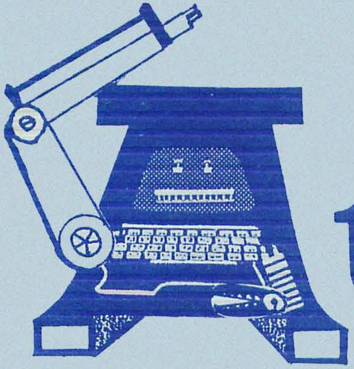
Your Atari Comes Alive



By
Richard Leinecker and
David Burd

ALPHA SYSTEMS

Your Atari Comes Alive

By
Richard Leinecker and
David Burd

ALPHA SYSTEMS

Table of Contents

Preface

The purpose of this book is to enhance the understanding of Atari home computers by providing instructions for hardware projects which illustrate general concepts and give examples of the related programming techniques. While these projects are simple enough for the novice to understand, they can easily be elaborated upon to create much more complex devices.

We are planning other publications as well in the future edition of this book. Your feedback is actively encouraged in order to fulfill your needs and reflect your ideas. The best way to reach us is to call our BBS numbers listed below:

The Burdsnest BBS
305-233-2873

While reasonable care has been exercised with regard to the accuracy of this book, the authors assume no responsibility for errors, omissions, or suitability for any applications. Neither do we assume any liability for any damage resulting from the use of this book.

Chapter 1
Electronics Review

Introduction

This section may contain a lot of information you already know. Advanced users can use this section for reference and review. Beginners will probably want to read it carefully before starting on the projects.

Before you begin a finished product, it's a good idea to assemble the project on a solderless breadboard to iron out any bugs, or make any necessary modifications. It's a relatively quick and easy method of putting together projects, and you can use the same parts for your permanent project.

About Boards

To use a solderless breadboard, simply insert the components into the holes, which have underlying conductor strips. Perforated breadboards are good for the final version of your creations. The components can be soldered or wire wrapped.

A breadboard with about 90 - 100 terminal points will be big enough for most projects in this book. If you need more space for a large project, such as the phone answering circuit, just put two together. You can use a jumper wire kit, or just strip the ends of your wires.

Before you begin construction on your final version, sketch the layout and pencil in the planned component locations. This way you can plan a neat, compact board. If you plan to solder the components, make sure you have sockets for the ICs and transistors.

Soldering

When you solder, make sure all surfaces are clean. Do not use wire with an oxidized metal end. If you do, you must strip the end. Use small rosin core solder, and a soldering iron between 15 and 35 watts. When soldering semi-conductors (diodes, LEDs, transistors, etc.) make sure all leads have a heat sink attached before soldering, and limit heating time to 10 seconds or less.

If you want to use an etched PC board, you can get the necessary parts from the suppliers listed in Appendix A, or from Radio Shack. For a really professional job, you can design the artwork, and have one of the companies in Appendix A make the PC board for you.

Resistors

Resistors limit the amount of current in a circuit branch. You can use Ohm's law to calculate the correct resistance needed for the job:

$$E=I \times R \qquad I=E/R \qquad R=E/I$$

E is the voltage (in volts), I is the current (in amps), and R is the resistance (in ohms).

For example, suppose a component requires 5 volts, draws 2 amps of current, and the power source is 12 volts. You can find the correct resistor using Ohms Law R=E/I, where R = 7 volts/2 amps. In this case, R=3.5 ohms.

Most circuits in this book need resistors with a 1/4 watt rating. If you think the resistor will be exposed to more wattage, you can calculate the wattage using one of these formulas:

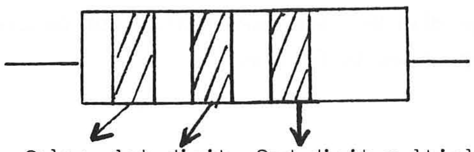
P=E x I   P=I2 x R   P=E2/R

To calculate the voltage drop for a device such as an LED, use the formula:

R = (Vs-Vd)/Id

Here Vs is the voltage source, Vd is the correct device voltage, Id is the current the device draws, and R is the necessary resistance.

Most resistors are color coded as follows:

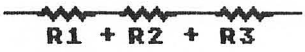


Color	1st digit	2nd digit	multiplier
Black	0	0	1
Brown	1	1	10
Red	2	2	100
Orange	3	3	1,000
Yellow	4	4	10,000
Green	5	5	100,000
Blue	6	6	1,000,000
Violet	7	7	10,000,000
Gray	8	8	100,000,000
White	9	9	none

A fourth band may or may not be present. If not, the tolerance is +20%. A silver band indicates a tolerance of +10%, and a gold band indicates a +5% tolerance.

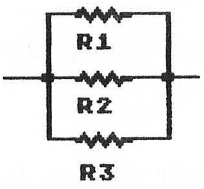
"K" means thousand, so a 5K resistor is really 5,000 ohms. Meg means 1,000,000 when used with resistors, so a 2 megaohm resistor has a resistance of 2,000,000 ohms.

When resistors are used in a series, as below, simply add their values to find the total resistance.



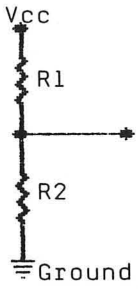
R1 + R2 + R3 = Total Resistance

When resistors are used in parallel, as below, you can find the total resistance with the formula 1/R1 + 1/R2 + 1/R3 = 1/Rtotal



Voltage Dividers

In this book, voltage dividers are used mainly for paddle inputs. A voltage divider works as follows:



$$V\ Out = \frac{R2}{R1+R2} \times Vcc$$

Capacitors

Capacitors are used in this book either to pass AC signals and block the DC current, to regulate voltage fluctuations, or to perform a timing function.

When you pass an AC signal and block the DC current, you must make sure the AC signal is not attenuated. You can do this by calculating the capacitance value. The capacitance value is based on the lowest frequency you need to pass. You can find this value with the following formula, where C is the capacitance in farads, and f is the frequency in cycles per second.

$$C=1/2 \times 3.14 \times f \times 1,000$$

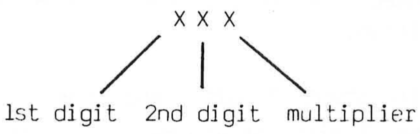
If the capacitor will be used to filter power fluctuations, just use the same formula to calculate the capacitance value. For half-wave recitifiers, the frequency is 60, for full wave, 120.

Capacitors don't charge and discharge immediately. To find out about how much time it will take, use the following formula. The resistance value is in ohms, the capacitance value is in farads, and the time is in seconds.

$$\text{Time} = \text{Resistance} \times \text{Capacitance}$$

In our schematics, if a capacitor has a polarity marking such as +-¶¶-, then it is an electrolytic capacitor, and care must be taken to observe the polarity when constructing the circuit. Both the polarity and value are usually written on the case, along with a maximum voltage. Do not exceed the maximum value, or you will damage the capacitor.

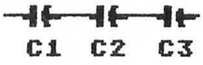
The other capacitors we use are disk capacitors and the values marked on them can be interpreted as follows:



Read the first and second digits as they appear. To get the value in picofarads, simply move the decimal point to the right the number of places indicated by the multiplier. A picofarad is a millionth of a millionth of a farad. To convert from picofarads to microfarads, move the decimal six places to the left. For most of these applications, you can be as far off as 20% and still be OK.

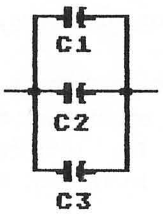
To find the total capacitance value of capacitors in a series, add their values together.

C1+C2+C3=Ctotal



To find the total value for capacitors in parallel, use this formula:

1/C1 + 1/C2 + 1/C3 = 1/Ctotal



Diodes

Diodes let a current flow in only one direction. We used two kinds of diodes in this book; recitifier diodes and light emitting diodes (LEDs).

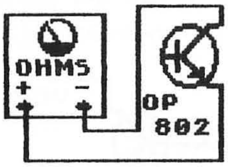
Recitifier diodes will have a dark band at one end. This is the cathode lead, and should be connected to the negative node in the current. This is called forward biasing. LEDs have a shorter lead, which is the cathode, and a longer one, which is the anode. Connect the shorter lead to negative, and the longer lead to positive.

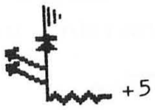
LEDs usually require about 2 volts and draw about .02 amps. Don't exceed these limits, or you may damage the LEDs. If your power source is greater than 2 volts, use Ohm's Law to find the correct resistor to lower the voltage.

For example, if your source is 5 volts, first find the voltage drop you need: 5 (source voltage) - 2 (needed voltage) = 3 (needed voltage drop). The LED will draw about .02 amps, so your formula will be:

R=E/I, or R=3/.02, so R=1500 ohms

Infrared LEDs are biased the same way. Check the manufacturers current specifications to find the right resistor. You can test infrared LEDs with this setup.

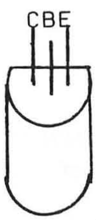


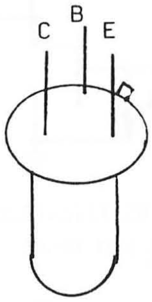


Transistors

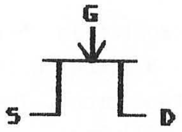
A single, general purpose transistor is very flexible. It takes up less space than an IC, and may be more practical when only one gate is needed.

Bipolar transistors (NPN or PNP) usually have 3 leads: the emitter (E), the base (B), and the collector (C). The diagram below shows the transistor from the bottom, with the leads pointing up. Transistors in a plastic package have one flat side to help you identify the leads. Those in a metal case have a small tab.

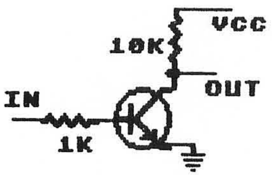


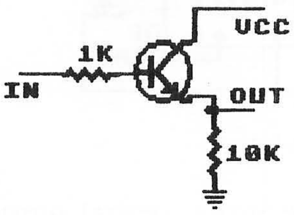


FET (or JEET) transistors don't have a standard format. Their leads are source (S), gate (G), and drain (D), as shown.



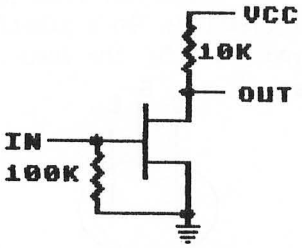
Transistor switches are small and compact. Even if you combine several switches, they are still smaller than many ICs. A transistor switch is either ON or OFF. They are biased so only two regions of the load line, cutoff and saturation, are used. Below are schematics for two simple NPN transistor switches. One is inverting, the other is non-inverting.



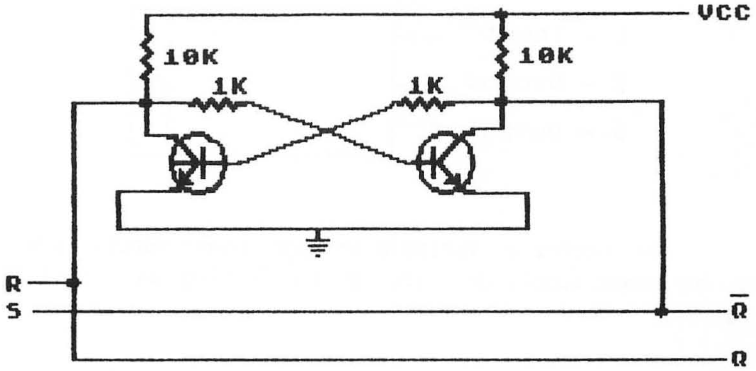


Inverting   Non-Inverting

You may want to use an FET switch instead. The input resistance is greater, and it won't load down previous stages as much. This is the schematic for an FET switch.



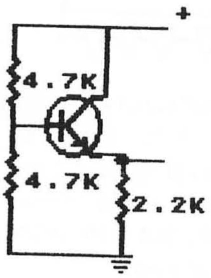
You can build an RS flip-flop with two NPN transistors. It can be used to set and reset the output of a device. The schematic looks like this.



Bipolar transistors can be used as voltage regulators. Use the formula below to bias the base for the necessary voltage.

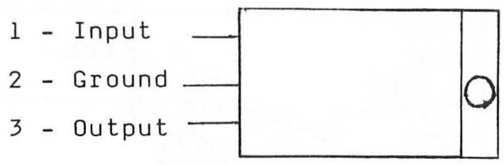
(R1/(R1 + R2)) x Vcc - .7 volts = Regulated Voltage

The schematic below shows a circuit that uses an NPN transistor to provide a regulated voltage output of 4.3 volts.



Voltage Regulator ICs

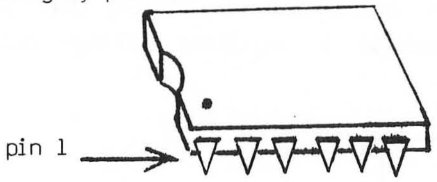
Fixed voltage regulators are ideal for adjusting a power supply to provide additional voltage, and for your own power supplies. Below is a diagram of a 7805 regulator, but it also applies to 7812 and 7815 regulators. A 7812 gives a regulated output of 12 volts, a 7815 gives 15 volts.



If you prefer a variable voltage power supply, you can use the power supply described in the Testing section of this chapter.

ICs

Integrated circuits are complete circuits in themselves, and all operating requirements must be observed. A notch or circle is near pin one. When the IC is right side up, pin 1 is at the lower left, as shown below. There will be a dot or other marking by pin 1.



Sometimes the date of manufacture is printed on the IC along with the part number. It's best to keep all ICs separate and labeled. When building an IC project (except on a solderless breadboard), always use a socket. Sockets with gold leads are easier to solder, but tin leads are fine, too. Always connect unused pins to ground.

Three kinds of ICs are used in this book; TTL, CMOS, and linear. TTL means Transistor-Transistor Logic. In some places, we used LS versions of TTL ICs. LS means low power schottky, and refers to the way the chip was made. CMOS means

Complimentary Metal Oxide Semi-Conductor. Linear means the IC has output that's proportional to the input. Some linear ICs can be used non-linearly.

TTL ICs have some operating requirements. They are:

1. Vcc must not exceed 5.25 volts.

2. Input signals must not fall below ground.

3. Unused inputs usually assume the high state. If you want it to be high, connect it to Vcc.

4. If an input should be in the low state, connect it to ground.

5. Connect unused inputs to Vcc, to prevent unnecessarily high current use.

6. Avoid excessively long connecting wires.

One TTL output will drive up to 10 TTL gates, or 20 LS gates. One LS output will drive up to 5 TTL gates, or 10 LS gates.

If your circuit doesn't work right, check these things:

1. All pins go somewhere.

2. All IC pins are in the socket.

3. All operating requirements are met.

4. All connections have been made.

5. Vcc is no more than 5.25 and no less than 4.75 volts.

CMOS ICs use less current than TTLs, and operate on a 3 - 5 volt range. They also have some operating requirements. They are:

1. The input to any pin should never exceed the Vcc (except for 4049 and 4050 ICs).

2. All unused inputs must be connected to the Vcc or ground.

3. Do not connect a signal when Vcc is off.

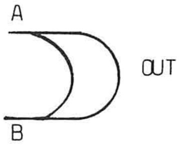
4. Avoid static at all costs. These chips are very sensitive, and easily destroyed. Always follow these precautions:
   A) Always store them in a conductive container.
   B) Never set them on a non-conductive surface.
   C) Make sure you have no static buildup when you touch them.

5. When they are interfaced with TTL, make sure the Vcc is at least 5 volts.

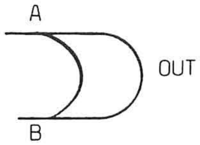
Logic Gates and Truth Tables

If you want to design your own logic gate for input to or output from the computer, you can use the following basic logic gates and their truth tables. L means low, and corresponds to a binary 0. H means high, or a binary 1.

AND GATE



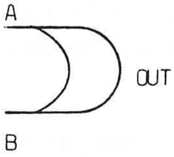
NAND GATE



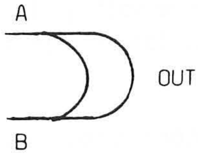
A	B	OUT
L	L	L
L	H	L
H	L	L
H	H	H

A	B	OUT
L	L	H
L	H	L
H	L	L
H	H	L

OR GATE



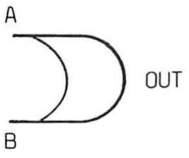
NOR GATE



A	B	OUT
L	L	L
L	H	H
H	L	H
H	H	H

A	B	OUT
L	L	H
L	H	L
H	L	L
H	H	L

EXCLUSIVE OR GATE

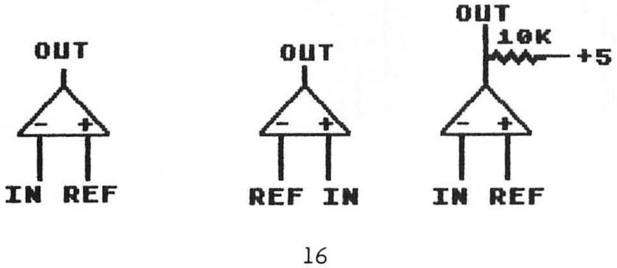


A	B	OUT
L	L	L
L	H	H
H	L	H
H	H	L

Voltage Comparators

Voltage Comparators (specifically the LM339, which we used frequently) compare a reference voltage with another voltage. Usually the reference voltage is set up with fixed resistors, but it's also possible for it to float. The basic configurations are below.

A non-inverting comparators output will be low if the input voltage (pins 5, 7, 9, or 11) falls below the reference voltage (pins 4, 6, 8, or 10). With an inverting comparator, the output will be low when the input voltage exceeds the reference voltage.



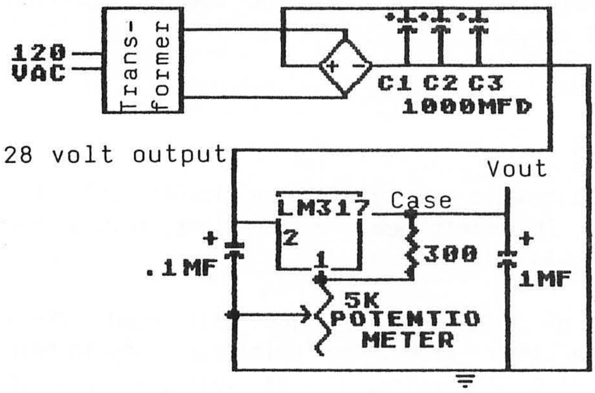
For TTL compatible output, connect a 10K resistor from the output to +5 volts.

Testing

When you've assembled your preliminary circuit, it's time to test it. If it doesn't work quite right, review the previous sections in this chapter.

Two pieces of test equipment you will need are a multimeter and a logic probe. The multimeter should have voltage ranges of 0 - 5 volts, 0 - 25 volts, and 0 - 125 volts. It should have current ranges of 0 - 100 mA and 0 - 500 mA. A current range of up to one amp is sometimes helpful, but not necessary. You should be able to measure resistances from 0 - 6 megaohms, and have an accuracy within 5%. The logic probe you choose should be able to handle both TTL and CMOS levels, indicate pulses, and protect against reversed polarity.

One more piece of equipment that will make your testing easier is a power supply. A combination 5 volt/12 volt supply is fine for the projects in this book, but a variable range power supply will give you more flexibility. Your power supply should be able to handle at least one amp. If you prefer to build your own, we've provided the schematic for a variable power supply. If you like, you could put two separate packages together for two separate voltage levels.



120 VAC
Trans- former
C1 C2 C3
1000MFD
28 volt output
Vout
LM317
Case
2
1
300
+
.1MF
+
1MF
5K POTENTIO METER

Chapter 2
Software Techniques & Comments

The first section of this chapter is a brief review of binary and hexidecimal numbers. The remaining sections explain the programming techniques used in the programs we've provided, along with some tips to help you write your own programs and machine language subroutines.

Binary and Hexidecimal Numbers

A computer is really a complex set of switches, or bits. Each bit can be either On or Off. Computers use a number system called binary, or base 2, to perform calculations. Binary has two digits, 0 and 1. If a switch, or bit, is Off, the computer assigns that bit a value of 0. If it's on, then the computer assigns it a value of 1. As you can see, binary is a perfect number system for the computer to use.

In binary, the number 0 is simply 0. The number 1 is simply 1. But how do you represent a '2' in binary, when the binary system doesn't have a digit for 2? Think of our normal base 10 decimal system, with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. How do you represent 9+1, when there is no digit for that number? In decimal, 9+1 is represented with a '0' in the ones place, and a 1 in the tens place (10). A 2 in binary is represented in the same way. We put a 0 in the ones place, and a 1 in the next place up. In decimal, it is the tens place, but in binary, it's the twos place. So 2 in binary is 10. Notice the pattern as the numbers increase in the chart of Decimal/Binary Numbers.

Decimal/Binary Numbers

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

To convert a binary number to decimal, simply multiply each digit by its place value, then add all the values up. For example, let's convert the binary number 1010. The place values and digits are:

	Place Value	8	4	2	1
times	Binary Number	1	0	1	0
	Equals	8 +	0 +	2 +	0 = 10

Below is a chart of Place Values. The binary digits are represented by xs.

Place Value	128	64	32	16	8	4	2	1
Binary Digits	x	x	x	x	x	x	x	x

Computers also use the hexidecimal number system, or base 16. This number system has 16 digits. The next chart shows some smaller binary, hexidecimal, and decimal numbers.

Binary/Decimal/Hexidecimal Numbers

Binary	Decimal	Hexidecimal
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
10001	17	11

If you can convert binary numbers to decimal numbers, you can convert hexidecimal numbers to decimal numbers. Convert them the same way, multiplying each digit by its place value, then adding them up. Use the place value chart below, and remember the decimal values for the alphabetic digits, boxed in on the chart above.

Place Value	19456	256	16	1
Hexidecimal Digits	x	x	x	x

Example:

Hex number 10C	1	0	C
times the value	x256	x 16	x1
Equals	256 +	0 +	12 = 268

Memory locations

The contents of certain memory locations are used to control your devices and signal the computer. The BASIC command PEEK enables you to look at the contents of a particular memory location. POKE enables you to put any value you want in a location. Below is a list of some important memory locations.

Memory Locations

18, 19, and 20	The Internal Real Time Clock
82	Controls the left margin
83	Controls the right margin
195	The error number that occurred last. When you set up a TRAP in your BASIC program, you can look here to find the exact error.
203-206	A series of 4 free bytes. We often use them for short machine language routines.
546-547	Immediate VBI vector
548-549	Deferred VBI vector

564	Light Pen Horizontal value
565	Light Pen Vertical value
624-631	Paddle registers. XL and XE computers use only the first 4 locations.
632-635	Stick shadow registers. XLs and XEs use the first four only.
644-647	Trigger registers. XLs and XEs use only the first two.
710	Screen color in Graphics 0
741, 742	Pointer to the top of free memory
752	Cursor inhibit flag. Zero turns the cursor on, 1 turns it off
764	The value of the last key pressed. 255 means all keypresses have been processed.
1536 to 1791	Page six. The OS doesn't use it. This is a good place for ML routines or data.
54016 and 54018	Used to configure the joystick ports. These locations are explained in Chapter 6.

For more information about memory locations, refer to Mapping the Atari from COMPUTE! Publications.

Graphics Modes

In most of our demonstration programs, we used graphics mode 0 for text and 7 for pictorial displays. The chart below shows the various graphics modes, available colors, and resolutions.

Mode	Columns	Rows	# of Colors
0	40	24	1
1	20	24	5
2	20	12	5
3	40	24	4
4	80	48	2
5	80	48	4
6	160	96	4
7	160	96	4
8	360	192	1

The USR function

Calling a Machine Language Routine from BASIC

The USR function is a command that calls a machine language (ML) subroutine from BASIC. BASIC is a slow language, but easy to program. Machine language is fast, but harder to program. You can get the speed of machine language by using ML subroutines for complex calculations where speed is crucial. To write your own ML routine, first write the source code. You can use any language you have a compiler for, such as PASCAL or ACTION. Then assemble your code. An assembler puts the routine into binary file form. Save each routine you write, in case you need it again for another program.

If you don't have the time to write, assemble, and debug your own routines, you can use pre-written routines.

Professionally written routines are fast, efficient, compact, and best of all, bug free. BASIC Turbocharger, from Alpha Systems, is a good source of routines. It contains over 120 written, tested, and debugged routines. All you have to do is ENTER them into your BASIC program.

There's three ways to call an ML subroutine from a BASIC program. You can load the routine into a specific spot in memory, and call it by using that address. Or, you can put the routine into a string, and call it with the address of the string. Lastly, you can put the code into a string, and put the string right inside your program.

If you decide to call the routine from a specific address, the first step is to load the routine into the proper place in memory. Page six (locations 1536 - 1791) is a convenient spot. There's two easy ways to load your routine in. The first way is to rename the routine AUTORUN.SYS. The second is to use data statements, and POKE the routine into place. When you're ready to use the routine, you can call it with the statement X=USR(location). X is a dummy variable, it's simply a place holder. Location is the starting address of the routine.

When an ML routine is put into a string variable, each consecutive byte of the routines' code is assigned to a consecutive byte of the string variable. When you're ready to use the routine, you can call it with the command X=USR(ADDR(A$)). Once again, X is a dummy variable. A$ is the string variable that contains the routine.

A machine language routine can also be put inside the USR command itself. When you need to use the routine, simply type X=USR(ADDR(Routine)). The routine is entered in control characters.

You can also pass values to and from your routine. These values are called arguments. The routine may need them for reference, or it can perform calculations with them. To pass values, put their variable names in the USR statement, for example X=USR(location, VALUE1, VALUE2, ...).

If you decide to write your own routines, you'll need to understand how the computer passes these values. When a BASIC program calls an ML routine, the computer puts the important addresses and values in a place called the stack. The first two bytes it puts in the stack are the address of the current program, so the computer will know where to go when it's finished the routine.

Then it puts the arguments, if any, on the stack. The values are always 2 bytes long, with the most significant byte first, followed by the least significant byte. The last thing on the stack is a single byte, which contains the number of arguments passed. If you didn't pass any arguments, the byte will be a 0.

When the computer is finished, the result is literally a stack of numbers. The number of arguments is on top, followed by the arguments themselves, in descending order, with the address to return to on the bottom. To get to any particular number in the stack, you must first pull off everything on top of it.

OF VALUES
VALUE #3
VALUE #2
VALUE #1
ADRESS

When you pull a value off the stack, it comes off most significant byte, followed by least significant byte. If you didn't pass any values, don't forget to pull off the single byte on top, which indicates the number of arguments. If you don't, the computer will interpret that byte as part of the address it's supposed to return to.

If you want to learn more about machine language routines, refer to one of the many excellent books available on the topic. BASIC Turbocharger, from Alpha Systems, is a good one. It contains over 120 pre-written routines you can study as examples, or modify and experiment with.

The program USRMAKER, on the disk, will take a binary load file and put it into USR format. To use this program, first write and assemble your routine. Then run USRMAKER. The computer will ask you to enter the drive number and the filename of your routine. USRMAKER will load in your binary file, and ask what form you want the data to be put in. The three choices are 1) Address of a string of characters, 2) Data statements, and 3) String variable. If your routine contains a 155, you must use data statements. Once you enter the number you want, USRMAKER will put the routine into the specified format, enter the lines in memory in BASIC, then delete itself. You can use the subroutine in any program you like. You may have to change the line numbers and list the lines to disk, so they can be entered into your program.

Chapter 3
Display Memory & Display Lists

The visual display you see on your Atari screen is controlled by the ANTIC chip. The Display List is the data the ANTIC chip uses to set up the screen. When the ANTIC chip updates the screen, it stops the 6502 processor, gets the display list data, then restarts the 6502. The higher the graphics mode, the longer it takes the ANTIC chip to update the screen. To get maximum speed for heavy duty number crunching, you can turn off the ANTIC chip and the screen display by using location 559 ($22F). This way, the 6502 can work uninterrupted.

The display list is a set of numbers that designate the graphics for each line on the screen. Each graphics mode has its own display list number.

When you begin experimenting with display lists, the first thing you must do is find the starting address of the display list. The address and the length of the list are different for each graphics mode. You can find the current address in locations 560 and 561. Next is a sample display list.

CODE	Represents	Comments
112	Blank Line	These lines
112	Blank Line	prevent
112	Blank Line	overscan
66	64+2	64 - next 2 bytes are adr of screen memory, 2 is graphics mode
XXXX	LSB/MSB adr	
XXXX	of screen memory	
next 23 lines		
2	23 lines of gr mode 2	
65	64+1	64 - next 2 bytes are adr of next DL 1 - wait for VBI
XXXX	LSB/MSB adr	
XXXX	of screen memory	

When you write your own ML routines that use a graphics mode other than 0, you have to modify the pointers to the display list (the pointers are 560 and 561). We change graphics modes in machine language by copying the display list from BASIC after issuing the GRAPHICS command, and then poking the values into memory.

An essential part of the screen display is display memory. As you can see from the sample display list, the fifth and sixth bytes of the display list are the pointers to display memory. These two bytes are in LSB/MSB form.

Basically, display memory is just a list of numbers that tell the ANTIC chip what to display on the screen. In most instances, display memory for graphics mode 0 begins at 40000. The following BASIC commands will put the address of display memory into the variable DISPLAY.

```
DL=PEEK(560)+PEEK(561)*256
     DISPLAY=PEEK(DL+4)+PEEK(DL+5)*256
```

These lines can be entered in direct mode in BASIC. The variable DISPLAY will have the beginning address of display memory. Display memory starts with the upper-left corner of the screen. The lower-right corner of the screen is the last byte of display memory. Each position on the screen has a corresponding location in display memory. In graphics mode 0, the end of display memory would be at 40960 (assuming the beginning is at 40000). You can put characters directly on the screen by poking display memory with the internal value of the character. The internal value is not ASCII or ATASCII, but the actual value of the character the computer uses. "Mapping the Atari" by Compute! Books has a complete list of the ASCII, ATASCII, and internal values of each character.

Now, let's look at how the 6502, ANTIC, and C/GTIA chips all work together. First, the ANTIC chip interrupts the 6502 to get information from the display list and display memory. Then the 6502 goes back to work. ANTIC checks the graphics mode and sends that information, as well as the actual data to be displayed, to the C/GTIA chip. The C/GTIA chip then converts the information from the ANTIC chip into signals that your TV/Monitor can display. The whole process repeats every 1/60th of a second.

The program called DDISPLAY, on the disk, will show you some information about the display list currently in memory. There's a short BASIC routine described at the end of Chapter 5, Overlays, that will convert ATASCII values into internal values for you.

Chapter 4 Display List & Vertical Blank Interrupts

One small problem with programming in BASIC is that a status or operation is only executed when that commmand is encountered in a program line. When something needs constant attention, there is no real way to do it in BASIC, because the language is so slow. You can solve this problem by using a DLI (display list interrupt) or a VBI (vertical blank interrupt).

Display List Interrupts

As you remember, the display list is composed of a series of numbers that tell the ANTIC chip what should be displayed on each line. The monitor draws the pictures on the screen with an electron gun, inside the monitor. The gun draws the picture, one line at a time, starting in the upper left hand corner of the screen. When it reaches the far right edge of a line, it pauses to stay in sync, then returns to the left side of the screen.

The time between the moment the electron gun finishes one line and begins the next is when a DLI routine is executed. A DLI routine is run every 1/60th of a second, every time the screen is updated. The DLI is limited to about 36 machine cycles, so only short routines should be put here.

The whole DLI process begins when the ANTIC chip finds a byte in the display list which has the seventh bit set to 1. When the ANTIC chip reaches the end of that display line, it checks location 54286 ($D40E). If bit 7 of location 54286 is also a 1, the DLI routine is run. If bit 7 of location 54286 is a 0, then the ANTIC chip simply continues with screen update.

To perform a DLI, the computer looks at the pointer in locations 512 and 513 ($200;$201), in LSB/MSB format. Then it jumps to that address, and performs the DLI routine. When it's finished, it goes back to the main program.

Your DLI routine must save all the 6502 registers before it does anything. You can do this by using this routine:

```
PHA  * Push the accumulator
TXA  * X register to the accumulator
PHA  * Push the accumulator
        (X register value)
TYA  * Y register to the accumulator
PHA  * Push the accumulator
        (Y register value)
```

Now all the registers are saved, and you can begin your routine. If you don't use the registers in your routine, you don't need to save them to the stack.

Your DLI can check or change almost anything. DLIs are often used to change the graphics that are displayed on the screen. When your routine is finished, you must restore everything to it's original state. You can use this routine to do it.

```
PLA  * Get the Y register value
TAY  * Put it into the Y register
PLA  * Get the X register value
TAX  * Put it into the X register
PLA  * Get the original contents of
        the accumulator
```

Your DLI must end with an RTI, which returns control to the main program.

To set up your DLI, first load it into memory, then set locations 512 and 513 ($200,$201) to point to the address of your routine. Next you must choose the display list byte to alter. You can use any byte except the pointers to screen memory. Remember, your DLI will execute after the electron gun has finished drawing the line with the altered byte. To set up this byte, which is called a flag, just change the seventh bit to 1. You can do this by adding 128 to the value of the byte. To start the DLI, store a 192 in location 54286. If location 54286 is not set, then the ANTIC chip will think the flag was not a flag at all, just another piece of screen data. If you've written your DLI routine correctly, everything will function normally with your routine running.

VERTICAL BLANK INTERRUPTS

The vertical blank interrupt is executed in the time it takes the electron gun to go from the bottom-right corner back up to the upper-left corner. It's still executed once every 1/60th of a second, every time the screen is redrawn. The VBI offers more time for a routine to run than a DLI.

There's two kinds of VBIs. One type of VBI, an immediate VBI, allows your routine to be about 2,000 machine cycles, the other kind, a deferred VBI, allows about 20,000. An immediate VBI runs from start to finish during the time it takes the electron gun to travel back to it's beginning position. This allows screen parameters to be changed, and show up as a whole new screen. A deferred VBI starts at the same time, but after about 2,000 machine cycles the screen redraw begins, even though your routine is still running. If you plan to use the VBI to update something on the screen, you should use an immediate VBI.

When you set up a VBI,the first thing to do is choose which type you want. If 2,000 cycles is enough time, then an immediate VBI is fine. Remember that longer routines steal more processing time from your main program. If speed is crucial, keep your VBIs short.

Once you have decided which type you want and what it will do, you have to write your routine. Then you have to load it into memory. You can put it anywhere in free memory, page six (1536;$600) is always a good place. If it's an immediate VBI, set the two locations 546 and 547 ($222 and $223) to point to your routine in MSB/LSB format. If your VBI routine is a deferred VBI, set the locations 548 and 549 ($224 and $225) to point to your routine, also in MSB/LSB format.

You must end your routine with a JMP. For an immediate VBI, JMP $E45F, for a deferred VBI, JMP $E462. If you want to increase the speed of your immediate VBI by jumping over the OS VBI, end it with a JMP $E462 instead.

When you are setting the pointers of the VBI to your routine, there is a very small chance that the interrupt will occur in between loading both pointer bytes. If that happened, the vector would not send the routine to the correct address, but instead send it somewhere into memory, and anything could happen. Fortunately, the Atari OS has a way to prevent this from occuring. The routine located at $E45C will store the pointers to your routine safely. To use this routine, load the X register with the high byte of the address of your routine, and load the Y register with the low byte. Then load the accumulator with a 6 for an immediate VBI, or a 7 for a deferred VBI. Now do a JSR $E45C. The pointers will be put in safely. This is a sample routine.

```
  LDY #LSB  * Load  the least significant byte of the  VBI
              address
  LDX #MSB  * Load  the  most significant byte of the  VBI
              address.
  LDA #6    * 6 is immediate, 7 is deferred
    (or 7)
  JMP $E45C * You'll  be returned here after the  pointer
              routine is done.
```

```
10 FOR X=1 to 11
20 READ A
30 POKE 1536,A:REM Put this short routine wherever you have
room, or put it in a string
40 NEXT X
50 X=USR(1536)
60 REM The VBI pointers are installed
100 DATA 104,160,LSB,162,MSB,169,MODE,32,92,228,96
```

Note that you must put your MSB/LSB pointers and the mode (6 or 7) in the DATA statement.

Chapter 5
Overlays

The overlay is a very useful tool, because it provides a space on the screen for displaying information, and it stays in the same place, regardless of how much the rest of the screen scrolls. It's great for displaying the status of registers, variables, or locations while you're programming.

As you remember, the display list is a list of bytes that indicate what to display on the screen. The 66 in the sample display list in the pevious chapter ment the next two bytes of the display list was the LSB/MSB address of display memory, and also to display a graphics mode 0 line.

An overlay is created by changing the address of certain parts of display memory. More than one place in memory can be used for display memory. Only the screen memory stored in locations 40000 and above are updated from the keyboard. You can create an overlay by placing a 66 in the display list where you want the overlay to be, and making the next two bytes the LSB/MSB address of the display memory for your overlay. If you put the display memory for your overlay lower than location 4000, your overlays display memory will not be affected by the keyboard. We usually put overlays at the top of the screen.

There's three overlay programs on the accompanying disk. OVRLAY1 and OVRLAY2 create overlays that monitor the joystick ports. OVERMAKE allows you to create your own custom overlays.

To run OVRLAY1 or OVRLAY2, rename the program AUTORUN.SYS and reboot the computer. OVRLAY1 displays four pieces of information. In the upper left hand corner of the screen, it shows the status of port 1 in binary form. If an individual bit is connected to ground, or receiving a TTL level low, a 1 will be displayed. If the bit is not connected to ground, or if it is receiving a TTL level high, a 0 is displayed. Beneath this number is its decimal value. In the upper right hand corner is the status of the trigger bit of port 1. Below that is the status of Paddle 0, labeled ENTER. If Paddle 0 is connected to ground, it will say ON. Otherwise, it will say OFF. To turn the overlay off, press START. To turn it back on again, press OPTION. All the overlay values are automatically updated with a VBI.

OVRLAY2 is designed to work with the data selector hardware described in this book. It displays the same information as OVRLAY1, plus some additional information for the data selector.

The program called OVERMAKE will make a custom overlay for you. When you run the program, it asks how many lines you want for the overlay, and if you want a line to separate the overlay from the rest of the screen. Then it asks where in memory you want to put your overlay data. Page six is fine, but only 6 text lines, or 5 text lines and the separating line, will fit. If you place it towards the top of BASIC, MEMTOP will automatically be adjusted. Then the program will ask for a string to be displayed on the overlay. If you don't want anything, just hit RETURN. If you do type in a string, the maximum number of characters is 40 per line.

Now OVERMAKE will create a BASIC routine that will generate your overlay, then delete itself. Type RUN, and your overlay will appear. You can change the line numbers of the overlay routine, LIST it to disk, and ENTER it into your own BASIC programs.

When you've set up your custom overlay, you may want to run the program DDISPLAY. It will give you all the pertinent information about your display list.

Your custom overlay uses information in internal values. These values are not the same as the ATASCII values usually used in BASIC programming. The following routine will change ATASCII values into internal values, to use with your custom overlay.

```
10 REM X is the ATASCII value
20 REM Y is the internal value
30 Z=0:IF X>128 THEN X=X-128:Z=128
40 IF X<32 THEN Y=X+64+Z:GOTO 70
50 IF X<96 THEN Y=X-32+Z:GOTO 70
60 Y=X+Z
70 REM Now do the next character, or insert this one into
memory
```

Chapter 6
The Joystick Ports

The older Atari 800 computers have four joystick ports across the front of the machine. Atari XL and XE computers have two joystick ports on the right side of the machine. When you program in BASIC, the ports are numbered beginning with 0. For an 800, you have ports 1, 2, 3, and 4, called ports 0, 1, 2, and 3 in BASIC. for an XL or XE, ports 1 and 2 are called 0 and 1 in BASIC.

The command X=STICK(0) will give you the value of joystick port 1. Below is a chart showing which number to use in BASIC to get the values from a port, and the registers for each port. You can read the hardware registers by PEEKing into that location. Note there are two paddle inputs for each port.

	Port #			
	1	2	3	4
BASIC NUMBER				
STICK (#)	0	1	2	3
STRIG(#)	0	1	2	3
PADDLE(#)	0,1	2,3	4,5	6,7

HARDWARE REGISTERS

Stick	632	633	634	635
Trigger	644	645	646	647
Paddles	270	272	274	276
	271	273	275	277

The Pins

Each joystick port has 9 pins. Below is a diagram of the pins' arrangement, and a chart showing the function of each pin.

Pin #	Function
1	Data Bit 0
2	Data Bit 1
3	Data Bit 2
4	Data Bit 3
5	Paddle A
6	Trigger
7	5 volts (safe to draw 300 ma)
8	Ground
9	Paddle B

Pins 1-4 are held to 5 volts by a 10K internal resistor, and are TTL compatible. Normally, these pins are set as 'read only'. You can also set them as I/O or write only.

Pins 5 and 9 are connected to separate internal analog to digital converters. These pins can read voltages from 0 to 5 volts, and all values in between.

Pin 6 corresponds to the trigger button.

Pin 7 is a 5 volt source. We used it to power several devices in this book. If you choose to use this as a power source, always draw less than 300 milliamps. This is a safe current level to draw, but avoid any greater drains.

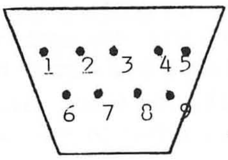
Pin 8 is ground. It must be connected to the ground of every device you plug into the port.

The Port

The connector that plugs into the joystick port is called a DB-9 female socket. The port is called a DB-9 plug. When you order the part to plug into the port, make sure you order a DB-9 Female Socket.

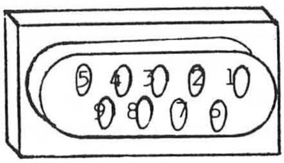
When you solder the DB-9 socket, keep track of the pins. It's easy to make the connections exactly backwards, since you'll be looking at the wrong side. The chart below will help you avoid confusion.

Pins on Computer

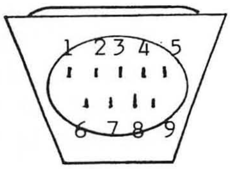


The port on the computer

9 Pin Socket



Front - Plugs into the computer



Back - Connect wires here

The plastic hoods on the DB-9 female socket may prevent the connector from plugging all the way in. The hoods are there to protect the soldered connections. If the socket won't fit all the way into the port, snip off the top and bottom plastic pieces which extend beyond the metal part of the DB-9. Install the screws that hold the hood together (but not the two that fit in the snipped off part). Then superglue the hood together, and hold it while it sets.

When you build your projects, <u>NEVER connect pin 7 (5 volts) directly to pin 8 (ground)</u>. This will draw more current than your system is capable of delivering, and may burn out your PIA chip. In some projects, we used resistors in between these two pins. This is acceptable, as long as the current is less than 300 milliamps. It is safe to draw anything under 300 milliamps.

Reading the pins

There's several ways to read the data from the joystick ports. You can also send data to your devices, through the joystick ports. To read pins 1-4 in BASIC, you can use the command X=STICK(#). The value you get will be between 0 and 15. When pins 1-4 are all high, or at 5 volts, the value will be 15. If they are all low, or connected to ground, the value will be 0. If pins 1-4 are in different states, the value will be somewhere between 0 and 15. You'll need to check each bit of the register to determine the state of each pin.

When you PEEK in location 54016, you'll get a binary number which is a combination of pins 1-4 in ports 0 and 1. On older 800s, location 54017 will give you the combined values of pins 1-4 in ports 2 and 3. This 8 bit number has one bit for each pin, as shown in the chart. The place values are used to set the pins as write only, described in the next section.

	Pin	Bit	Place Value
Port 1	1	0	1
	2	1	2
	3	2	4
	4	3	8
Port 2	1	4	16
	2	5	32
	3	6	64
	4	7	128

When the pins are in their normal, read only state, you can look in the hardware register and see a value of 0 or 1 for each pin. If the pin is connected to ground, or receiving a TTL low signal, it will have a value of 0. If it is not connected to ground, or if it is receiving a TTL high signal, it will have a value of 1. If the pins are set as I/O, they will be at a level approaching 0 for a TTL low signal, and at a level approaching 5 volts for a TTL high signal.

You can read the paddle pins 5 and 9 with the BASIC command X=PADDLE(#), or by PEEKing the hardware register. The values from these pins will be between 0 and 228, depending upon the voltage. There is a bug in the operating system that prevents the entire range of possible values (0-255) from appearing. There's more about the paddle pins in the Analog to Digital chapter in this book.

The BASIC command STRIG(#) will give you the value of pin 6, the trigger pin. If pin 6 is low, or connected to ground, the value will be 0. If it's high, or not connected to ground, it will have a value of 1.

Writing to the ports

Before you send data to your devices through the joystick ports, you must first set up the ports for output. You can write to pins 1-4 in any combination. Bits 1-4 of ports 1 and 2 are combined into a single 8 bit number. If you are using an older 800, pins 1-4 of ports 3 and 4 are also combined into a single number.

To configure the ports for output, first POKE 54018,56. Then decide which pins you'll be writing to. Now look at the chart in the previous section, and determine which bits you'll need, and the place values for those bits. Then add up the values. Now you have a total value.

For example, let's say you want to write to port 1, pins

1, 2, and 4, and port 2, pin 2. These are bit numbers 1, 2, 4, and 6. The place values for these bits are 1, 2, 8, and 32. Now add them up; 1+2+8+32=43. Your total value is 43. You could put it in a variable called BITS.

Now POKE your total value into 54016. In the example above, you'd POKE 54016,BITS. Then POKE 54018, 60. Those pins in the joystick ports are now ready to receive data, and send it along to your devices.

Chapter 7
Switches

Switches are the first devices we'll cover. Switches are fairly simple, but extremely useful. These switches will give you a chance to become familiar with the Process of building and programming devices. Many of the techniques used to build the switches are also used for the more complex projects in this book. Some of those projects use the switches described here, so if you build them first, you'll have a head start on some of the other devices.

First, we'll start with a brief review of the joystick ports, and then we'll explain how these switches work. Next is a parts list, and the assembly instructions for the switches. The last thing in this chapter is a discussion of the software, which lets you use these simple switches in many different ways.

Pins 1-4 of the joystick ports each have a corresponding data bit. The data bits for pins 1-4 of joystick ports 1 and 2 are combined into a single 8 bit number stored in location 54016. When any of these pins are low, or approaching ground, the pin will return a bit value of 0. If the pin is high, or approaching 5 volts, the bit value will be 1.

Inside the computer, pins 1-4 of each port are connected to 5 volts through a 10K resistor, When nothing is plugged into the joystick ports, the resistors hold the bits values to high. But, when the pins are connected to ground, the internal resistors have no effect on the values.

By connecting a switch between pins 1-4 and ground, you can control the state of the pins and their corresponding bit values. When the switch between a pin and ground is closed, making a connection to ground, the bit value for that pin will be 0. When the switch is open, and the connection is broken, the value will be 1.

Parts List

1 to 4 DB-9 Female Sockets (Hoods Optional)

1 to 4 Momentary, Normally Open, Pushbutton switches

Hookup wire - 26 gauge solid wire or telephone wire

Project Box (optional - for a finished look)

One of the switch programs described in this chapter will work with only one switch, connected to pin 1. The other two programs require at least two switches, connected to pins 1 and 2.

To assemble this project, connect each switch to pins 1-4 and ground. Connect switch 1 to pin 1, switch 2 to pin 2, etc. Remember, do not make a direct connection between pin 7 (5 volts) and pin 8 (ground). If you become confused about which pin is which when you solder your DB-9, refer to the diagram in Chapter 6, The Joystick Ports.

The Software

The switch hardware is very simple, but the software makes it extremely flexible. There's three programs on the disk that work with the switches, SWITCH, STOPWTCH, and LOGIC.

The first program, SWITCH, lets you use your switch as a momentary switch or a toggle switch. When you first run the program, it begins in Momentary Mode. The values of the bits for pins 1-4 of joystick port 1 will be on the screen. If a switch is closed, making a connection to ground, the corresponding bit value on the screen will be 0. Otherwise, it will be a 1.

Chapter 8
Event Detectors

Event detectors signal the computer that something has happened. In a way, they give the computer a very rudimentary sense of 'sight' and 'touch'. We'll discuss different kinds of switch detectors, white light and infrared sensors, and touch switches. The last thing we'll cover in this chapter is the program called DETECT, which monitors these devices. The applications for these detectors is unlimited, so let your imagination go.

Switch Detectors

Switch detectors are the simplest kind of event detectors. The switches are placed between a pin and ground. When the status of the switch changes, the bit values for the pins will change, and the computer will realize that something has happened.

The first kind of switch detector you might want to build is a door alarm. You can monitor up to 10 switches at once; 8 from pins 1-4 in ports 1 and 2, and one from each trigger pin in each port. With a data decoder, discussed in a later chapter, you could monitor up to 255 different alarm switches.

To build your alarm, you'll need door alarm switches, hookup wire, and one or two DB-9 sockets. Hook up the alarm switches to the doors and windows you want to monitor. Connect the switches between the pins and ground, the same way you built the switches in the previous chapter. When the switches are closed, the alarm is set. When a switch is opened, and the connection to ground is broken, the alarm is triggered.

You could also use micro switches as alarms. Very little movement is required to open and close these switches, so they can be used in places where a normal door alarm switch is impractical.

You can use the program DETECT, discussed at the end of this chapter, or a variation to control your alarm system. You can combine your alarm system with some of the other projects in this book to create a complete custom security system. For example, the computer could turn on the lights if it senses motion in a specific area, or sound an alarm if a door or window is opened. You might want to program in delays, so you can enter and leave without triggering the alarm.

Visible Light Detectors

There's several kinds of visible light detectors, but they all operate on the same general principles. A narrow beam of light is aimed at a light sensor. The sensor is connected to the computer through the joystick ports. When the light beam is broken, the computer can 'see' it by the change in the bit values from the ports.

BASIC PHOTOTRANSISTOR DETECTOR

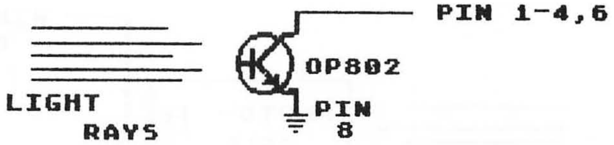
This light detector is made from a single phototransistor. An OP802 works fine, but you can use any low current phototransistor with a low saturation resistance.

Parts

1 DB-9 Female socket
1 OP802 or other phototransistor
Connecting wire
Light Source

An OP802 has three leads. When you look at it from the bottom, you'll see the three protruding leads, and a metal tab. The lead closest to the tab is the emitter, and it is connected to ground. The one farthest from the tab is the collector, and it's connected to the pin in the joystick port. The middle lead isn't used, and can be snipped off.

Solder the wires to the OP802's leads, then connect them to the appropriate pins in the socket. Set the OP802 up on one side of the area you'll be monitoring. Set your light source up across from it, and aim the beam directly at the phototransistor.



When the phototransistor is saturated with light, it has a very low resistance, and the computer will think the joystick pin is connected to ground. When the light beam is broken, the phototransistor will have a very high resistance, and the computer will think the connection to ground is broken.

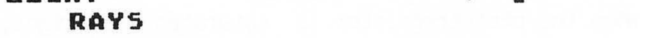
If there is too much ambient light, or light coming from other sources and not ment for the detector, the phototransistor may not be able to tell when the light beam is broken. The next two detectors solve that problem.

BASIC PHOTOCELL DETECTOR

The Basic Photocell Detector works a little differently than the phototransistor detector. The photocell produces a DC voltage when light touches it. This voltage is fed into pins 5 or 9. The A/D converters inside the computer will produce a value from 0-228, corresponding to the level of light touching the photocell.

If you constantly average these values, you can produce a reference level voltage. This lets the computer 'adjust' the detector to work in different lighting conditions. Of course, the hardware itself is not adjusted in any way. The software compares the information from the photocell to the reference voltage. The reference voltage will change as the ambient light in the environment changes.

To build the Photocell Detector, you'll need a photocell, wire, DB-9 Female connector, and, of course, a light source.



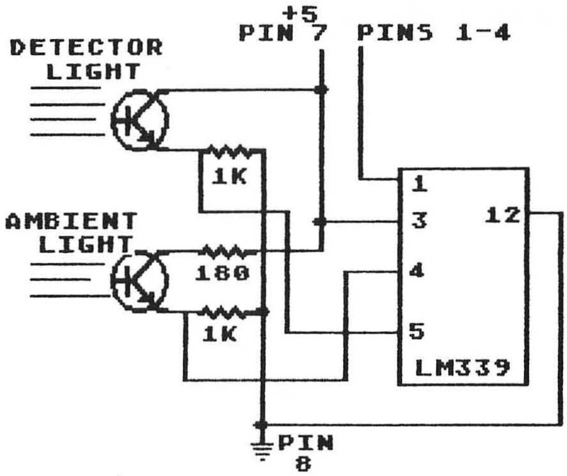
ENHANCED DETECTOR

The Photocell Detector is more flexible than the Basic Phototransistor Detector, but ambient light may still prevent it from working properly. The Enhanced Detector solves the problem by using two separate phototransistors. One sensor sets up a reference voltage based on the ambient light. The other sensor is the actual detector. When the beam of light touching this phototransistor is broken, the detector software is triggered. This detector is very sensitive, it takes only a slight variation in the light level on the second detector to alert the computer.

Parts

1 LM339 comparator
2 OP802s or other phototransistors
1 DB-9 connector
2 1K resistors
wire

Build the detector as shown in the diagram. Set your light source up across the room from the detector, and aim the beam at the detector sensor.



Infrared Detector

Both the Basic Phototransistor Detector and the Enhanced Detector work well with infrared LEDs. Choose an infrared LED with a high output. Radio Shack carries one that emits about 1.5 milliwatts (catalog # 276-143(TIL906-1)).

You can increase the range of your detector with a 6 watt infrared LED. The angle of radiation, the angle at which the beam is emitted, is important. Too high an angle can prevent the detector from working properly. 20 to 30 degrees is best. Avoid those with an angle over 50.

Connect your infrared LEDs with a 1/2 watt 330 ohm voltage dropping resistor. Use the circuit described in Chapter 1 to test your LEDs.

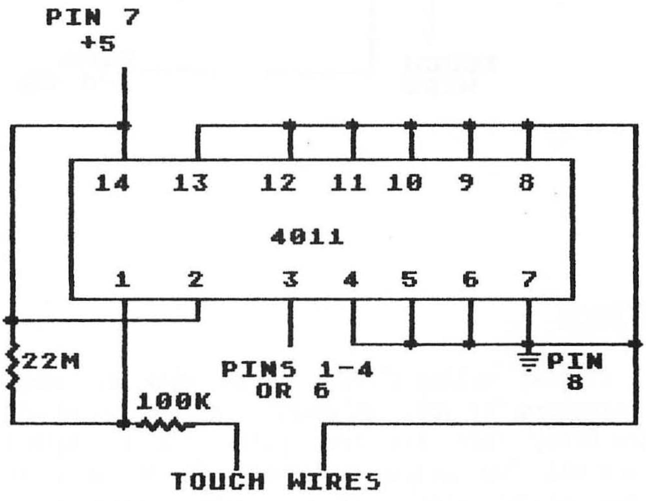
Touch Switches

Touch switches are event detectors that are triggered by the touch of a finger. Touch Switch #1 is activated by touching two wires at once with the same finger. You may want to use an etched metal plate instead of two touch wires. Touch Switch #2 works when a single wire is touched. It's recommended for indoor use only.

TOUCH SWITCH #1

Parts

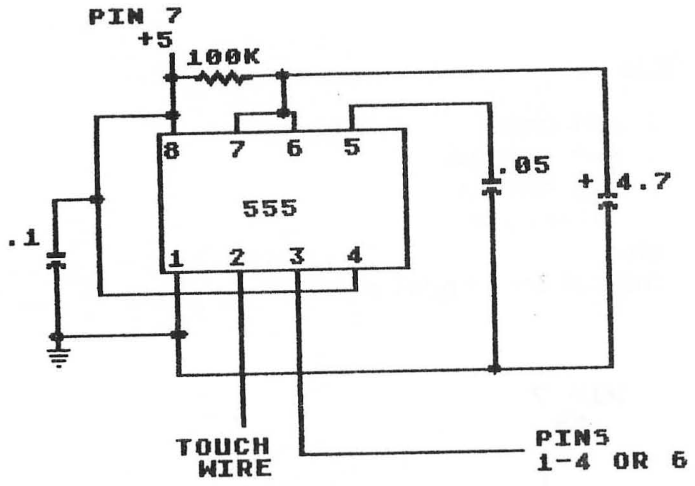
1 4011 CMOS
1 BD-9 connector
1 100K resistor
1 22M resistor
wire
Optional etched metal plate



TOUCH SWITCH #2

Parts

1 555
1 100K resistor
1 DB-9 connector
3 capacitors - .1, .05, & 4.7
wire



The Software

The program called DETECT on the disk will monitor all these detectors for you. It displays the status of all eight joystick bits, (pins 1-4 from ports 1 & 2), both trigger bits, and all four paddle registers. The Photocell Detector uses the paddle pins, all the other detectors use the joystick data bits (pins 1-4) and the trigger bits (pin 6). This program is a good starting point for your custom security system software, or anything else you want to use these devices for.

Chapter 9
Motion Sensors

The devices in this chapter are advanced variations of the light sensors described in the previous chapter. They will enable you to determine the speed and acceleration of a moving object.

First, we'll quickly review some basic physics and mathematic principles about motion, speed, and acceleration. Then we'll describe how to build a simple motion and acceleration sensor, and discuss the programs that work with it. We'll also describe a setup that will monitor the motion of a pendulum, and discuss a pendulum motion program, which will provide you with some interesting data about the pendulum's motion.

About Speed and Acceleration

Speed is a number that describes how fast an object is moving. Technically, speed has no direction. Velocity is speed with direction. To keep things simple, we use only average speed.

To find the average speed of an object, take the distance the object traveled, and divide it by the length of time it took the object to travel that distance. This is the formula Speed=Distance/Time.

For example, let's say the total distance is 5 miles, and the time is 5 minutes. To compute the average speed, you would use the calculation Speed=5 miles/5 Minutes. This divides out to 1 mile/1 minute (or 60 mph).

You are probably familiar with speed measured in miles per hour. Most physicists and other scientists measure speed in meters per second. Since space is a vital consideration in these projects, we measured speed in centimeters per second (a centimeter is 1/100th of a meter). The basic formula remains the same, no matter what units you use to measure time and distance.

Acceleration is a change in velocity or speed. Average acceleration is the difference between one speed and another speed, divided by the time it took to change speeds. The formula is Acceleration=(Speed2-Speed1)/Time.

For example, let's take a car traveling at 40 meters per second (Speed1=40 mps). The car speeds up to 60 mps (Speed2=60 mps). It took the car 40 seconds to go from speed1 to speed2 (time=40 seconds). To calculate this acceleration, the formula looks like this: (60 mps - 40 mps)/40 seconds, or 20 mps/40 seconds. When it's divided out, we get 1 meter per second/2 seconds.

Because the unit of time, in this case seconds, appears twice in the formula, physicists square the units to get them out of the way. In this case, the seconds are squared. So the car's acceleration is 1/2 meter per second ^{2}.

THE MOTION DETECTOR

When you set up your motion detector, you'll know the distance between each individual sensor. You need only two sensors to determine speed. When an object passes the first sensor, the computer begins keeping time. When the second detector is triggered, the computer stops. Now you have the time, and you already knew the distance, so the average speed is easy to find.

We used four detectors and a slightly simplified version of the acceleration formula to compute average acceleration. We found two accrelerations, added them, then divided by 2.

When an object passes detector #1, the timer begins. The speed between detector #1 and #2 is speed1. The speed between #2 and #3 is speed2. If you subtract speed1 from speed2, you get acceleration1 (Speed2-Speed1=Acceleration1). The speed between #3 and #4 is speed3. If you subtract speed2 from speed3, you get Acceleration2 (Speed3-Speed2=Acceleration2). To find the average acceleration, simply add both accelerations, and divide by 2 (Acceleration1 + Acceleration2 / 2 = Average Acceleration). So, our formula looks like this: ((Speed2-Speed1) + (Speed3-Speed2)) / 2 = Average Acceleration.

For example, let's say Speed1=1, speed2=2, and speed3=3. Our computations would look like this: Acceleration1=(2-1), or 1. Acceleration2=(3-2), or 1. So (Acceleration1+Acceleration2)/2 is really (1+1)/2, or 2/2, so the Average Acceleration is 1. If we used the speeds alone to calculate the average acceleration, it would look like ((2-1)+(3-2))/2, or (1+1)/2, or 2/2, or 1.

Because the acceleration formula we used is somewhat simplified, the acceleration program we've provided will produce correct results only if acceleration is constant. If the acceleration changes while it is being measured, the calculations will be off.

Parts

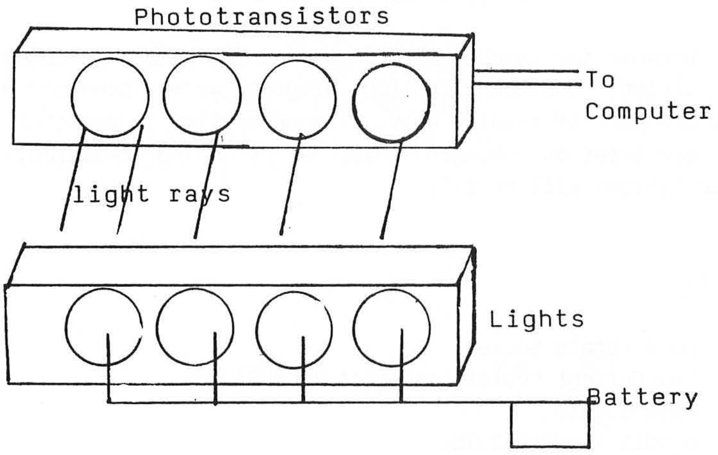
1	DB-9 Female Socket
4	Low Current Phototransistors (OP802)
4	Flashlights
4	6 volt krypton bulbs
1	6 volt lantern battery

3ft baseboard moulding
Hookup Wire

The theory behind the device is relatively simple, and construction is not difficult. There are four phototransistors in a row, and across from each is a light source. When a light source strikes a phototransistor, the computer will think the corresponding pin is connected to ground, so the bit for that pin will have a value of 0. When the light beam is broken, the bit value changes, and the computer begins keeping time.

For our light source, we took four flashlights, and replaced the bulbs with krypton bulbs. Then we wired them to a single lantern battery. This gave us brighter light beams, and the lantern battery lasted longer than the normal 'D' cell flashlight batteries. You could also use four AC powered light sources, if that is more convenient for you.

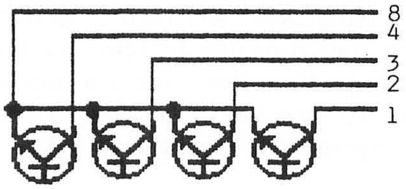
Begin building the project by mounting all four phototransistors in a row, all equally spaced. We used plastic moulding from a lumber store, drilled four holes just big enough for the phototransistors to fit snugly, and then superglued them in place. Below is a diagram:



We used the same system to set up the light sources. After we replaced the bulbs and wired them to the lantern battery, we mounted them in a piece of baseboard moulding. We drilled holes and glued the lights in place, as shown in the diagram below.

NOTE: When you mount the phototransistors and the lights make sure that the distance between the center of each phototransistor is exactly the same. Make sure that the distance between the center of each light is exactly the same as the distance between the center of each phototransistor.

Now you must wire the phototransistors to the computer. Each one will trigger a separate pin. Set it up so that bit four (pin four) will be the first to be tripped, followed by bit three, bit two, and finally, bit one. In other words, detector 1 will correspond to pin 4, detector 2 to pin 3, detector 3 to pin 2, and detector 4 to pin 1. Below is a diagram of the connections.



Now you're ready to set everything up. Set up the lights on one side of the object's path, and the sensors on the other side. Make sure the lights and sensors are lined up perfectly.

Load in the program you want to use. We've provided two. All you have to do now is turn on the lights, and begin measuring the motion of your objects.

The Software

The program SPEED will calculate the average speed of an object for you. You need to enter the distance between the phototransistors, in centimeters. The computer will do the rest for you.

The program SPDACCEL will calculate the average speed and average acceleration of an object for you. You need to enter the distance between the phototransistors, in centimeters. The computer will do the rest. This program uses the simplified method of determining average acceleration, as we described earlier. It is accurate only if the object has a constant acceleration.

Pendulum Motion

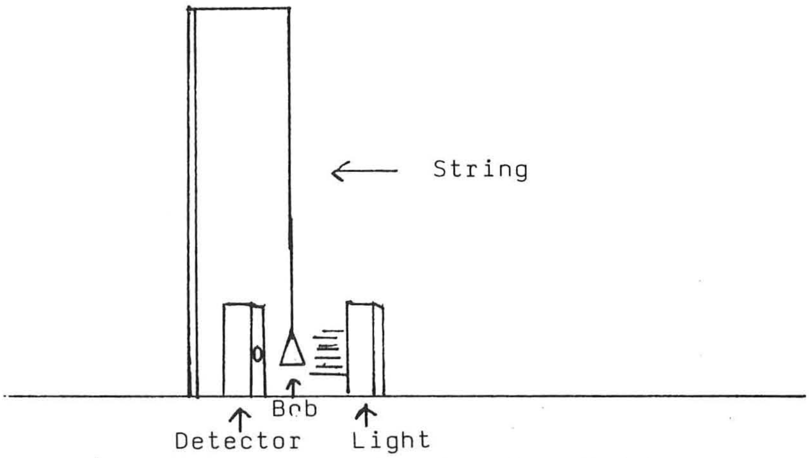
You can use the motion detectors to perform physics experiments with a pendulum. A simple pendulum is an object suspended on a string. The mass of the object, called a bob, must be substantially greater than the mass of the string.

When you pull the bob to one side and release it, the pendulum will swing back and forth, gradually coming to a stop. The path the pendulum follows as it swings from one side all the way to the other side and back again is called a cycle. A period is the amount of time it takes the pendulum to complete a cycle. The frequency is the number of cycles per second.

The period and frequency depend upon the length of the string, the mass of the bob, and the force of gravity. The theory and mathematics are too complicated to cover here. If you are interested in learning more about pendulum motion, any good physics book can provide you with a detailed explanation.

The pendulum project we built used a single phototransistor, a light source, a DB-9, wire, and, of course, a pendulum.

We wired the phototransistor to pin 1 and pin 8 (ground) in port 1. We set up the detector and the pendulum as shown in the diagram below.



The program called PENDULUM will calculate some statistics about the pendulum's motion. It will ask you to enter the length of the string. Then it uses information from the phototransistor to give you the frequency, period, and bob mass.

Chapter 10
Light Pen

A light pen is a fairly simple device. Once you understand how it works, you can use it for almost anything. We'll start off this chapter with a brief review of how your monitor works. Then we'll discuss how the light pen works, and show you how to build your own. Finally, we'll talk about the light pen software we've provided on the disk.

About your monitor and light pen

Your TV or monitor screen is really a big vacuum tube, shaped somewhat like a pyramid with rounded corners. The large, flat end is the part you see as the screen. Inside the tube, located towards the small end, is a cathode 'electron gun'. This is why monitors used to be called Cathode Ray Tubes (CRT). The popular term now is Video Display Terminal (VDT), but it's still the same thing.

The electron gun shoots a beam of electrons at the screen. These electrons produce the image that you see. The picture is drawn in tiny dot-like units called pixels. The brightness and color of each pixel is determined by the Display Memory (see Chapter 3 for a review of Display Memory).

The electron gun draws the picture one line at a time, from left to right. Each line is called a scan line. The gun begins in the upper left hand corner, and moves down the screen line by line. When the electron stream passes accrossa pixel, the pixel is 'fired' and begins to glow.

As you remember from the discussion of DLI and VBI routines, the gun pauses at the end of each line to stay in sync, then returns to the left side of the screen. Your DLI routines run during this pause. When the gun reaches the lower right hand corner, it pauses, then returns to the upper left corner to begin drawing the screen again. This pause is called the vertical blank, and this is when your VBI routines are performed. It takes only 1/60th of a second for the gun to draw an entire screen, from start to finish.

When you place the light pen against the screen, the pen waits for a pixel underneath it to be fired. When the pixel is fired, a phototransistor inside the pen 'sees' it. The pen signals the computer that it has 'seen' the pixel fire. The computer's operating system automatically figures out the pen's location on the screen.

LIGHT PEN

The light pen uses a voltage comparator, a voltage divider, and a phototransistor. The phototransistor is connected to the voltage divider. The phototransistor's resistance varies, depending upon the amount of light shining on the phototransistor. When the phototransistor is exposed to more light than normal (such as when a pixel beneath it fires), the resistance decreases. As the resistance falls, the voltage out of the divider rises.

The output from the divider is fed to an LM339 voltage comparator. If the voltage from the divider is below the reference level, nothing happens. But, when the voltage rises above the reference level, the comparator's output goes low.

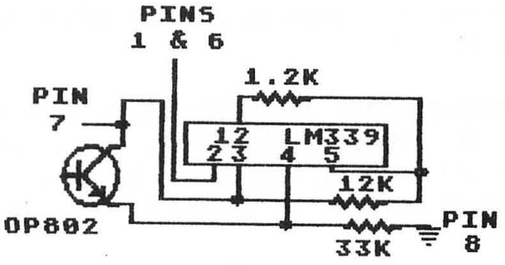
When the comparator's output goes low, the value of the bit corresponding to pin 6 (the trigger pin) changes. This tells the computer that the pen has 'seen' a pixel fire.

The software takes over from here. The computer's operating system has special routines that determine the horizontal and vertical coordinates of the light pen. The program for your light pen can read these coordinates, and perform whatever activity is associated with that location on the screen. In our sample program, screen locations are used to turn sound channels on or off, or change the pitch of the sound from a particular channel.

In a nutshell, the whole process works like this: 1) A pixel below the light pen fires. 2) The phototransistor is flooded with light, and it's resistance falls. 3) When the resistance falls, the voltage out of the divider rises. 4) When the voltage from the divider rises above the reference voltage, the voltage comparator's output goes low. 5) The low output tells the computer that the light pen has 'seen' the pixel fire. 6) The computer's OS figures out where the light pen is positioned on the screen. 7) Your program gets the location, and does whatever. In our sample program, it could turn a sound on or off, or change the pitch.

<u>Parts</u>

1	DB-9 Female Socket (Hood Optional)
1	Phototransistor (OP802)
1	LM339 Quad Comparator
1	1.2K Resistor
1	12K Resistor
1	33K Resistor
1	Old Pen case or similar cylinder to contain your pen
	Wire



Once you've constructed the circuit, you'll probably want to put it in some sort of a case. We used the case from an old pen, but any cylinder or rectangular case will work.

You can adjust the sensitivity of your light pen by altering the reference voltage. Increasing the 1.2K resistor will increase the reference voltage, and decrease the sensitivity. It will take a brighter pixel to trigger the pen. Decreasing the 1.2K resistor will decrease the reference voltage, and increase the sensitivity. A dimer pixel will trigger the pen.

The Software

As we mentioned earlier, the computer's operating system finds the light pen's position for you. It uses locations 564 (horizontal value) and 565 (vertical value) to determine the position of the pen on the screen. The software we've provided isn't sensitive enough to let you draw accurately with the light pen, but with some experimentation (and these locations) you can easily write more sophisticated software for your pen.

The program called LGHTMUSC on your disk works with the light pen. It displays four vertical bars, corresponding to the four sound channels. Above each bar is a switch. You can turn the sound channels on and off by placing the light pen turn the sound channels on and off by placing the light pen over the switches. Move the pen up and down the bar to change the pitch of the sound. You can have all four channels on at once.

When you're ready to leave the program, hold the pen up to any bar or switch, and press BREAK.

When you write your own light pen programs, you might want to keep the backround dark, and important places on the screen bright. If you use switches or boxes, leave some space in between each one, and don't make them too small. This will make your software easier to use.

Chapter 11
Using Analog Input

Most of the devices in this book so far have used digital data. Digital data is data that has distinct values. Analog data can be any number over a continuous range of values. For example, let's say you have data that can have a value from 1-5. If your data was in digital form, it could have one of five possible values, 1, 2, 3, 4, or 5. A string of data might look like this: 1, 4, 2, 5, 3, 3, 2, 5, 1, 4. If the data was in analog form, it could have any value between 1 & 5, including fractions. An analog string might look like this: 1, 3.7, 4, 2.1, 4.9, 3, 2.9, 2.1, 4.7, 1, 4.

How does this apply to the data from the joystick ports? Let's look at the data bits for joystick pins 1-4. These pins and their data bits can be either high (approaching 5 volts), which is 1, or low (approaching 0 volts), which is 0. This kind of data (it can be only 0 or 1) is digital. By combining the bits from all four pins (creating a nibble, or half a byte), the data could have any whole number value between 0 (0000 in binary) and 15 (1111 in binary).

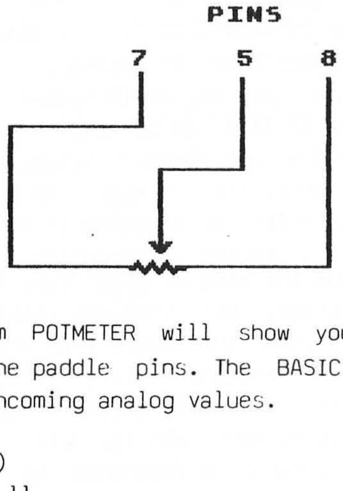
Analog data doesn't have distinct values. The paddle pins 5 & 9 are analog pins. The voltage can be approaching 0 volts, approaching 5 volts, or anything in between, such as 2 1/2 volts. Pins 1-4 wouldn't recognize or understand a voltage of 2 1/2, but the paddles pins can. Not only do they understand this voltage, but they can assign it a value between 0-256. The voltage that comes into the paddle pins is analog data. It can have any value, including fractions, between 0 & 5 volts. When this data is assigned a value between 0 & 256, it becomes digital data. It can have any whole number value between 0 (00000000 in binary) and 256 (1111111 in binary). Now the data is in a form the computer can use. The pins use their own A/D converters to transform the analog data into digital data.

Because of a bug in the Atari computer's operating system, the paddle pins can't use values from 229-256 (11100101-1111111 in binary). You can overcome this by using the external fix described in this chapter.

A paddle controller is just a potentiometer. When you turn the paddle knob, the resistance inside the paddle is changed. When the resistance changes, the voltage changes. The A/D converter inside the pins transforms the analog voltage level into digital data the computer can understand.

POTENTIOMETER

This is the schematic for a potentiometer, or a paddle controller. You may want to experiment with it to become familiar with analog data and A/D conversion before you try the harder A/D projects.



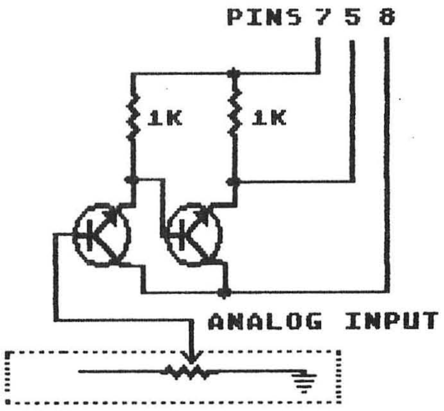
The program POTMETER will show you the digital data coming from the paddle pins. The BASIC program below will show you the incoming analog values.

```
10 X=PADDLE (0)
20 POSITION 20,11
30 PRINT X;" "
40 GOTO 10
```

PADDLE PIN FIX

This fix will let you use values from 229 (11100101), or about 1.5 volts (the actual voltage will depend on your computer) and 256 (11111111), or 0 volts.

Use general PNP transistors, such as 2N3906. You can use anything for the analog input, we used the potentiometer as an example.



Chapter 12
Sound & Wave Generators

The world of sound is both complex and fascinating. We'll begin this chapter with a brief review of sound, then show you how to build some simple sound synthesizers for your Atari.

A Little About Sound

The sounds you hear in the world around you are really vibrations in the air. Your ear senses these vibrations and sends them to your brain. Your brain recognizes the pattern of the vibrations, so you can understand a friends words (if he's speaking a language you know!) or enjoy a piece of music.

A picture of these vibrations is a sound wave. Each sound has it's own pattern, or wave shape. Sounds are made up of many different tones, undertones, overtones, reverberations, envelopes, and frequencies. The more complex the sound, the richer the sound quality. A piano or violin note, for example, is very complex.

Very simple sounds have very simple sound waves and easily identifiable sound wave shapes, such as sine waves or triangle waves. These sounds aren't as pleasant to hear, but they are easy to generate, manipulate, and study. They can also be combined to create richer, fuller sounds.

Technically, frequency is a measurement of a sound, based on the wave shape. Sounds with higher frequencies usually have a higher pitch, sounds with lower frequencies usually have a lower pitch. For the projects in this book, you can think of frequency as pitch.

Sound, sound waves, and sound manipulation are far too complex to cover in detail here. Any good physics book will explain the basic physical properties of sound and sounds waves. "Introduction to Electro-acoustic Music", by Barry Schrader, contains a very thorough discussion of sound, the different components of sounds waves, and how they are used in electronic music. There's also several magazines, such as Electronic Musician, devoted to the topics of electronic, digital, and computer controlled sound and music.

Sound and Your Atari

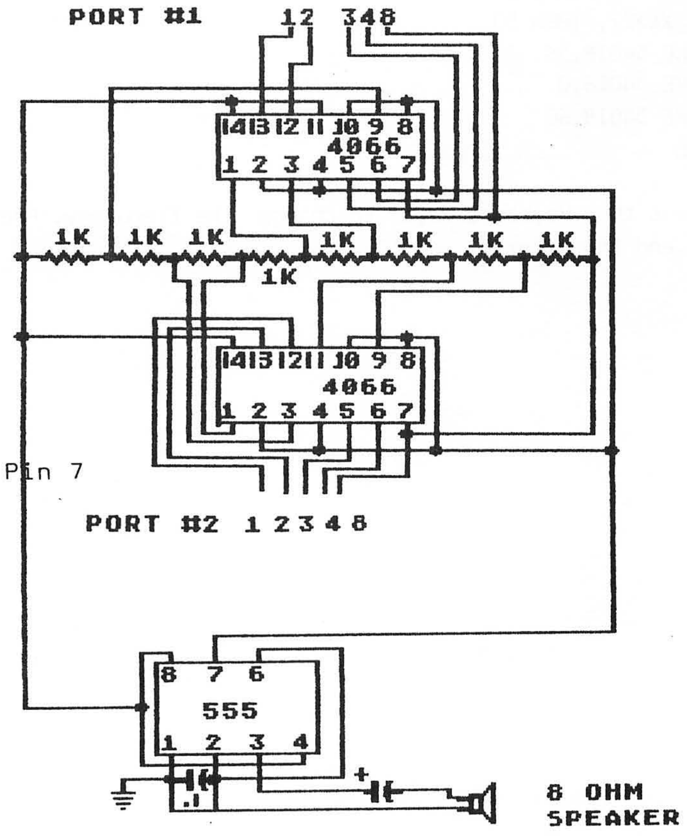
Atari computers have a powerful sound chip. The computer can't hear sounds or see sound waves, so it treats sounds as strings of data. The data is digital, which means that each piece of data can be any whole number in a certain range of numbers (usually 0-15 or 0-256). Every sound your Atari stores, creates, or produces is a digital sound, because it is generated with digital data.

In this chapter we'll build a Tone Generator and a Waveform Generator. The Tone Generator is really a simple oscillator. It creates a very simple sound wave, and the computer generates the sound of that wave.

TONE GENERATOR

Parts

1	DB-9 Female socket
1	555 Timer
2	4066 Quad Bilateral Switches
10	10K Resistors
10	4.7K Resistors
1	.1 Microfarad Disk Capacitor
1	1K Resistor



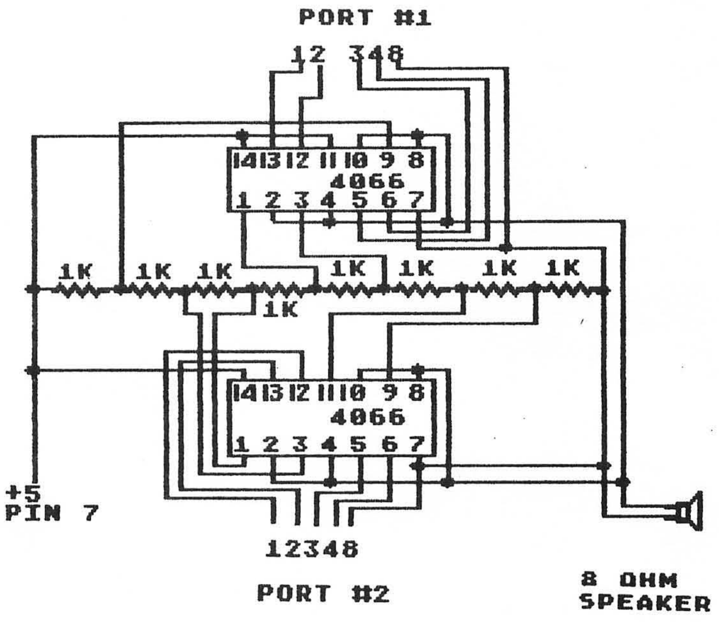
The 555 timer produces the frequency (or sound wave). The 4066 quad bilateral switches change the frequency. There wasn't room for a demonstration program on the disk, but you can use the BASIC program below.

```
 10 OPEN #2,4,0,"K"
 20 POKE 54018,56
 30 POKE 54016,255
 40 POKE 54018,50
 50 GET #2,X
 60 POKE 54016,255-X
 70 IF X<>27, THEN 50
 80 POKE 54018,56
 90 POKE 54016,0
100 POKE 54018,60
110 END
```

Press the keyboard keys to change the frequency. Press ESC to end the program.

WAVEFORM GENERATOR

This circuit creates a more complex sound wave for you.



The waveform generator produces a voltage level. The values that correspond to the voltage levels are poked to the joystick register. The waveforms are produced by varying the voltage.

The program called WAVEFORM will create a waveform for you. It takes a number from 0-256. The waveform generator will generate the frequency that corresponds to that number. The larger numbers produce larger voltages, smaller numbers produce smaller voltages. Listen to the differences in the sounds created by large and small numbers. Type in the numbers you want to hear. Alternate between different voltage levels, and listen to the the changes in the sound.

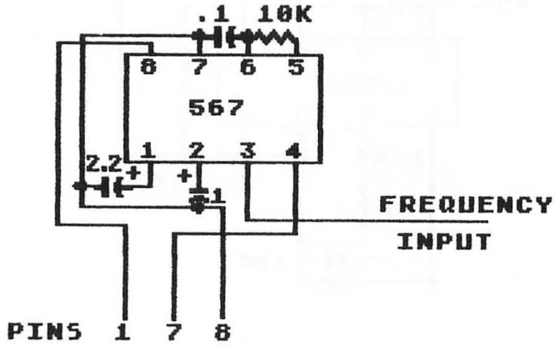
Chapter 13
Tone Decoder

Two of the devices in this chapter are tone decoders. The first decoder is able to recognize preset frequencies. The other is a frequency to voltage generator. It transforms sound waves into voltage levels, which the paddle pin can transform into digital data. You can use these devices by pluging in a microphone and whistling. If your microphone doesn't give enough gain, use the simple amplifier described in this chapter. We've also included a frequency generator to use with either of these tone decoders.

TONE DECODER

Parts

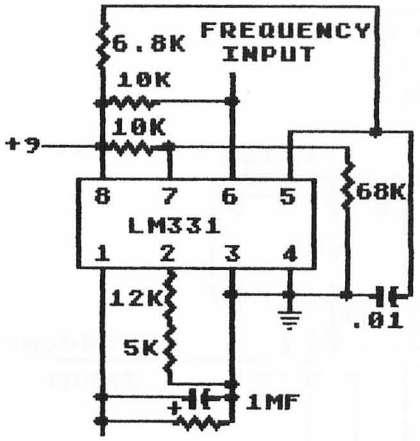
1	567 Tone Decoder IC
1	10K Resistor
1	.1 Capacitor
1	1 Capacitor
1	2.2 Capacitor



The IC output goes to pin 1 in the joystick port. When the IC "hears" a frequecy it recognizes, its output goes low. This tells the computer that the IC "heard" a tone, and it should take appropriate action.

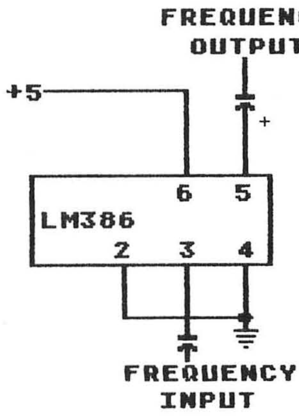
FREQUENCY TO VOLTAGE CONVERTER

This tone decoder is a little more sohpisticated. It uses a frequency to voltage converter. The frequency it "hears" is converted to a corresponding voltage level. The voltage level is sent to the paddle pin. The paddle pin converts the voltage level into a corresponding binary number. Then the computer can perform whatever task is associated with that number.



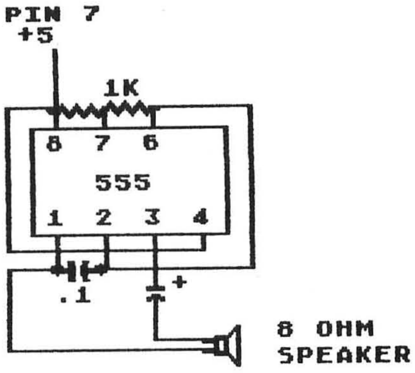
AMPLIFIER

If your microphone doesn't have enough gain to work well with these decoders, you can use this amplifier instead.



FREQUENCY GENERATOR

The tone decoders in this book are fairly picky about what frequencies they will recognize or reject. This frequency generator produces exact frequencies, making it easier for the tone decoders to spot a match (very helpful for poor whistlers). You don't need to keep the generator right next to your computer. You can put it wherever you like, such as in another room, as long as the tone decoder can hear the tones.



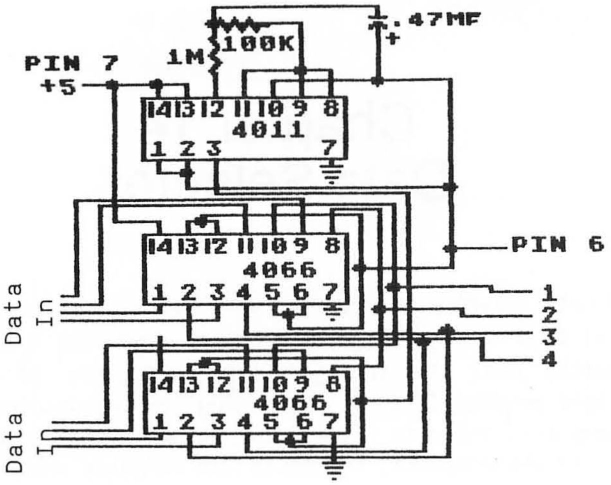
Chapter 14
Data Selector

A data selector is a device which lets you plug two other devices into a single joystick port. The data selector has two ports, just like your joystick ports. You can plug two different devices into it, then plug the data selector into the computer. The data selector sends data from one device at a time to the computer, and tells the computer which data set it's sending.

Our system uses an external clock to tell which data set is in use. You could use software to switch the data sets.

Parts

1	DB-9 female socket
1	4011 Quad NAND Gate
2	4066 Quad Bilateral Switches
3	14 Pin Dip sockets
1	.047 microfarad electrolytic capacitor
1	100K resistor
1	1M resistor
2	DB-9 male plugs



The data selector is plugged into port 1. The port uses an external clock, connected to pin 6 (the trigger pin) to switch from one set to another. When the clock is high, the trigger pin is high. When the trigger pin is high, the computer is receiving the first data set. When the clock is low, the trigger pin is low, and the compuer is receiving the second data set.

The Software

The program called OVERLAY2 reads the data from the selector. It shows both sets of data, and the toggle state. If the toggle state is Yes, then the selector is toggling between both sets of data. If it says No, then the data selector isn't switching. Instead, it's reading only one set of data.

To run the program, rename it AUTORUN.SYS, and reboot the computer. It creates an overlay at the top of the screen. You can write your own BASIC program to use the data selector, and still see the data on the overlay. The overlay data is updated with a VBI.

Turn the overlay off by pressing SELECT. To turn it back on, press START.

You could write your own software that would automatically switch between data sets, or let you switch them from the keyboard. To do this, use one of the bits from the joystick port as a control bit. If you don't want the external clock, you could use that pin.

Chapter 15
Encoders

One problem with sending data to the computer through the joystick ports is the limited number of pins. Even if you use both port 1 and 2, you can quickly run out of pins.

One solution is to use each pin for more than one detector, but then you can't identify which device was triggered. The way to solve this problem is to build an encoder. With an encoder, you can plug in 16 detectors, and still see which device was tripped. In this chapter, we'll show you a simple diode encoder, two IC encoders, and a keypad, as well as a program to control these encoders.

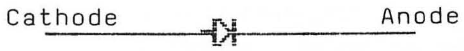
DIODE ENCODER

The first encoder, a diode encoder, is simple, and it has some limitations. It's not priority encoded. This means if more than one detector is tripped, the reading will not be accurate.

Parts

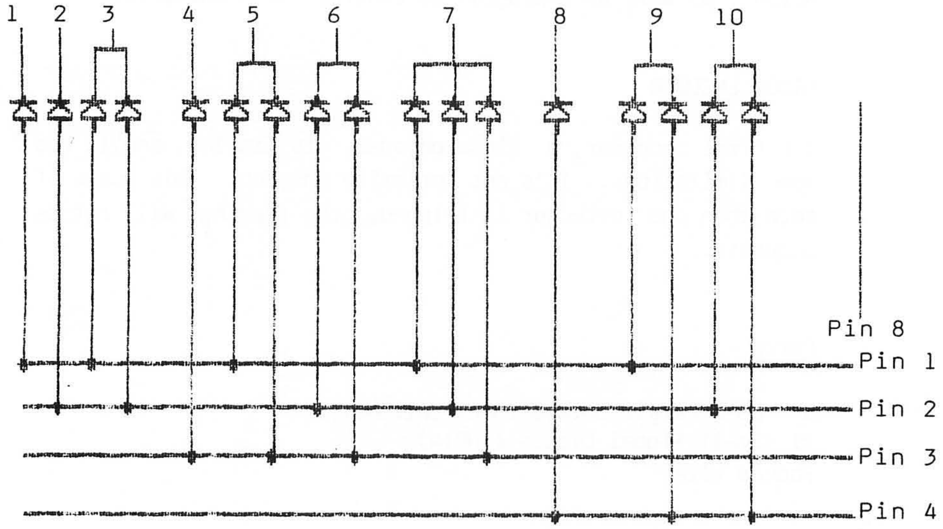
1 DB-9 Female Socket (Hood Optional)
31 Small Signal Diodes (1N914)
Hookup Wire

The diodes have two different ends; one is the cathode and one is the anode. The cathode has a dark band around it. The schematic of a diode looks like this:



Notice that the cathode end is the end with the vertical line, which corresponds to the band on the actual diode. In the schematic, the anode end is the one with the small triangle. Make sure the cathode always goes to ground, and the anode always goes to the data pin.

This encoder will encode up to 10 data lines. Each device uses one data line, so you can connect up to 10 devices with this encoder. Each device triggers a different pattern of pins. The different pin patterns will produce a different four bit number for each device. The setup looks like this:



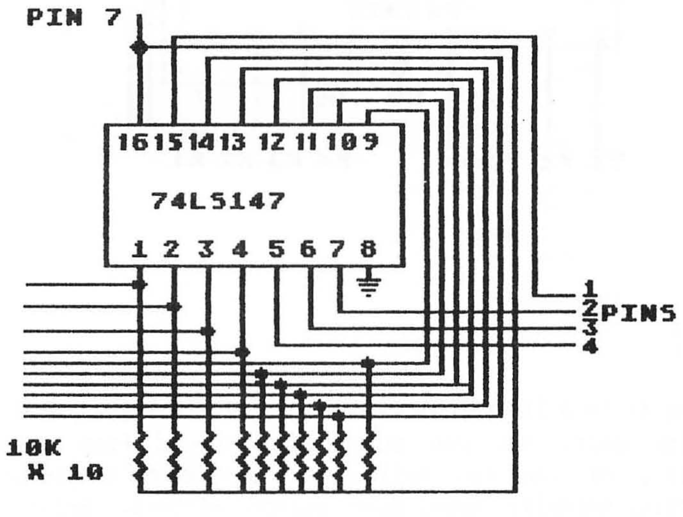
Although this encoder will work, it does have a serious drawback. It must be reset each time a device is triggered. If two devices are triggered at once, the binary number that represents the devices will be wrong.

PRIORITY ENCODER

This encoder, unlike the diode encoder, will always give you accurate data. No matter how many devices are triggered, it will always tell you which one was triggered first.

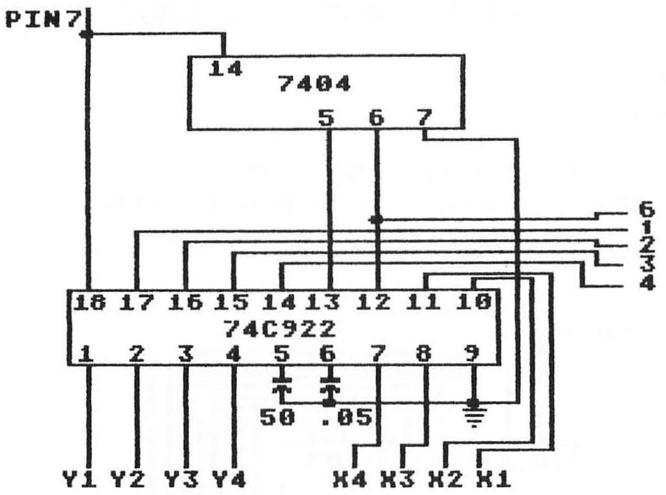
Parts

1 DB-9 Female Socket (Hood Optional)
1 10 to 4 priority encoder - we used a 74LS147
10 10K Resistors
Hookup wire



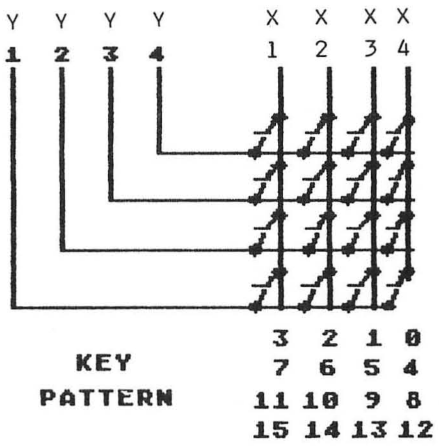
16 TO 4 LINE ENCODER

A 10 to 4 line encoder may not provide enough lines for your needs. If so, you can use this 16 to 4 line encoder. It uses a 74C922 16 to 4 line encoder. This IC has more external data lines than the 74LS147 that we used in the Priority Encoder. It also has a pin which can detect keypresses when it's used with a keypad.



KEYPAD

The 16 to 4 line encoder lets you plug a keypad into the joystick port. You can purchase a keypad from a parts supplier, or you can build your own with single pole, momentary, normally open, push button switches. Below is a diagram of the setup.



The Software

The program for these encoders is called (what else?) ENCODE. It displays the data from port 1. The default mode (the mode it's in when you first load it) works with the Diode and the Priority Encoders. Press START to use it with the 16 to 4 Line Encoder. Press OPTION to return to the original mode.

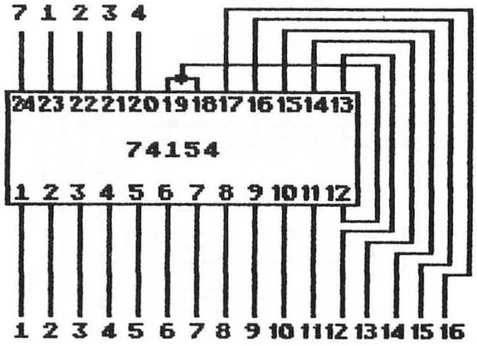
Chapter 16
Decoders

As we've seen, one problem with using the joystick ports to communicate with other devices is that we can quickly run out of pins and ports. Decoders allow you to control up to 16 devices from one joystick port. First we'll describe a 4 to 16 Line Decoder, then a T Flip-flop.

Build these circuits on solderless breadboard, and use LEDs to test them. The program DECODE on the disk will work with this decoder, with or without the flip-flop, and joystick port 1. You could change it to use both ports.

4 TO 16 LINE DECODER

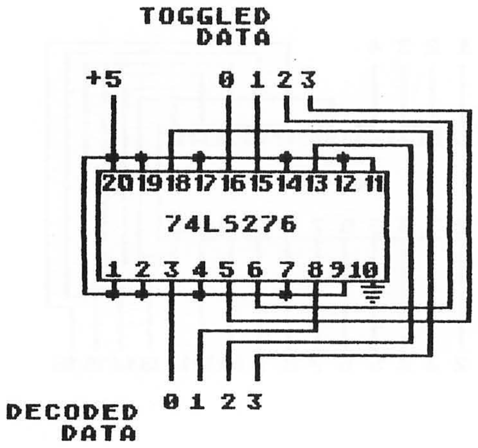
This decoder uses a 74LS154 4 to 16 line decoder. One drawback with this decoder is that it lets you control only one device at a time.



Pins 1-11 and 13-17 are used for the output. Connect them to your devices. Pin 12 is ground. Pins 20-23 are connected to pins 1-4 in your joystick port, and they carry data from the port to the 74LS154. Pin 24 is +5 volts. Finally, pins 18 & 19 must be held low for the decoder to work properly.

T FLIP-FLOP

The T Flip-flop output will toggle back and forth between low and high. It begins with a low input and a low output. When the input goes high, then low again, the output toggles to high. If the input goes high, then low again, the output will toggle back to low. The IC we used is a 74LS276 quadruple JK flip-flop, set up in T flip-flop format.



Chapter 17
Device Control

The individual bits inside your computer can be either 0 or 1. You could call these states on and off. If you set up your joystick ports for output, you can turn your devices on and off by changing the bits in the joystick ports. You may want to review Chapter 6, The Joysticks, before you begin this chapter. We'll be covering LEDs, Reed Realys, Transistor drivers, Opto Triacs, Solid State drivers, and applications.

LEDs

We briefly covered LEDs in the Electronics Review. LEDs are particularly useful when you are writing data to the joystick ports, because they let you see the state of the pins.

TTL ICs can't provide enough current to drive LEDs, but can make them glow dimly. To get as much brightness as possible, connect the LED's cathode to the IC's output and the anode to a positive voltage, through a voltage dropping resistor. This way, when the IC's output goes low, the LED's cathode is connected to ground, and the LED will glow.

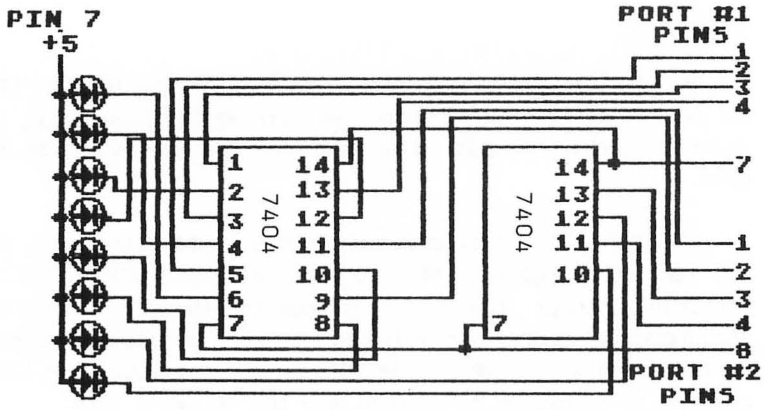
The IC output can sink enough current, even though it cannot source enough. The 10 - 30 milliamps (depending on the LEDs) will go to ground through the IC's output. Just don't connect the anode directly to the voltage without a resistor, or the LED will burn out. Use Ohms Law (discussed in the electronics review) to calculate the correct current. A 1K resistor will almost always work well.

You could use an external power supply, but it's probably easier to use the 5 volts from the joystick port. If you decide to use an external power supply, make sure everything is connected to a common ground.

The LED will light up when the ICs output goes low. If you use an inverter, the IC's output will go low when the joystick bits are high. So, to turn on an LED, set the corresponding data bit high.

LED Control

2 DB-9 Female sockets
2 7404 inverters
8 LEDs
1 1K resistor (or other resistor)
wire



This particular setup uses 8 LEDs from 2 inverters, controlled by joystick ports 1 & 2. Each IC has 6 gates, so two are needed. The power is pin 7, the ground is pin 8.

This is a fairly simple design, but it's possible to build elaborate, intricate LED displays using these techniques. You could use the pins (1-4 in ports 1 & 2) as eight independent channels. This method is used in the Christmas light display described later in this book. You could even synchronize the display with music from the sound chip.

REED RELAYS

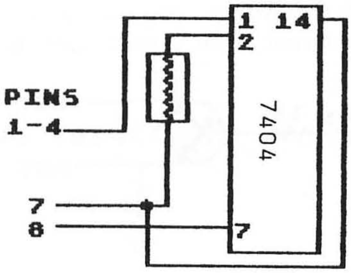
A 5 volt reed relay is extremely flexible. It will handle anywhere from 0 to 120 volts. At 120 volts, it can carry 1 amp. This lets you power a surprising number of different kinds of devices.

As an example, let's find the biggest light bulb a 5 volt relay will power. We'll use the formula P=ExI. As you remember from the Electronics Review, E is the voltage in volts, I is the current in amps. Our formula looks like this: P=120 volts x 1 amp. So the relay can power up to a 120 watt light bulb.

The relay has three leads. The two leads directly across from each other are for the primary coil, and should be connected to the IC's output. Connect the third lead to 5 volts, without a resistor.

When the IC's output goes low, one side of the relay's primary goes to ground. This opens the relay's secondary. Remember, the IC's output must be low to enable the relay. The easiest thing to do is use an inverter. Otherwise, keep that fact in mind when you write your software.

1 DB-9 Female Socket
1 7404 Inverter
1 5 volt Reed Relay
wire



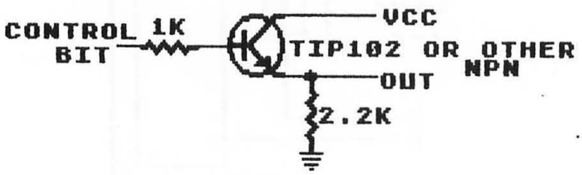
You can power up to 4 or 5 relays with the computer's internal power source. If you need more relays, use an external power supply.

If you need more power, you can use a 5 volt miniature PC relay. Radio shack carries a 5 volt mini relay that draws 72 milliamps for the primary coil, the secondaries can handle 3 amps at 120 volts. You will need an external power supply for these realys.

If you still need more power, you can use a bigger relay. These higher amp relays usually require 12 volts. You'll need to use a transistor driver and an external power source. Or, you could use a reed relay to control the higher amp relay.

TRANSISTOR DRIVERS

A transistor driver will power devices which require higher voltage than a TTL IC can provide. A TIP102 provides a high current, according to the manufacturer's specifications it can provide up to 8 amps. It comes in a TO-220 case, and can be from 5 to 15 volts, depending on your device. If you plan on a high current, use a heat sink to prevent damage.

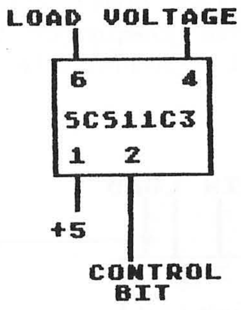


Almost any NPN transistor will work in this circuit. Just make sure you won't need more current than it can handle. A good medium current transistor is the TIP31A. It handles up to 1 amp, and comes in a TO-220 case, so a heat sink can be attached.

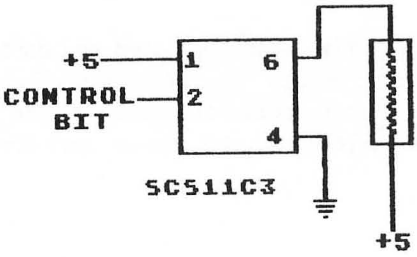
A transistor driver can be used to control a stepper motor. Most stepper motors have four stator coils. The motor moves when the stator coils are pulsed. The combination of pulsing stators determines the speed and direction of the motion.

OPTO TRIACS

An opto triac will drive a low current, 120 volt device. Other opto isolators will also work in this circuit.

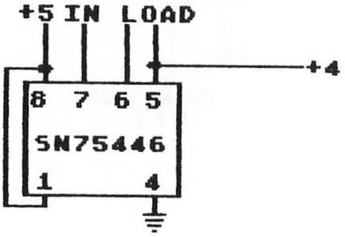


An opto triac can also control a relay, as in this circuit.

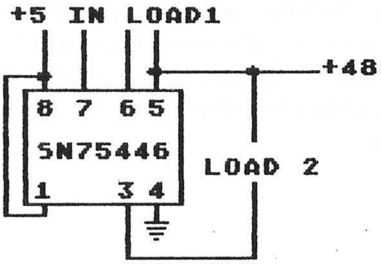


SOLID STATE DRIVERS

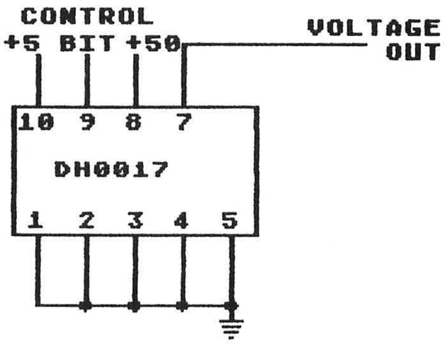
Some ICs can driver higher current, higher voltage devices. Here are three examples of these drivers. In the first, we used a dual peripheral driver, a 75446, to drive a single device.



This circuit uses the same chip to drive 2 separate devices.



This circuit uses a DH0017. At 50 volts, it provides more current than the previous circuits.



APPLICATIONS

You could use these drivers to control your household devices with your computer. Choose which circuits you need, depending on what devices you want to control. The data selector and the decoder will let you add more devices. You can use any program on the disk that writes to the joystick ports to test your hardware. When you've designed and built your system, you'll probably want to write your own custom control software.

Chapter 18
Display Lighting

The display lighting circuit was designed for a christmas light display, but you could use it for a store sign or other lighting display.

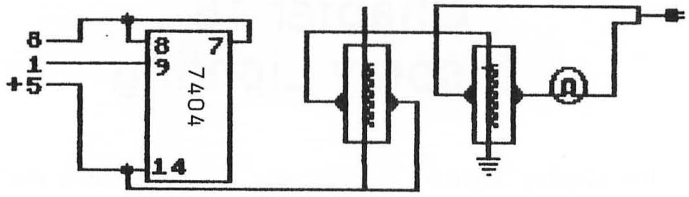
The hardware is simple, but tedious, to build. There's a lot of connections to make, but if you are patient and careful when you make them it will reward you with a delightful display.

We recommend you use a perforated breadboard for the circuit. Use sockets for all the ICs, and the relays, too, if you like.

The entire circuit uses eight channels. There are three main components for each channel. The first component is an inverting buffer. This buffer protects the computer from any possible damage. The second component is a 5 volt reed relay. This relay enables the high current relay, which turns the lights on and off. The large relay we used handles 3 amps, and powers up to 360 watts of lights. Use a higher amp relay for more lights.

The external power supply powers the lights. It should be 12 volts, with a current capability of about 1 amp. We used a 5 volt regulator, so we could have both 5 and 12 volts.

Below is the diagram for one channel of the circuit. You'll need eight of these, one for each channel.



The Software

The software that controls the light display is called LIGHTS. When you run the program, the screen display will show you when the channels are on. This screen display slows the program down. You can run your light pattern faster by turning off the screen. Press SELECT to turn it off, and START to turn it back on.

The patterns are in the data statements starting at line 1000. You can change the patterns by changing the data statements.

Each signal in the pattern consists of three pieces of data. The first byte is the sum of all the bits (lights) to be turned on. The second byte is the length of time the lights will be on, in fourths of a second. The third byte is either 0 or 1. If it's 0, then the pattern is complete. If it's 1, then the pattern isn't finished yet.

The number of times a pattern is repeated is at line 10000. You can change the number of repetitions by changing this line.

Chapter 19
Serial Data Transfer

Computers communicate in two ways, with serial data transfer and parallel data transfer. In serial data transfer, the data bits are sent one at a time. In parallel data transfer, several bits are sent at once. Naturally, parallel transfer is much faster, but serial transfer has some advantages, too.

Serial data transfer requires far fewer lines and connections. The serial transfer methods we use in this chapter need only two lines, one for the clock, the other for data. Most modems use serial data transfer.

There's two kinds of serial data transfer, synchronous and asynchronous. Timing is very important in both kinds of transfer. In Asynchronous transmission, the data is sent one character at a time. Each character consists of a set number of bits, plus two extra control bits. The length of time between each bit is identical, although the time between characters may vary. Both the sending and receiving computers keep track of time with their own clocks.

For example, let's say that one computer has three characters to send to another computer. When it sends the first character, it will send the bits one at a time, with an equal amount of time between each bit. It may take a longer pause before it begins to send the second character, but when it does, the amount of time between each bit will be exactly the same as it ws before. It may decide to send the third character right away, but once again, the amount of time between each bit will be the same.

The two extra control bits are first bit of every character, called a start bit, and the last bit of every character, called a stop bit. The start and stop bits identify the beginning and end of every character (or byte) of data. These control bits keep both computers in sync.

In synchronous communication the message is sent as a continuous string of characters. Each character is a set numberof bits. The first set of data is a series of control codes. These codes synchronize the receiver's clock and the senders clock, so both computers are exactly together in time. Or, they could share a single clock, called a "common clock". The receiving computer can tell when a character starts and stops by the timing.

Synchronous transmission is faster than asynchronous transmission. An extra block of data is sent at the beginning of a synchronous message, but the message itself usually has fewer bits than an asynchronous message. A synchronous message doesn't need the extra control bits for every character. Since there's fewer bits, it's faster to send.

Synchronous circuits are also easier to build, because they do not require a separate clock in the receiver or decoder.

In the circuits here, we used a 75LS164 shift register to decode the data.

Shift registers are really quite simple. A shift register has eight registers, or boxes. Each piece of data is one byte (eight bits) long. The data is sent with the least significant bit first, the most significant bit last.

The clock tells the shift register to get a data bit. The shift register looks into the data line, and takes the bit. It puts the bit into the first register, or box (see diagram).

The clock signals the shift register again. The register takes the first bit out of the first box, and puts it into the second box. Then it goes to the data line, takes the next bit, and puts it into the first box (see diagram).

When the third bit is ready, the register moves the first two bits over, then puts bit #3 in the first box. When all eight bits have been sent, all eight boxes will be full (see diagram).

When the first bit of the next byte is sent, the register must move everything over so it can go in the first box. This time, however, all eight boxes are full. It will take the very first bit we sent, dump it out of the box, an throw it away. Now the register moves everything over, and puts the new bit in the first box.

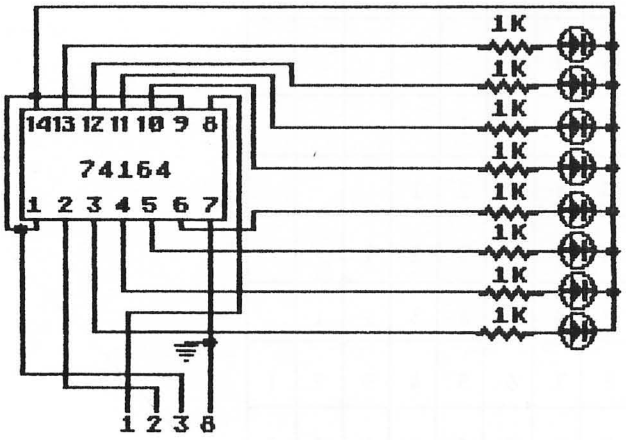
1							
2	1						
3	2	1					
4	3	2	1				
5	4	3	2	1			
6	5	4	3	2	1		
7	6	5	4	3	2	1	
8	7	6	5	4	3	2	1
9	8	7	6	5	4	3	2

Last bit (1) lost

SHIFT REGISTER

Parts

1	74LS164 Shift Register
8	LEDs
8	1K Resistors
1	DB-9 female socket



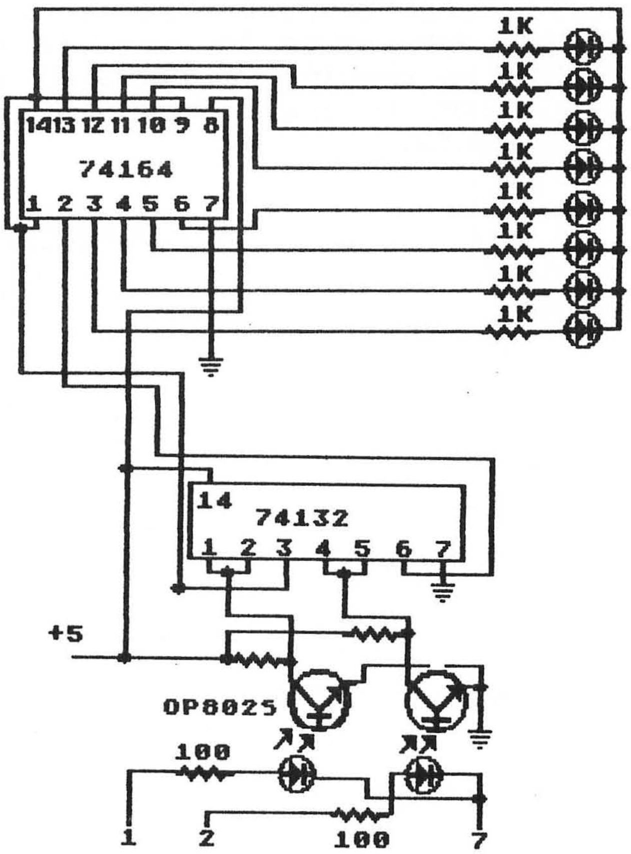
This register uses pins one and two in joystick port 1 to send the data. Pin one is used as a clock, and pin two carries the data.

One LED is attached to each of the IC's parallel data output pins. They will show you the data in each of the eight boxes. Attach the cathode end of the LEDs to the gate of the IC. The LED will light up when the output of the gate is low. The software inverts the data before it's sent, so a lit LED can be interpreted as a 1.

The register will take a new data bit whenever the clock pin goes low, then high.

SHIFT REGISTER WITH INFARED

This circuit works like the shift register above, except that it can transfer data over infared beams.



The Software

The programs SHIFT8 and SHIFT16 work with the first shift register. They read the data, invert it, and send it through pin 2 to to the shift register. They use pin 1 as a clock. SHIFT8 will send any number from 0-255. SHIFT16 will send any number from 0-65535. The program INFARED works with the Infared Shift Register.

Chapter 20 Networking

A network is a system of computers that are connected together. They can communicate and share data with each other. Every network is set up differently. Our network uses parallel data transfer.

As you remember from Chapter 19, computers can transmit data in two ways, serial or parallel. In serial communication, the data is sent one bit at a time. In parallel communication, several bits are sent at once. In our network, we send four data bits at a time (one half a byte, or a nibble).

In our network, the computers use two pins to send control codes to each other. The process of exchanging control codes and other information about the data is called "handshaking". Every network uses it's own kind of handshake, called the Network Protocol or the Communication Protocol, or just the Protocol.

Our protocol works like this. Pins 5 & 6 of one computer are connected to pins 6 & 5 of the other computer. Each computer sends it's handshake through pin 6, and gets a handshake through pin 5. Both computers send through 6, and receive through 5. This is why pin 6 of one computer is connected to pin 5 of the other. You could think of it as two telephones. You talk into the mouthpiece (bit 6) on your phone. Your friend hears you on the earpiece (bit 5) of his phone. He talks to you through his mouthpiece (bit 6), and you hear him in your earpiece (bit 5).

The protocol begins with bit 6, (pin 6, the trigger pin), low. Remember, pin 6 of one computer is connected to pin 5 of the other. Since both bit 6s are low, both bit 5s are low, too. This means that neither computer has any data to send.

Now, let's say computer #1 has data to send. It sends the data down the data line, and it sets bit 6 high. When it sets bit 6 high, computer #2, sees it's own bit 5 go high. This tells computer #2 that data is coming.

Computer #2 goes to the data line and takes the four bits of data. Then it "tells" computer #1 that it has the data, by setting it's own bit 6 high.

When computer #2 sets bit 6 high, computer #1's bit 5 goes high. Computer #1 knows that computer #2 has gotten the first nibble, so computer #1 sends the second nibble, and sets it's bit 6 low.

Computer #2 sees it's bit 5 change from high to low. It knows that the second nibble is in the data line, so it goes and gets it. Then computer #2 sets it's bit 6 low.

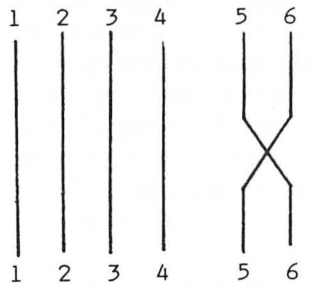
When computer #1 sees it's own bit 5 go from high to low, it knows that computer #2 has received the data. Computer #1 has no more data, so we're back to the beginning. If one of the computers needs to send data, it will set bit 6 to high, and the process will begin again.

You may be wondering why we sent only four bits, or 1/2 byte, at a time, when the joystick port has 9 pins. Pins 1-4 are used for data. These are the four bits. Pins 5 & 6 are used for handshaking, or the protocol. Pin 8 is the ground. That leaves pins 7 & 9. The most data that could be sent at once is six bits. It would still take two passes tosend a whole byte. So, we decided to save pins 7 & 9 to use in protocol for networks with more that 2 computers.

The diagram below is an example of the protocol our network uses.

Computer #1	Computer #2
Has data to send	
sets bit #6 high	
	Sees bit #5 go high
Sends data	
	Gets data
	Sets bit # 6 high
Sees bit #5 go high	
Sets bit #6 low	Sees bit #5 go low
	Sees bit #5 go low
Sends data	
	Gets data
	Sets bit #6 low
Sees bit #5 go low	

Below is the wiring diagram.



The Software

The network program on the disk is called TALK. The source code is called TALK.SRC. TALK lets both computers in your network communicate with each other. It also recognizes special commands that can be sent from one computer to the other.

TALK consists of two parts, a machine language subroutine that sends and receives the data, and a BASIC program that process the data. The subroutine and the BASIC program use their own sort of protocol to communicate.

When the subroutine gets a byte of data, it stores the data in location 1789. Then it sets location 204 to 1.

When location 204 is 1, the BASIC program realizes data is waiting in location 1789. It goes to 1789 and removes the data. Then it sets location 204 back to 0. When location 204 is 0, the subroutine knows it's OK to put more data into location 1789.

The subroutine also looks for the special character \, called a backslash. This character is used to signal that a control code is coming. When it sees this character, it checks the next two bytes. If a control code is coming, the second byte will be the name of the function, and the third byte will be a carriage return. If the first byte is a backslash, but the third byte is NOT a carriage return, the backslash and second byte are treated as normal data. If the third byte is a carriage return, then it's treated as a special control code.

When the subroutine spots a special control code, it sets location 205 to 1, and puts the name of the control code in location 1790. When the BASIC program sees location 205 turn to 1, it knows a control code has been sent. Then it looks in location 1790 to find the control code. Lastly, the BASIC program sets location 205 back to 0, so the subroutine will know that the BASIC program has the control code.

Now the BASIC program must decide what the control code is, and what it must do. It takes the ATASCII value of the control code, and multiplys it by 100. Then it takes the result, and goes to the line number that equals the result. The instructions for that function begin at that *line number*.

We built four special functions into the BASIC program, but you could add many, many more. Just pick a control character for your function. Then multiply the character's ATASCII value by 100. Use that number as the first line number of your subroutine. Below is a list of the built in functions.

You make a four computer network, using a fifth computer and pins 7 & 9 to control it.

When you run the program, do not press START until both computers say to do so.

Appendix A
Parts Suppliers & PC Board Services

PARTS SUPPLIERS

Acive Electronics
PO Box 9100
Westborough, MA 01581

All Electronics Corp
905 S Vermont Ave
PO Box 20406
Los Angeles, CA 90006

B & C Computer Visions
3283 Kifer Rd
Santa Clara, CA 95051
(408) 749-1003

Best Electronics
2021 The Alameda, Suite 290
San Jose, CA 95126
(408) 243-6950

Copper Electronics
4200 Produce Rd
Louisville, KY 40218

DIGI-Key
Highway 32 South
Thief River Falls, MN 56701

Heathkit
Dept 021-892
Benton Harbor, MI 49022

Hitachi Amarica Ltd
2210 O'Toole Ave
San Jose, CA 95131

Jameco Electronics
1355 Shoreway Rd
Belmont, CA 94002

JDR Microdevices
1224 S Bascom Ave
San Jose, CA 95128

Mark V Electronics
248 East Main St Suite 100
Alhambra, CA 91801

Mouser Electronics
2401 Hwy 287 North
Mansfield, TX 76063

RF Electronics
1056 N State College Blvd Dept R
Anahiem, CA 92806

PC BOARD SERVICES

Kit Circuits
Box 235
Clawson, MI 48017

Express Circuits
314 Cothren St PO Box 58
Wilkesboto, NC 28697

T.O.R.C.C.
Box 47148
Chicago, IL 60647

Appendix B
Programs

DECODE - Works with the decoder circuits

DETECT - Displays the bit values for the joystick pins, for use with event detectors

DLDISPLAY - Provides information about the display list

ENCODE - Works with the encoder circuits

FREQUNCY - Works with the tone deocders to identify sound frequencies

INFRARED - Transmits data serially with infared signals

LGHTMUSC - uses the light pen to play tones

LIGHTS - runs a light display

LOGIC - Logic gate and truth table demonstration

OVERLY1 - overlay program that displays the status of the joystick ports

OVERLY2 - same as above, for use with the data selector

OVERMAKE - creates custom overlay routines

PENDULUM - works with a light detector to provide information about a pendulum

POTMETER - works with the potentiometer

SHIFT8 - transmits an 8-bit number serially

SHIFT16 - transmits a 16-bit number serially

SPDACCEL - calculates the speed and acceleration of an object

SPEED - calculates the speed of an object

STOPWTCH - a simple timer program

SWITCH - works with switches

TALK - network software

USRMAKER - creates Machine lanuguage subroutines

WAVEFORM - used with tone generators to create sound wave forms

OVERLY1.SRC - Source code for OVERLY1

OVERLY2.SRC - Source code for OVERLY2

SHIFT.SRC - Source code for bith SHIFT programs

SH8.SRC - Source code for SHIFT8 display

SH16.SRC - Source code for SHIFT16 display

TALK.SRC - Source code for TALK

devices. The devices send coded information to the decoder, which translates the information and tells the computer which device is signaling.

Detector - A device which senses that an event has occurred, and signals the computer.

Digital Data - Data which can have any distinct value over a range of distinct values. It cannot have a value outside the range of possible values, and it cannot have a value that is inside the range, but is not one of the possible values.

Diodes - A semi-conductor that allows current to flow in one direction only.In this book, we used recitifier diodes, and LEDs.

Display List - A list of data which tells the ANTIC chip what to display on the monitor screen.

Display List Interrupt - A machine language routine that runs when the end of a certain screen line is displayed, and before the next line begins. DLIs run every 1/60th of a second.

Display Memory - The location in memory of the display list.

Encoder - An interface between a computer and several other devices. It receives information from the computer, puts it into a code, and transmits it to the other devices. Only the device that can receive that code can act upon the information.

Envelope - Part of a sound wave which defines its overall shape.

Frequency - A measure, usually how often something happens in a given period of time. In this book we used it to measure pendulum motion and sound. In pendulum motion, it is the number of cycles per second. In sound, it is a measure of the

Glossary

Acceleration - A change in speed, usually an increase.

Amplifier - Raises the volume of a sound, makes it louder.

Analog Data - Data that can have any value (including fractions) over a continuous range of values.

Asynchronous - A form of serial data transfer in which each character has a start and stop bit, and the timing between characters is variable.

Binary numbers - Numbers in the base two system, consisting of 0's and 1's.

Bob - Part of a pendulum, suspended from a string.

Capacitance - The ability to store an electrical charge. Used here as the measure of a capacitors ability to hold a charge.

Capacitors - An element, usually consisting of two metal plates separated by a dielectric, used to temporarily store an electric charge.

Comparator - In this book, an element which compares one voltage, called a reference voltage, to an incoming voltage.

CRT - Cathode Ray Tube, a normal TV or monitor.

Cycle - In this book, a measurement of a pendulums motion.

Data Selector - A device which allows you to plug two devices into a single joystick port.

Decoder - An interface between a computer and several other sound wave. For the sound projects in this book, frequency means pitch.

Hexidecimal numbers - Base 16 numbers, using the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, & F.

Infrared - Light waves that cannot be seen with the human eye, with waves longer than those of visible light, but shorter than microwaves.

IC - Integrated Circuit, a small piece of material that contains a complete electronic circuit.

LED - Light Emitting Diode, a semi-conductor diode that gives off light when voltage is applied.

Logic Gates - Gates which compare two separate inputs, and close or open depending upon the inputs and the comparison criteria.

Mass - The amount of matter in a body. Mass is different from, but proportional to, weight.

Networking - The process of connecting two or more computers together so that they can work as a team or separately, they can communicate and share the same data.

Ohm's Law - E=IxR, where E is the voltage in volts, I is the current in amps, and R is the resistance in ohms. This useful law and formula allows you to calculate the third variable, based on the other two.

Overlays - A small area or window on the screen, separate from the rest of the screen.

Overtones - Components of a sound wave whose wavelength is significantly shorter than the length of the main sound wave.

Parallel Data Transfer - The process of transferring data between a computer and another computer or device, sending several bits simultaneously, over several separate wires.

Pendulum - A body suspended on a string from a fixed point that swings freely under the influence of gravity.

Photocell - An electronic device whose voltage output varies according to the amount of light rays touching the surface.

Phototransistor - A semi-conductor that performs the same functions as an electron tube, but smaller, and activated by light.

Pitch - The degree of highness or lowness of a sound.

Pixel - A small dot-like unit fixed in place on the monitor screen. It glows when touched by electrons.

Protocol - A standard procedure, the process two computers must follow to communicate or exchange data.

Recitifier - A semi-conductor junction which converts alternating current (AC) to to direct current (DC) by allowing more current to flow in one direction than the other. A Halfwave Recitifier transmits only one polarity of the alternating current, producing a pulsating direct current. Two halfwave recitifiers combined create a full wave recitifier, giving a continuous pulse which may be smoothed by a filter.

Relays - Electronic devices which channel current.

Resistors - Electronic device which resists the passing of a current.

Reverbrations - Component of a sound wave, the repeat vibrations and echoes of the sound, which can be heard at the same time as the original sound.

Serial Data Transfer - The process o transmitting data between computers and other devices one bit at a time.

Soldering - Applying a molton metal alloy to a joint to cement it.

Sound Waves - The patterns of vibrations made by sounds.

Speed - How fast an object is moving, measured in units of distance per units of time.

Transistors - Electronic devices which perform the same functions as electron tubes, but are much smaller.

Truth Tables - Tables which show the results of the comparison of two input states. The result depends upon both input states, and the criteria used for comparison.

Undertones - Components of a sound waves whose wavelenghts are significantly longer than the main wave length.

USR Function - A BASIC command which allows you to call a machine language subroutine from a BASIC program.

VDT - Video Display Terminal, and ordinary monitor.

Voltage - Electromotive force, measured in volts.

LIMITED WARRANTY

Alpha Systems warrants the original purchaser of this computer software product that the recording medium on which the software programs are recorded will be free from defects in materials and workmanship for ninety days from the date of purchase. Defective media returned by the purchaser during that ninety day period will be replaced without charge, provided that the returned media have not been subjected to misuse, damage, or excessive wear.

Following the initial ninety day warranty period, defective media will be replaced for a replacement fee of $6.50.

Defective media should be returned to:

ALPHA SYSTEMS
1012 SKYLAND DRIVE
MACEDONIA, OH. 44056

in protective packaging accompanied by: (1) a brief statement describing the defect; (2) a $6.50 check or money order (if beyond the ninety day warranty period); (3) your return address; (4) the problem disk.

What is Not Covered by this Warranty

This warranty does not apply to the software programs themselves. the programs are provided "as is".

This warranty is in lieu of all other warranties, whether oral or written, express or implied. Any implied warranties, including imputed warranties of merchantability and fitness for a particular purpose, are limited in duration to ninety days from the date of purchase. Alpha Systems shall not be liable for incidental or consequential damage for breach of any express or implied warranty.

The provisions of the foregoing warranty are subject to the laws of the state in which the disk is purchased. Such laws may broaden the warranty protection available to the purchaser of the disk.

Tell Us What You Think

We at Alpha Systems are sincerely interested in bringing you the best possible products at the lowest possible prices. Please write us if you experience any difficulties with our products, or have any comments or ideas for improvements. We will do our best to make our products better meet your needs. When you write, please enclose the following: 1) Your name, address, and phone number. 2) Your comments, or a description of your problem. 3) A description of your system. 4) If you are reporting a problem, please also include a description of what you were doing when the problem occurred, any printouts or other output showing the problem if possible, and any suggestions you may have regarding the cause and solution.

Your Atari Comes Alive

<u>DIGITIZE YOUR WORLD</u>

Now for the first time ever you can connect your 8-bit Atari to a wide range of external interfaces, devices, and sensors. This "How-To" book and disk package has everything you need to build and use a wide variety of unique devices. It details the construction, use, and programming of

Light Pens
Alarm Systems
Networks
Infrared Sensors
Light and Motor Controls
Light and Motion Sensors
Data Encoders and Decoders
Tone Decoders

The commercial versions of some of these devices sell for hundreds of dollars, but this book will show you how to build your own for a fraction of the cost.

<u>Your Atari Comes Alive</u> provides a thorough overview of electronics, as well as detailed schematics and instructions. It covers the programming techniques used to control the devices, gives building, testing, and proramming tips, and offers suggestions for additional applications, systems, and software. The disk contains over 18 listable, ready to run BASIC and Assembly language programs, to communicate with your devices.

Create your own exciting devices
and save money!!

ALPHA SYSTEMS