #    LDA RETURN
0013| 48                    #    PHA
0014| 60                                RTS
0015|
0015|
0015|
0015|
```

Listing Continued . . .

Continued Listing

```
0000|                                   .PROC CHANGETIME,1
Current memory available:    7980
0000|                                   POP RETURN
0000| 68                    #    PLA
0001| 85 00                 #    STA RETURN
0003| 68                    #    PLA
0004| 85 01                 #    STA RETURN+1
0006| AD C0C0                           LDA TORB
0009| 29 10                             AND #10
000B| 85 10                             STA H
000D| AD C0C0                L          LDA TORB
0010| 45 10                             EOR H
0012| F0F9                              BEQ L
0014| 68                                PLA
0015| 8D C6C0                           STA T1LL
0018| 68                                PLA
0019| 8D C7C0                           STA T1HL
001C|                                   PUL RETURN
001C| A5 01                 #    LDA RETURN+1
001E| 48                    #    PHA
001F| A5 00                 #    LDA RETURN
0021| 48                    #    PHA
0022| 60                                RTS
0023|
```

The lower the speed of the motor, the greater the change of speed can be. When a new speed is entered, the program determines if it has to go to higher speeds (PROCEDURE GOHIGHER) or to go to lower speeds (PROCEDURE GODOWN). When the given frequency is reached, a new frequency can be entered. The CPU of the Apple is only used when changes of the step-rate are made. When there are no changes, timer 1 of the 6522 stays in the free-running mode, and the CPU is free to perform other tasks. This program uses more machine-language programs, as shown in Figure 9.18. The PASCAL program is shown in Figure 9.19.

Figure 9.19 PASCAL Control Program

```
PROGRAM TIM;
CONST A=1E6;
VAR I,J,K,L,F,R,OLD,NEW,DESTINATION:INTEGER;
    CH:CHAR;

PROCEDURE TIMEINIT;
EXTERNAL;

PROCEDURE SETTIMER(T:INTEGER);
EXTERNAL;

PROCEDURE CHANGETIME(T:INTEGER);
EXTERNAL;

PROCEDURE STOP;
EXTERNAL;



PROCEDURE GODOWN(VAR AL,NE:INTEGER);
PROCEDURE DOWN(STEP:INTEGER);
  BEGIN
    WHILE AL>DESTINATION DO
    BEGIN
      IF AL-DESTINATION<STEP
      THEN BEGIN
             AL:=DESTINATION;
             R:=AL;
           END ELSE
           BEGIN
             AL:=AL-STEP;
             R:=AL;
           END;
      IF R <> 0 THEN
      BEGIN
      F:=TRUNC(1/R*A);
      CHANGETIME(F);
      END;
    END;
END;    (* DOWN *)
  BEGIN
  IF AL>1500 THEN
```

Listing Continued . . .

Continued Listing

```
  BEGIN
    REPEAT
      IF NE>1500 THEN DESTINATION:=NE ELSE DESTINATION:=1500;
      DOWN(1);
    UNTIL AL=DESTINATION;
  END;
    IF(AL<>NE) AND (AL>1000) THEN
    BEGIN
         REPEAT
           IF NE>1000 THEN DESTINATION:=NE ELSE DESTINATION:=1000;
           DOWN(2);
         UNTIL AL=DESTINATION;
    END;

    IF(AL<>NE) AND (AL> 500) THEN
    BEGIN
         REPEAT
           IF NE> 500 THEN DESTINATION:=NE ELSE DESTINATION:= 500;
           DOWN(5);
         UNTIL AL=DESTINATION;;
    END;

    IF (AL<>NE) AND (AL> 100) THEN
    BEGIN
         REPEAT
            DESTINATION:=NE;
            DOWN(10);
         UNTIL AL=DESTINATION;
    END;


  END;   (* GODOWN *)



PROCEDURE GOHIGHER(VAR AL,NE:INTEGER);
PROCEDURE UP(STEP:INTEGER);
  BEGIN
    WHILE AL<DESTINATION DO
    BEGIN
      IF DESTINATION-AL<STEP
      THEN BEGIN
             AL:=DESTINATION;
             R:=AL;
           END ELSE
           BEGIN
             AL:=AL+STEP;
             R:=AL;
           END;
      IF R <> 0 THEN
      BEGIN
      F:=TRUNC(1/R*A);
      CHANGETIME(F);
      END;
```

Listing Continued . . .

Continued Listing

```
     END;
 END;   (* UP *)
   BEGIN
   IF AL<500 THEN
   BEGIN
     REPEAT
       IF NE<500 THEN DESTINATION:=NE ELSE DESTINATION:=500;
       UP(10);
     UNTIL AL=DESTINATION;
   END;
     IF(AL<>NE) AND (AL<1000) THEN
     BEGIN
          REPEAT
            IF NE<1000 THEN DESTINATION:=NE ELSE DESTINATION:=1000;
            UP(5);
          UNTIL AL=DESTINATION;
     END;

     IF(AL<>NE) AND (AL<1500) THEN
     BEGIN
          REPEAT
            IF NE<1500 THEN DESTINATION:=NE ELSE DESTINATION:=1500;
            UP(2);
          UNTIL AL=DESTINATION;;
     END;

     IF (AL<>NE) AND (AL<2000) THEN
     BEGIN
          REPEAT
             DESTINATION:=NE;
             UP(1);
          UNTIL AL=DESTINATION;
     END;


   END;   (* GOHIGHER *)


 BEGIN
   TIMEINIT;
     WRITE('STARTING FREQUENCY=');READLN(K);
     F:=TRUNC(1/K*A);
     SETTIMER(F);
     OLD:=K;
  REPEAT
    WRITE('NEW FREQUENCY ? ');READ(CH);
     IF CH <> 'N' THEN
     BEGIN
       WRITE(' =');READLN(K);
       NEW:=K;
       IF NEW>OLD THEN GOHIGHER(OLD,NEW) ELSE GODOWN(OLD,NEW);
       WRITELN('THE END');
     END;
   UNTIL CH='N';
 END.
```

Finally, we will use a third language for controlling the step motor. This language is FORTH. The program is shown in Figure 9.20. The definition of the verbs begins with the word START. PB0 and PB1 of the 6522 are set for outputs (assuming the 6522 board is in slot 4, as usual). The S means STORE only to Port B. The verbs RIGHT and LEFT set the starting conditions for a left or right turn. The following verb STEP is the FORTH implementation of the machine-language program in Figure 9.11 and is explained in Figure 9.21.

Figure 9.20 FORTH Program

```
( ***************************
 * STEPPER MOTOR CONTROL    *
 ***************************
)
HEX
FORGET STEPS
: STEPS ;
: START 0003 C0C2 ! DEC ;
: S C0C0 ! ;
: RIGHT 01 00 ;
: LEFT 02 00 ;
: STEP SWAP 03 EOR DUP
       ROT ROT EOR DUP
       S ;
: WAIT  20 0 DO LOOP ;
: GO 0 DO STEP WAIT LOOP ;
```

Figure 9.21 Definition of Step

		TOS (Top of Stack) ↓		
	01	00		
	00	01	SWAP	
00	01	11	03	
	00	10	EOR	
00	10	10	DUP	1. STEP
10	00	10	ROT	
10	10	00	ROT	10
	10	10	EOR	
10	10	10	DUP	
	10	10	S	
	10	10	SWAP	
10	10	11	03	
	10	10	EOR	
10	01	01	DUP	2. STEP
01	10	01	ROT	
01	01	10	ROT	11
	01	11	EOR	
01	11	11	DUP	
	01	11	S	
	11	01	SWAP	
11	01	11	03	
	11	10	EOR	
11	10	10	DUP	3. STEP
10	11	10	ROT	
10	10	11	ROT	01
	10	01	EOR	
10	01	01	DUP	
	10	01	S	
	01	10	SWAP	
01	10	11	03	
	01	01	EOR	
01	01	01	DUP	4. STEP
01	01	01	ROT	
01	01	01	ROT	00
	01	00	EOR	
01	00	00	DUP	
	01	00	S	

We consider only the two lowest bits of the number on top of the stack. The top of stack (TOS) is represented in the rightmost column. We start with the direction RIGHT. After running for the first time through STEP, a 10B is stored in Port B. As we continue to run through STEP, we store an 11B, then an 01B, and then a 00B. At the end of the fourth step, we have the same starting conditions as for the first step. Looking at Figure 9.5, we start here with step 4; then step 5 follows, which is the same as step 1, and so on.

In the program a wait loop follows with the verb WAIT. This is a constant time delay between each step. The last verb (GO) is the main loop which uses STEP and WAIT. Before calling this verb, the number of steps must be on top of the stack. With the following input, the step motor makes 100 steps to the right:

```
START
RIGHT
100 GO
```

First we set the starting conditions, next we define the direction, and finally, we'll tell the program how many steps the step motor has to perform.

As we have seen, a step motor can be easily controlled by a computer, creating many possible applications. One last application should be mentioned here. With a step motor it is possible to create a very exact number of revolutions per second. The timer of the 6522 is controlled by the quartz of the computer. The number of revolutions is therefore controlled in the same manner. This is very important in testing mechanical vibration equipment. The accuracy of the number of revolutions per second is approximately 10 to the minus 6th. That accuracy couldn't be duplicated by an ordinary electric motor.

PARTS SUPPLIERS

The Four Stars

Digi-Key Corporation, Hiway 32 South, P. O. Box 677, Thief River Falls, MN 56701. 800 346-5144. *COD, check, money order, credit cards. Volume discounts over $100; shipping, insurance prepaid.*

This company is in my opinion the hobbyist's best. Shipping is fast (five days from ordering to my door in Vermont), and most items are in stock. Their catalog is monthly, and items not stocked are not listed. All merchandise is prime; no bubble packs.

QT Computer Systems, 15335 S. Hawthorne Blvd., Lawndale, CA 90260. 800 421-5150. *COD under $100, check, money order; credit cards preferred. Quantity discounts, no insurance.*

A good hobbyist catalog similar to Digi-Key, with competitive prices. This is a new company, but they are already beginning to make a mark for promptness, exceeding courtesy, and prime parts. Their catalog is very complete and quite up-to-date.

Advanced Computer Products, 1310 E. Edinger, Santa Ana, CA 92705. 800 854-8230. *COD, check, money order, credit cards. Volume discounts, no insurance.*

One of the best catalogs in the business, prompt, but be wary of substitutions in orders. Specify voltages of devices and check upon receipt. Expect harrassment from Customer Service. Otherwise, they have what you can't get anywhere else.

Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002. 415 594-8097. *COD, check, money order, no credit cards. No discounts, no insurance.*

One almost wonders why to put Jameco in with the four best, but their selection is contemporary and their response prompt. They have items others don't have in stock for the popular computer hobbyist. Bubble pack stuff on retail store racks at Lafayette Radio and others. Highest prices in the business.

And Others

Jade Computer Products, 4901 West Rosecrans Ave., Hawthorne, CA 90250. 800 421-5500. *No CODs; checks, money order; credit cards preferred. Quantity discounts; insurance under 50 lbs.*

This company works hard at immediate hobbyist needs and some unusual items. Get their catalog, but consult monthly ads in electronics magazines for hot items.

Priority One Electronics, 16723C Roscoe Blvd., Sepulveda, CA 91343. 800 423-5633. *No CODs; check, money order, credit cards. Quantity discounts, insurance.*

Priority deals for the most part in larger items for computer hobbyists, with only a token selection of small parts. This company concentrates on boards, naked disk drives and heavier hardware.

Hobbyworld Electronics, 19511 Business Center Dr., Northridge, CA 91324. 800 423-5387, (800 382-3651 in CA). *COD ($1.25 extra), check, credit cards. No discounts, no insurance.*

Hobbyworld is the computer hobbyist's pop culture. It stocks all the hot items with a high turnover. Look to them for low prices on items you need right away.

Electrolabs, P. O. Box 6721, Stanford, CA 94305.

The best part is always their funny and schizophrenic catalog with an honest selection and a wealth of good information. For example . . . 'Save yourselve $6.75 and use a 25 cent transistor the next time you're looking for a temperature probe.' Also, 'TTL family rules of Incest are great.' The shipping was always prompt and the merchandise prime.

Not Recommended

Active Electronic Sales, P.O. Box 1035, Framingham, MA 01701. 617 879-0077. *Minimum $10, handling $2, check (wait to clear), money order. No discounts, no insurance.*

This group claims to be 'The World's Largest International Semiconductor Distributor', which implies lots of stock, in stock. No way. All my orders have been returned 25 percent filled, with 50 percent errors.

GLOSSARY

access
The operation of seeking, reading or writing data on a storage unit (in this case, the diskette).

access time
The time that elapses between any instruction being given to access some data and that data becoming available for use.

address
An identification (number, name, or label) for a location in which data is stored.

algorithm
A computational procedure.

alphanumeric (characters)
A generic term for numeric digits and alphabetic characters.

alphanumeric string
A group of characters which may include digits, alphabetic characters, punctuation characters and special characters, and may include spaces. (Note: a space is a 'character' to the computer, as it must generate a code for spaces as well as symbols.)

ASCII
Abbreviation for American Standard Code for Information Interchange. Pronounced: 'ass-key'. Usually refers to a standard method of encoding letter, numeral, symbol and special function characters, as used by the computer industry.

assembly language
A machine-oriented language for programming mnemonics and machine readable code from the mnemonics.

base
Quantity of characters for use in each of the digital positions of a numbering system.

base 2
The 'binary' numbering system consisting of more than one symbol, representing a sum, in which the individual quantity represented by each figure is based on a multiple of 2.

base 10
The 'decimal' numbering system — consisting of more than one symbol, representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 10.

base 16
The 'hexadecimal' numbering system — consisting or more than one symbol representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 16.

binary
See 'base 2'

bit
A single 'binary' digit whose value is 'zero' or 'one'.

Boolean
This word isn't really here (for you folks who paid attention to the general information section). A form of algebra applied to binary numbers which is similar in form to ordinary algebra. It is especially useful for logical analysis of binary numbers as used in computers.

'BOOT' — BOOTSTRAP
A machine language program file that is put onto every diskette by the 'FORMAT' routine. This routing is invoked when reset or power-on occurs. It automatically loads the necessary programs (SYS0/SYS) to cause the computer to respond to the DOS commands; i.e., the machine is 'BOOTSTRAPPED' or 'BOOTED' into operation.

buffer
A small area of memory used for the temporary storage of data to be processed.

buffer track
A track on a diskette used for the temporary storage of data or program material during a recovery process.

bug
A Software fault that results in the malfunction of a program. May also refer to hardware malfunctions.

byte
Eight 'bits'. A 'byte' may represent any numerical value between '0' and '255'.

command file
A file consisting of a list of commands, to be executed in sequence.

contiguous
Adjacent or adjoining.

control code
In programming, instructions which determine conditional jumps are often referred to as control instructions and the time sequence of execution of instructions is called the flow of control.

CRC error
Cyclic Redundancy Check. A means of checking for errors by using redundant information used primarily to check disk I/O while verifying.

data base
A collection of interrelated data stored together with controlled redundancy to serve one or more applications. The data are stored so that they are independent of programs which use the data. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within a data base. A system is said to contain a collection of data-based information if they are disjoint in structure.

data-base management system
The collection of software required for using a data base.

data element
Synonymous with 'data item' or 'field'

data type
The form in which data is stored; i.e., integer, single precision, double precision, 'alphanumeric' character strings or 'strings'.

DEC
Initials for Directory Entry Code.

decimal
See 'base 10'.

direct access
Retrieval or storage of data by a reference to its location on a disk, rather than relative to the previously retrieved or stored data.

DIRECT STATEMENT (IN FILE)
A program statement that exists in the disk file that is not assigned a line number.

DIRECTORY
A table giving the relationships between items of data. Sometimes a table or an index giving the addresses of data.

displacement
A specified number of sectors, at the top or beginning of the file, in which the 'bookkeeping' and file parameters are stored for later use by various program modules.

distributed free space
Space left empty at intervals in a data lay out to permit the possible insertion of new data.

double precision
A positive or negative numeric value, 16 digits in length, not including a decimal point (Example: 99999999999999.99).

DUMP
To transfer all or part of the contents of one section of computer memory or disk into another section, or to some other computer device.

embedded pointers
Pointers in the data records rather than in a directory.

entity
Something about which data is recorded.

EOF
Initials for 'end of file'. It is common practice to say that the EOF is record number nn or that the EOF is byte 15 of sector 12. Hence, it is a convenient term to use in describing the location of the last record or last byte in a file.

extent
A contiguous area of data storage.

file
A collection of related records treated as a unit; The word file is used in the general sense to mean any collection of informational items similar to one another in purpose, fφrm and content.

file parameters
The data that describes or defines the structure of the file.

FILESPEC
A file specification and may include the 'file name', the 'the file name extension', 'password', and 'disk drive' specification.

field
See 'data item'.

file area
The physical location of the file, on the disk, or in memory.

header record
A record containing common, constant or identifying information for a group of records which follow.

hexadecimal
See 'base 16'

index
A table used to determine the location of a record.

indirect addressing
Any method of specifying or locating a storage location, whereby, the key (of itself or through calculation) does not represent an address. For example, locating an address through indices.

INSTRING
Refers to the capability of locating a substring of characters that may exist in another character string. An example would be: Substring 'THE' String 'NOW IS THE TIME'. An INSTRING routine would locate the substring and return its starting position within that string. In this example, it would return a value of eight.

integer
A natural or whole number with no decimal point.

inverted file
A file structure which permits fast spontaneous searching for previous unspecified information. Independent lists or indices are maintained in records' keys which are accessible according to the values of specific fields.

inverted list
A list organized by a secondary key — not a primary key.

IPL
Initials for Initialize Program Loader; a program usually executed upon pressing of the 'RESET' button.

key
A data item used to identify or locate a record or other data grouping.

label
A set of symbols used to identify or describe an item, record, message or file. Occasionally, it may be the same as the address in storage.

least significant byte
The significant byte contributing the smallest quantity to the value of a numeral.

list
An ordered set of data items. A 'chain'.

load module
A program developed for loading into storage and being executed when control is passed to the program.

logical
An adjective describing the form of data organization, hardware or system that is perceived by an application program, programmer, or user; it may be different than the real (physical) form.

logical data-base description
A schema. A description of the overall data-base structure, as perceived for the users, which is employed by the data base management software.

logical file
A file as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

logical operator
A mathematical symbol that represents a mathematical process to be performed on an associated operand. Such operators are 'AND', 'OR', 'NOT', 'AND NOT' and 'OR NOT'.

logical record
A record or data item as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

LSB
See least significant byte.

machine-language
See assembly language.

maintenance of a file
(1) The addition, deletion, changing or updating of records in the database.
(2) Periodic reorganization of a file to better accommodate items that have been added.

monitor
A program that may supervise the operation of another program for operation or debugging or other purposes.

most significant byte
The significant byte contributing the greatest quantity to the value of a numeral.

MSB
See most significant byte.

multiple-key retrieval
Retrieval which requires searches of data based on the values of several key fields (some or all of which are secondary keys).

nibble
The four right most or left most binary digits of a byte.

null
An absence of information as contrasted with zero or blank for the presence of no information.

on-line
An on-line system is one in which the input data enter the computer directly from their point of origin, and/or output data are transmitted directly to where they are used. The intermediate stages such as writing tape, loading disks or off-line printing are avoided.

on-line storage
Storage devices and especially the storage media which they contain under the direct control of a computing system, not off-line or in a volume library.

operating system
Software which enables a computer to supervise its own operations, automatically calling in programs, routines, language and data as needed for continuous throughput of different types of jobs.

parity
Parity relates to the maintenance of a sameness of level or count, i.e., keeping the same number of binary ones in a computer word to thus be able to perform a check bsed on an even or odd number for all words under examination.

pointer
The address or a record (or other data groupings) contained in another record so that a program may access the former record when it has retrieved the latter record. The address can be absolute, relative, symbolic, hence, the pointer is referred to as absolute, relative, or symbolic.

primary entry
The main entry made to the directory.

random access
To obtain data directly from any storage location regardless of its position, with respect to the previously referenced information. Also called 'direct access'.

random access storage
A storage technique in which the time required to obtain information is independent of the location of the information most recently obtained.

read
To accept or copy information or data from input devices or a memory register; i.e., to read out, to read in.

record
A group of related fields of information treated as a unit by an application program.

relational operator
A mathematical symbol that represents a mathematical process to perform a comparison describing the relationship between two values (e.g. < *less than*... > *greater than*... *equal*... and combinations thereof).

search
To examine a series of items for any that have a desired property or properties.

secondary index
An index composed of secondary keys rather than primary keys.

sector
The smallest addressable portion of storage on a diskette.

seek
To position the access mechanism of a direct-access storage device at a specified location.

sequential access
Access in which records must be read serially or sequentially one after the other; i.e., ASCII files, tape.

single precision
A positive or negative numerical value of 6 digits in length, not including a decimal point (Example: 99999.9).

sort
To arrange a file or data in a sequence by a specified key (may be alphabetic or numeric and in descending or ascending order).

source code
The text from which executable code is derived.

system file
A program used by the operating system to manage the executing program and/or the computer's resources.

sub-strings
See INSTRING

table
A collection of data suitable for quick reference, each item being uniquely identified either by a label or its relative position.

token
A one byte code representing a larger word consisting of 2 or more characters.

track
The circular recording surface transcribed by a read/write head on the disk.

transaction
An input record applied to an established file. The input record describes some 'event' that will either cause a new file record to be generated, an existing record to be changed or an existing record to be deleted.

transparent
Complexities that are hidden from the programmers or users (made transparent to them) by the software.

vector
A line representing the properties of magnitude and direction. Since such a 'line' can be described in mathematical terms, a mathematical description (expressed in numbers, of course) of a given 'direction' and 'magnitude' is referred to as a 'vector'.

verify
To check a data transfer or transcription.

working storage
A portion of storage, usually computer main memory, reserved for the temporary results of operations.

write
To record information on a storage device.

zap
To change a byte or bytes of data in memory on on diskette by using a software utility program.

Computer Books and Software from IJG

Microsoft trademark Microsoft Corporation
TRS-80 trademark TANDY Corporation
Apple trademark Apple Computer Inc.
Electric Pencil © 1981 Michael Shrayer

Prices Subject to change without notice

BOOKS

TRS-80 Disk & Other Mysteries. H. C. Pennington. The "How to" book of Data Recovery. 128 pages. **$22.50**

Microsoft Basic Decoded & Other Mysteries. James Farvour. The Complete Guide to Level II Operating Systems & BASIC. 312 pages. **$29.95**

The Custom TRS-80 & Other Mysteries. Dennis Bathory Kitsz. The Complete Guide to Customizing TRS-80 Software & Hardware. 336 pages. **$29.95**

BASIC Faster & Better & Other Mysteries. Lewis Rosenfelder. The Complete Guide to BASIC Programming Tricks & Techniques. 290 pages.... **$29.95**

Electric Pencil Operators Manual. Michael Shrayer. Electric Pencil Word Processing System Manual. 123 pages. **$24.95**

The Custom Apple. Winfried Hofacker & Ekkehard Floegel. The complete guide to customizing the Apple Software and Hardware. Available July 1982..... **$24.95**

SOFTWARE

BFBDEM. Lewis Rosenfelder. Basic Faster & Better Demonstration Disk. 121 Functions, Subrountines & User Rountines For the TRS-80 Model I & II. Available in DISK ONLY...................... **$19.95**

BFBLIB. Lewis Rosenfelder. Basic Faster & Better Library Disk. 32 Demonstration Programs. Basic Overlays. Video Handlers. Sorts & more for the Model I & II. Available in DISK ONLY **$19.95**

Electric Pencil. Michael Shrayer. Word Processing System. Available in DISK **$89.95**
STRINGY FLOPPY or CASSETTE............... **$79.95**

Red Pencil. Automatic Spelling Correction Program. For use with the Electric Pencil Word Processing System. Available in DISK ONLY................. **$89.95**

Blue Pencil. Dictionary – Proofing Program. For use with the Electric Pencil Word Processing System. Available in DISK ONLY.................. **$89.95**

Add $4.00 shipping and handling charge per item.
California residents add 6% sales tax. Canadian residents add 20% for exchange rate.

IJG Inc. • 1953 West 11th Street • Upland, California 91786 USA • (714) 946-5805 the custom apple® & OTHER MYSTERIES Hofacker · Floegel APPLE INFORMATION SERIES - VOLUME I

iJG

ISBN 0 936200 05 7