

# **Easy Add-on Projects for Spectrum, ZX81 & Ace**

OWEN BISHOP





**EASY ADD-ON PROJECTS  
FOR SPECTRUM, ZX81 & ACE**

## OTHER TITLES BY THE SAME AUTHOR

- BP73 Remote Control Projects
- BP82 Electronic Projects Using Solar Cells
- BP104 Electronic Science Projects
- BP116 Electronic Toys Games and Puzzles

## OTHER BOOKS OF INTEREST

- BP109 The Art of Programming the 1K ZX81
- BP114 The Art of Programming the 16K ZX81
- BP119 The Art of Programming the ZX Spectrum
- BP128 20 Programs for the Spectrum and 16K ZX81

**EASY ADD-ON PROJECTS  
FOR SPECTRUM, ZX81 & ACE**

**by  
OWEN BISHOP**

**BERNARD BABANI (publishing) LTD  
THE GRAMPIANS  
SHEPHERDS BUSH ROAD  
LONDON W6 7NF  
ENGLAND**

## PLEASE NOTE

Although every care has been taken with the production of this book to ensure that any projects, designs, modifications and/or programs etc. contained herein, operate in a correct and safe manner and also that any components specified are normally available in Great Britain, the Publishers do not accept responsibility in any way for the failure, including fault in design, of any project, design, modification or program to work correctly or to cause damage to any other equipment that it may be connected to or used in conjunction with, or in respect of any other damage or injury that may be so caused, nor do the Publishers accept responsibility in any way for the failure to obtain specified components.

Notice is also given that if equipment that is still under warranty is modified in any way or used or connected with home-built equipment then that warranty may be void.

All the projects in this book have been designed and tested by the Author using models of the Spectrum, ZX81 and Ace microcomputers that were available at the time of writing in Great Britain. The Author and Publishers accept no responsibility for failure of a project to operate or any damage that may be so caused, should the manufacturers of the microcomputers radically change the specification of their machines in the future or for a particular overseas market.

© 1983 BERNARD BABANI (publishing) LTD

First Published – August 1983

British Library Cataloguing in Publication Data  
Bishop, O. N.

Easy add-on projects for Spectrum, ZX81 & Ace. —  
(BP124)

1. Sinclair computers

I. Title

001.64'04 QA76.8.S625

ISBN 0 85934 099 6

Printed and bound in Great Britain by Cox & Wyman Ltd, Reading

# CONTENTS

	<i>Page</i>
INTRODUCTION . . . . .	1
Interfacing to microcomputers . . . . .	1
Logic levels in computers . . . . .	1
The microprocessor . . . . .	2
Address decoder . . . . .	6
Using the ZX Computers . . . . .	8
Using the Jupiter Ace . . . . .	9
Using other Computers . . . . .	10
PROJECT 1: PULSE DETECTOR . . . . .	11
How it works . . . . .	11
Building it . . . . .	13
Using it . . . . .	13
Detecting a 'high pulse' . . . . .	14
PROJECT 2: PICTURE DIGITISER . . . . .	16
How it works . . . . .	16
Addressing . . . . .	21
Building it . . . . .	22
Final check . . . . .	26
Test program . . . . .	27
Programming . . . . .	28
PROJECT 3: FIVE-KEY PAD . . . . .	35
How it works . . . . .	37
Addressing . . . . .	39
Building it . . . . .	39
Testing . . . . .	41
PROJECT 4: MODEL CONTROLLER . . . . .	44
How it works . . . . .	44
Building it . . . . .	48
Programming . . . . .	51
PROJECT 5: BLEEPER . . . . .	54
How it works . . . . .	54

Building it . . . . .	56
Programming . . . . .	59
 PROJECT 6: LAMP FLASHER . . . . .	 61
How it works . . . . .	61
Building it . . . . .	64
Programming . . . . .	64
 PROJECT 7: LIGHT PEN . . . . .	 66
How it works . . . . .	66
Building it . . . . .	68
Programming . . . . .	71
 PROJECT 8: MAGNETIC CATCH . . . . .	 78
How it works . . . . .	78
Building it . . . . .	84
Programming . . . . .	85
 PROJECT 9: LAP SENSOR . . . . .	 87
How it works . . . . .	87
Building it . . . . .	90
Programming . . . . .	91
 PROJECT 10: PHOTO-FLASH . . . . .	 94
How it works . . . . .	95
Programming . . . . .	99
 PROJECT 11: GAMES CONTROL . . . . .	 102
How it works . . . . .	102
Building it . . . . .	104
Programming . . . . .	106
 PROJECT 12: RAIN DETECTOR . . . . .	 111
How it works . . . . .	111
Building it . . . . .	111
Programming . . . . .	113
 PROJECT 13: WEATHERCOCK . . . . .	 115



	<i>Page</i>
How it works . . . . .	115
Building the vane . . . . .	118
Addressing . . . . .	119
Building the circuit . . . . .	121
Programming . . . . .	121
 PROJECT 14: ANEMOMETER . . . . .	 124
How it works . . . . .	125
Building it . . . . .	127
Programming . . . . .	129
 PROJECT 15: THERMOMETER * . . . . .	 131
How it works . . . . .	131
Building it . . . . .	131
Calibration and programming . . . . .	133
 PROJECT 16: BAROMETER . . . . .	 137
How it works . . . . .	137
Building it . . . . .	141
Programming . . . . .	144
 PROJECT 17: SUNSHINE RECORDER . . . . .	 149
How it works . . . . .	149
Building it . . . . .	152
 Appendix A: THE ADDRESS DECODER . . . . .	 157
Building the project . . . . .	162
Controlling the decoder . . . . .	173
Programming the ZX81 . . . . .	174
Programming the Jupiter Ace . . . . .	178
 Appendix B: PIN LEAD-OUT DETAILS . . . . .	 180
 Special note for readers in USA . . . . .	 182



# INTRODUCTION

## Interfacing to microcomputers

This book describes how to build electronic circuits which you can attach to your ZX Spectrum, ZX81 or Jupiter Ace. All the devices have been tested with all of these computers. Most of the circuits probably work just as well when connected to a ZX80 micro or could be adapted to do so fairly easily, but they have not been tested with this computer, and full connecting details are not given here.

All the projects are simple and inexpensive ones, requiring only a few transistors or integrated circuits. The integrated circuits are of the less expensive kind. The wiring required for each circuit has been kept to a minimum, an unusual feature for a book of this kind, for microcomputer projects tends to need rather more wires than projects of other kinds. The result of this simplified approach used here is that all the projects can be built easily and cheaply, even by a relative beginner. Once built, they are simple to operate. Writing programs to control them need not be complicated. Each project includes a simple program or two to get you started. Of course, those readers who are more expert at programming can have a lot of fun in writing elaborate programs based on these projects, but the beginner can start with the short program and perhaps add extra features later.

The circuit designs apply equally well to the ZX81, ZX-Spectrum and the Ace. If you have a ZX81 now and later buy a ZX-Spectrum, the projects can be operated from your new computer without having to change them. You may need to modify the program slightly, since the Spectrum has a more extensive BASIC than the ZX81. The differences are explained later.

## Logic levels in computers

Computers deal with numbers and other information as a series of 0s and 1s. The projects show many examples of this.

The 0s and 1s, or binary digits (shortened to “bits”) are represented in the computer by voltage levels.

0 is represented by 0V, we refer to this as “low”.

1 is represented by +5V, we refer to this as “high”.

Numbers and other kinds of information, including instructions, are handled in the computer by making one of these two voltages (0V or +5V) appear on different wires (or lines).

## **The microprocessor**

Although these computers may differ in appearance and in their facilities, their microprocessor (the Z80) is identical in each.

The ZX micros and the Ace allow us to communicate directly with the microprocessor itself, through the input/output port. This is the set of metallized pads (Figs.0.1 and 0.2) on the edge of the circuit board, protruding from the rear of the machine. This is where you attach your printer or extra RAM. This is where you will soon be attaching some of the projects from this book. These are the connections used:

*The address bus:* This is a set of 16 lines, but we use only 8 of them here (the lower 8 lines, numbered from A0 to A7). The Z80 uses the address bus to inform any other part of the micro or any attached device (such as a project) that the Z80 wants to tell that part or device to do something or to receive information from that part or device. It does this by putting 0V or +5V on different lines of the bus. Each part or device is allocated its own address, and knows when its address is on the bus. Then it becomes active and does what the Z80 wants it to do. How it finds out that its address is on the bus is explained later. The point of having an address bus is that the Z80 can communicate with all other parts of the system, but with only one part at a time. Each part responds to the message intended for it and not to the messages directed to other parts.

*Data bus:* A set of 8 lines, of which we use only 4 (D0 to D3) in this book. Many of the projects use only one data line (D0).

Only connections required for the projects in this book are labelled. With respect to the key-way, these connections are in exactly the same positions in both ZX-81 and ZX-Spectrum.  
(N.B. some of the other connections, not shown here because they are not used, may not be in identical positions.)

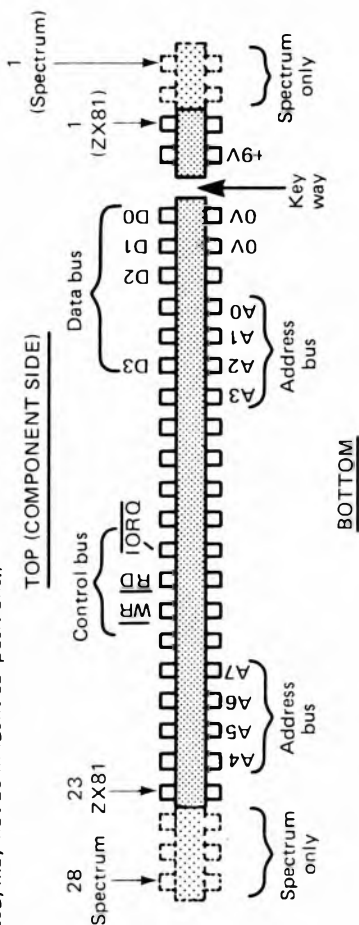
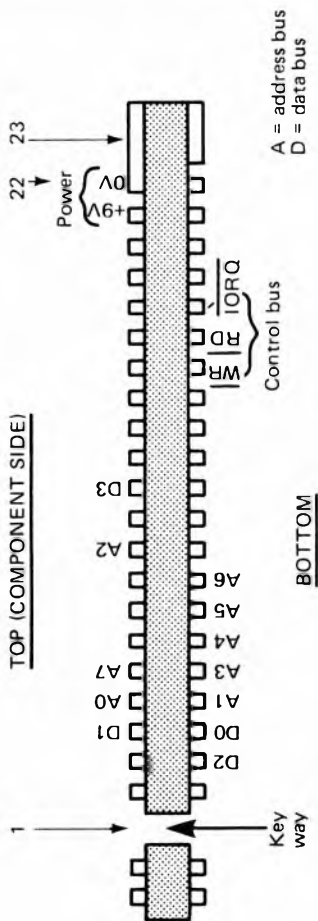


Fig. 0.1 The input-output port connector of ZX micros as seen from the rear of the machine



The data bus is used for transferring data (instructions or numbers) between the Z80 and other parts of the system (including projects). The data is represented by the voltage levels put on to its lines by the Z80 when it is sending data to another part of the system, or put there by another part of the system when it is sending data to the Z80.

*Input-output-request line:* This is one of the control lines, usually called  $\overline{\text{IORQ}}$  for short. The voltage level on this line is normally +5V (high) but, when the Z80 wants to communicate with a device which is attached to the micro (such as a project), it makes the level low (0V). The line over the letters IORQ mean that the voltage goes *low* when something is to happen; we say it is "*active-low*". The low level on  $\overline{\text{IORQ}}$  alerts the device to the fact that the Z80 has already placed an address on the address bus, and the device must check to see if this is its own address. The Z80 has another control line called "memory-request" ( $\overline{\text{MEMRQ}}$ ), which it uses when it wants to communicate with its memory. We do not use  $\overline{\text{MEMRQ}}$  in the circuits of this book. Because of this, the projects can not interfere with the normal working of the memory of the computer.

*Read line ( $\overline{\text{RD}}$ ):* This is another control line and is active low. The Z80 puts a low level on this line to indicate when it wants the addressed device to send data to it. The device should respond by putting the data on the data bus, so that the Z80 can *read* it.

*Write line ( $\overline{\text{WR}}$ ):* The Z80 puts a low level on this line when it wants to send data to a device.

*Power lines:* We use two of these. All circuits need to be connected to the 0V line of the micro, because this is the line against which all voltage levels are to be measured. The micros have a regulated +5V line which could be used to power the circuits of the projects. The difficulty is that there is usually not much power to spare, particularly if several devices are connected to the system at the same time. Instead, we use the

+9V line, which comes directly from the ZX or Ace Power Pack. This is an unregulated supply, but a voltage regulator IC is used to convert this to a regulated +5V supply.

## Address decoder

Every device that we attach to the computer must have an address decoder. This is a logic circuit which detects the voltage level on each line of the address bus and works out if these correspond to the address of the device concerned. The projects use 8 address lines, and the address line voltages for a given project might be:

Line	A7	A6	A5	A4	A3	A2	A1	A0
Voltage	5	0	0	5	5	5	5	5
Bit	1	0	0	1	1	1	1	1

In the second row above, the voltages are listed as we might measure them with a voltmeter (but see p. 11). The third row shows their logical equivalents. These 0s and 1s can be written out as a binary number, 1 0 0 1 1 1 1 1. Another way of writing this number is as a hexadecimal, in which form it becomes 9F (see Table 0.1). The decimal equivalent of this is 159. If a device has the address 159 then its address decoder logic should respond when the voltage levels are as shown above. For all other combinations of voltage levels it should show no response of any kind.

The logic of the address decoder must also take into account the levels present on the IORQ, RD and WR control lines. An address decoder is not a simple circuit yet, because it is essential for every project, it is something we can not do without. Fortunately there is a relatively simple way of overcoming the problem. Building it with a PCB design as given on p. 166 is easy, and it is not necessary for you to understand how it works to make full use of it.

The decoder provides the address decoding for all the other projects. It also provides a regulated +5V power supply and has several other useful features. Once you have built the decoder you rarely need to bother about address decoding again, and the construction of the projects is therefore made



*Table 0.1 Some binary numbers and their equivalents*

	<i>Binary</i>	<i>Hexadecimal</i>	<i>Decimal</i>
	0	00	0
	1	01	1
	10	02	2
	11	03	3
	100	04	4
	101	05	5
	110	06	6
	111	07	7
	1000	08	8
	1001	09	9
	1010	0A	10
	1011	0B	11
	1100	0C	12
	1101	0D	13
	1110	0E	14
	1111	0F	15
	1 0000	10	16
	1 0001	11	17
	1 0010	12	18
.....			
*	1 1111	1F	31
	10 0000	20	32
	11 0000	30	48
*	11 1111	3F	63
	100 0000	40	64
*	101 1111	5F	95
*	111 1111	7F	127
	1000 0000	80	128
*	1001 1111	9F	159
*	1011 1111	BF	191
*	1101 1111	DF	223
*	1111 1111	FF	255

---

\* these are the 8 addresses used for the projects

much simpler. The decoder itself is not *really* complicated, for it is easy to understand how it works and, if correctly built, should work perfectly first time. The main complication is that it has a lot of wires, so you need to take care to make the correct connections. Fig. D.4 (p. 166) is a printed circuit board master for this project to help you get the connections right. It is also possible to buy this PCB ready-made (see address of supplier, p. 179).

To begin interfacing to your computer, build the decoder. After that, you can choose which of the other projects you want to make and connect them to the decoder. You can connect several projects at once, if you wish, yet control them individually.

## Using the ZX Computers

Both of the Spectrum and ZX81 computers use Sinclair BASIC, so programs written for one may be adapted for the other fairly easily. One important way they differ is that the Spectrum has the IN and OUT commands which allow it to read or write data directly to the input/output port. It is by these commands that we control the interfaces. The ZX81 lacks these two commands. Instead of these, we use simple machine-code routines, as explained on p. 174.

Of the devices which provide data for the micro to read, most use only the lowest line of the data bus, line D0. It is a property of the logic circuits used in these micros that if a line is unconnected and is not being used, it behaves as if the input to it is 1. So when we are using only line D0, with lines D1 to D7 unconnected, an input of 0 on line D0 appears as:

1 1 1 1 1 1 1 0

All lines D1 to D7 are 'high'. On the Spectrum the IN command returns 254, the decimal equivalent of this binary number. When line D0 has a 'high' level on it, the result of a read operation is 1 1 1 1 1 1 1 1, which is 255 in decimal.

The ZX81 gives different results. The machine-code routine on p. 174, in conjunction with the command LET=USR 16514, returns a value which shows the contents of the B and

C registers of the microprocessor. The B register always holds 1 1 1 1 1 1 1 1 while the C register shows the state of the data bus, in the way described above. So if line D0 is 'low' we get:

1 1 1 1 1 1 1 1 1 1 1 1 1 0  
which is 65534 in decimal.

If D0 is 'high' we get:

1 1 1 1 1 1 1 1 1 1 1 1 1 1  
which is 65535 in decimal.

So in the descriptions of programs in this book, you need to add 65280 to values expected with the Spectrum to find what values are to be expected with the ZX81.

## Using the Jupiter Ace

As far as the projects in this book are concerned, the Ace provides all the connections required. The edge connector on the Ace board has exactly the same connections as on the ZX81 or Spectrum, but on the Ace they are in a different order. The design for the Decoder board therefore needs amending slightly, or you can make a simple adaptor which brings each line to its correct position for plugging into a Decoder designed for the ZX computers.

The main advantage of using the Ace is that its language, FORTH, gives it very high operating speed. This makes it particularly suitable for the pulse-timing operations required by several of the interfaces in this book. Whereas a machine-code program is essential with the ZX computers, for a few of the projects the Ace operates fast enough when using FORTH. The FORTH language was, after all, designed to be used in controlling astronomical telescopes so it is to be expected that it should excel in controlling other kinds of interface too.

The devices in this book have all been tested with the Ace and work as well on this machine as on the ZX Computers. There is one difference. On the Ace (or at the least on the model used during the writing of this book), unconnected lines usually read as 0, except for line D4 which reads as 1. Consequently, with input to line D0 only, the readings are 32

(0001 0000) or 33 (0001 0001) instead of the values found on the ZX computers. In practice it was found that the values on some of the unconnected data lines 'floated' occasionally, reading either as 1 or 0 at random. For this reason, it is preferable to AND the data input with the value 0000 0001 to eliminate any values appearing on the unconnected lines. The command required is:

31 IN 1 AND

where 31 is the address being read from. This sequence leaves 0 or 1 on the top of the stack, depending on the state of line D0.

## Using other computers

This book is written for three particular computers but all computers which are based on the Z80 or Z80A microprocessor are the same at heart. Common examples are the Exidy Sorcerer, Nascom 1 and 2, North Star Horizon, Research Machines 380Z, Sharp MX80, Superbrain, Transam Tuscan, TRS-80 and Video Genie. If you can make connections to the address bus, data bus and control bus of these machines, as required for the decoder (see Fig.D.1) there is no reason why you should not attempt to interface these machines with the circuits described in the book. However, it must be stressed that the circuits in this book have *not* been tested with any of the machines listed above, so there is no certainty that they will work. Neither can one be certain that they will not in some way affect the normal operation of the computer, though this is very unlikely. You should also check the terms of the guarantee issued by the manufacturer of the machine to ensure that you will not be invalidating it by connecting your own devices to it.

# Project 1

## PULSE DETECTOR

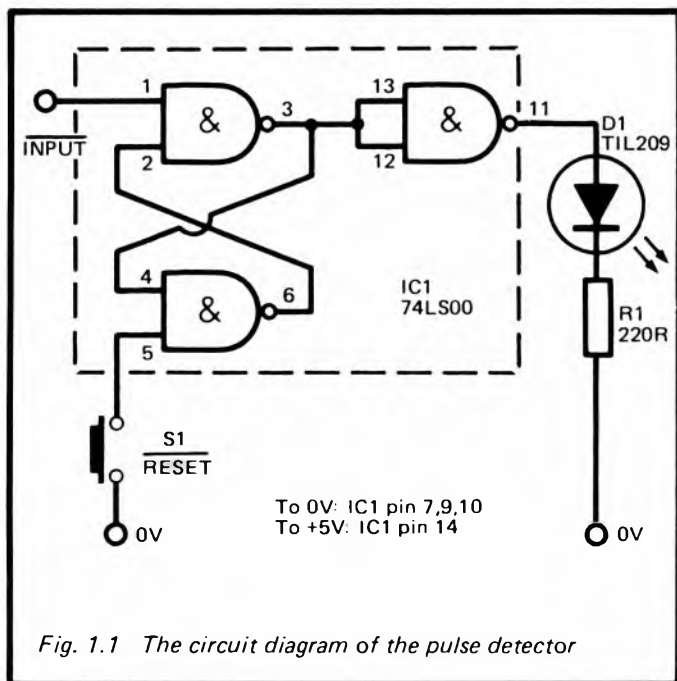
This simple device is a very useful one for testing the other interfaces. The trouble with micros is that they work very fast. The micro produces pulses of current which may last for only a fraction of a microsecond. This is far too short a time to affect an ordinary voltmeter. Even with an oscilloscope it is almost impossible to pick out the pulse you are interested in. This circuit is designed to detect a low pulse produced by a microcomputer. The words 'low pulse' mean that the voltage at the point of testing is normally 'high' (anything between +2V and +5V), but goes 'low' (falls below +0.8V) for a short period. This circuit is quick enough to detect low pulses lasting as little as a fiftieth of a microsecond, which is fast enough for almost all pulses that a microcomputer can produce.

### How it works

The detecting section of the circuit consists of two logic gates, wired as in Fig.1.1 to make a flip-flop. The output from this section (at pin 3) may be 'high' or 'low', but must be one or the other. Remember that anything between +2V and +5V counts as 'high', while anything between 0V and +0.8V counts as 'low'. Voltages can not be higher than +5V because that is the voltage at which we supply power to the circuit.

With the flip-flop in its 'reset' state, waiting for a pulse to arrive, the output at pin 3 is 'low'. Both of its inputs, the one labelled INPUT and the one labelled RESET are high. INPUT is high because it is connected to the point at which a low pulse is expected at any moment. Although RESET is not connected to anything (for the button S1 is not being pressed), it acts as if it is high. This is a property of the TTL integrated circuits used in most of the projects in this book. Any input which is unconnected acts as if it was receiving a high voltage level.

The 'low' output from pin 3 goes to another logic gate,



*Fig. 1.1 The circuit diagram of the pulse detector*

which 'inverts' it. That is to say, the output of this gate (at pin 11) is 'high'. The light-emitting diode, D1 is lit. You may wonder why we do not drive the LED directly from pin 3. The reason is that, if we do this, the flip-flop does not respond as quickly and might miss the short low pulse it is waiting for.

When the low pulse arrives, the flip-flop changes state. It becomes 'set'. The output at pin 3 goes 'high', which makes the output at pin 11 go 'low', and the LED goes out. To reset the flip-flop and put the LED on again, we press the 'reset' button for an instant. Provided that the pulse is finished and the voltage at INPUT has returned to a 'high' level, pressing S1 makes the flip-flop return to its 'reset' state.

## Building it

This project can be built up on any small scrap of strip-board. There is hardly any need to put it in a case but it is easy to find a small plastic container which will suit the purpose of enclosing it. The push-button and LED are mounted on the case.

There is no need for a power supply switch. The best arrangement is to have three wires about 20cm long to connect the Detector to the circuit (or the line of the micro) which is being tested. Solder a crocodile clip to the flying end of each wire. Preferably use clips with insulating plastic covers, and have clips of three different colours so that you know which is which. One of the wires is for connecting the Detector to the 0V line of the micro, or test circuit. This wire connects also the IC and to the other points shown on Fig.1.1. The second wire is for taking power from the +5V rail of the micro, or tested circuit. This wire connects to pin 14 of the IC. The third wire is the INPUT wire, going to pin 1.

## Using it

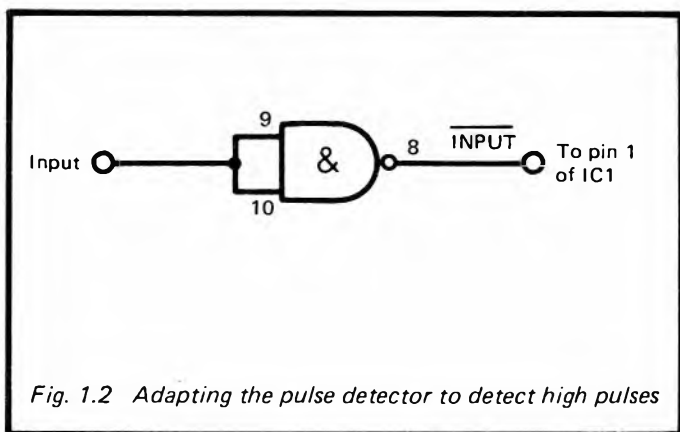
Switch off the micro or the circuit you are testing. Clip the 0V and +5V leads to the 0V and +5V rails of the micro or other circuit. Clip the INPUT wire to the line you want to test for low pulses. Then switch on the power supply.

The LED may or may not light to begin with, depending on which state the flip-flop goes to when switched on. If the LED does not light, press S1 for an instant. If the LED does not light then, it may be that the tested line is not actually 'high', or that a series of 'low' pulses are coming along it so often that the flip-flop becomes 'set' almost immediately after you reset it. If this is the case, there is nothing more you can do, for the detector is not designed to work with a rapid series of 'low' pulses.

If the LED comes on and stays on when you press S1, operate the micro or test circuit in the way which you believe is going to produce that single 'low' pulse. If or when the pulse arrives, the LED goes out.

## Detecting a 'high' pulse

Fig.1.2 shows how the Detector may be adapted to detect a 'high' pulse on a line that is normally 'low'. As it happens, you are more likely to be looking for 'low' pulses in most of the projects in this book, but occasionally the important pulse is a 'high' one. All that the gate of Fig.1.2 does is to invert the pulse before sending it on to the flip-flop. The line to which INPUT is connected must be one that is normally 'low'. The output from pin 8 will therefore normally be 'high'. When a 'high' pulse arrives at INPUT, pin 8 will give out a 'low' pulse and so trigger the flip-flop.



The gate used for the adapted version is the spare gate of IC1. Instead of joining its inputs to the 0V line, they are joined to each other and used to receive the INPUT signal. You could wire up your Detector so that it is possible to make temporary connections by means of crocodile clips whenever you want to look for 'high' pulses instead of 'low' ones.



## *PARTS REQUIRED for the PULSE DETECTOR*

### *Resistors*

R1 220R carbon, 0.25W, 5% tolerance

### *Semiconductor*

D1 TIL209 or similar light-emitting diode

### *Integrated Circuit*

IC1 74LS00 quadruple 2-input NAND gate

### *Miscellaneous*

S1 Push-to-make push-button switch

Scrap of stripboard (2.5mm matrix)

1mm terminal pins (5 off)

Crocodile clips, miniature size (3 off, of different colours)

(Better still are miniature clip-on probes, such as  
'E-Z-Hooks'.)

14-pin IC socket (optional, but recommended)

Small plastic case (optional)

Connecting wire

## **Project 2**

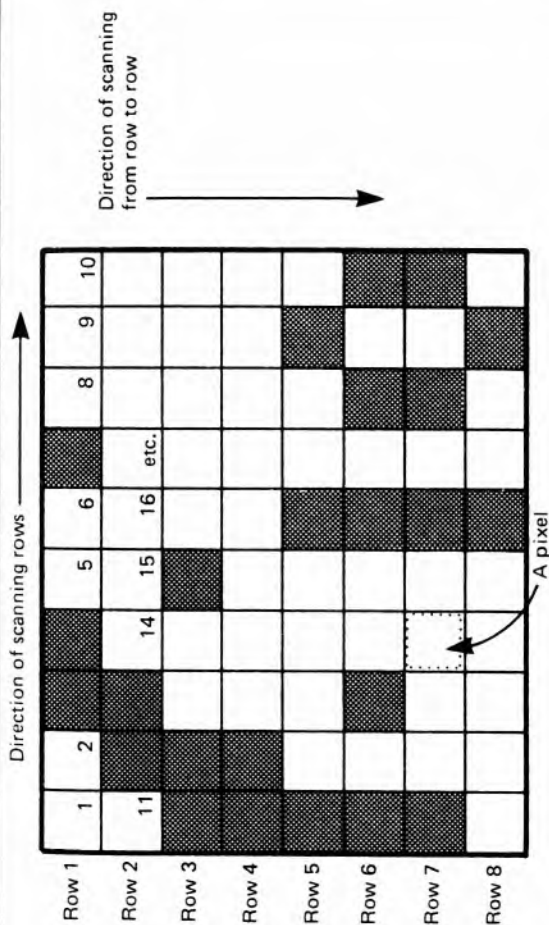
### **PICTURE DIGITISER**

Using this device, you can scan a photograph or drawing and see it appear on the screen of your micro. The scanner is rather a low resolution one, so it works best when the picture has large bold areas of contrasting shades. Fine detail will be lost but, provided that you choose a suitable subject, it is fascinating to see the result building up block by block on the screen. The scanner is moved by hand, so it works fairly slowly and depends on you to move it accurately. It clearly demonstrates the principle behind professionally-made (and highly expensive) picture digitising equipment. There is a great amount of fun to be had from the simple digitiser.

#### **How it works**

The idea behind this device is that the picture is broken down into a number of picture elements, or pixels. The picture area consists of rows and columns of pixels, as in Fig.2.1. The scanner moves (or, in this instance, is moved) along each row of pixels in turn. It measures the brightness of each pixel, and reports this to the micro. Brightness is something which could be anything between a brilliant white and the darkest of blacks. There is an unlimited number of shades of grey in between. If the picture is a coloured one, the different colours can be considered to be equivalent to different shades of grey.

The micro can not accept an unlimited number of different possible shades. To keep the circuit simple, this scanner is designed to recognise only 4 different shades. The lightest is white, the darkest is black and there are 2 shades of grey (light grey and dark grey or equivalent colours) in between. When the scanner measures the overall brightness of a pixel it tells the micro to which of these 4 possible shades the pixel shade is closest. The micro plots each pixel on the screen as it is reported. It plots small squares which show a corresponding range of brightness. White is a solid block of light on the



*Fig. 2.1 Part of the face of a clock (near 10 o'clock) drawn as a set of pixels.*

In this example the pixels are either black or white, but pixels can be in shades of grey or in colour. The numbers show the order in which the pixels are measured by a scanner and then plotted on a screen

screen, and black is a blank area of screen. The greys are represented by patterns of dots in which light grey has more white dots than dark grey.

The way you represent the greys depends on the graphics provided by your micro. The ZX81 has only one shade of grey, so both shades of grey have to be plotted the same, though the dark one could be plotted as black and the light one as white if this makes the picture better to look at. With the Spectrum you can define your own patterns with various proportions of dots, as described later, and possibly use BRIGHT 1 for white.

The circuit (Fig.2.2) relies on a photodiode (D1) to detect the light reflected from the picture. The light comes from a torch-bulb, which is mounted with the photodiode in the scanner (Fig.2.4). If light is falling on D1, a small leakage current flows through R1 and D1. The more light reaching D1, the larger the current. The larger the current, the greater the potential difference across R1. The greater the PD, the lower the voltage at point A. This is because the voltage at the top end of R1 is fixed at +5V, so any increase in PD across it must make the voltage fall at its other end.

The level of voltage at point A is measured by three comparators (IC1, IC2 and IC3). These each compare the voltage at their *inverting input* (—) with the voltage at their *non-inverting input* (+). In this circuit, the inverting inputs are all connected to point A, while the non-inverting inputs are each connected to the wiper of a variable resistor, (RV1, RV2 and RV3). Each resistor is set so that the voltage at its wiper has a fixed value between 0V and +5V. The setting is different for each variable resistor. For each comparator, if the voltage at the inverting input (from A) is higher than the voltage at the non-inverting input (from the variable resistor), the output of the comparator rises sharply toward +5V. If the voltage from A is lower than the voltage from the variable resistor, the output falls sharply toward 0V.

The variable resistors are set so that when D1 is receiving light reflected from white paper, the voltage at A is lower than the voltage from any of the variable resistors. The outputs of all three comparators are 'low'. If very little light is reaching

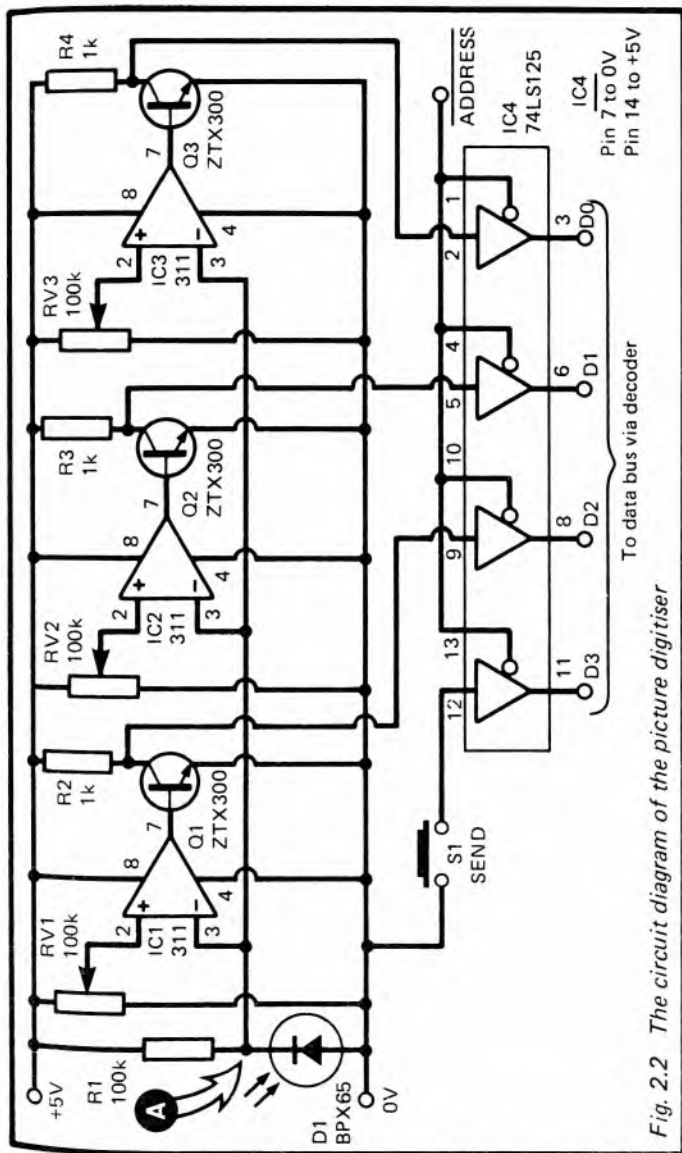


Fig. 2.2 The circuit diagram of the picture digitiser

D1 (reflection from black paper), the voltage from A is greater than any voltage from the resistors, so all outputs are 'high'. In between (shades of grey, or colours), one or more of the comparators has a 'high' output and the others have a 'low' output.

The output of each comparator goes to a transistor. When the comparator has a high output, the transistor is switched fully on. The voltage at its collector falls to 0V (= 'low'). The transistor *inverts* the output from the comparator. This voltage is fed to a buffer gate in IC4. When the buffer is enabled (see next section) and 'low' level is sent to the buffer, a 'low' level is put on to the data bus. Conversely, when the comparator has a 'low' output, a 'high' level appears on the data bus.

The various possible outputs of the circuit are listed in Table 2.1. The top section of the table shows what happens when the 'Send' button (S1, Fig.2.2) is *not* pressed. The output from its buffer is 'high' (see p. 11), so the micro reads the four data lines as a number greater than 7. This tells the micro that, whatever data may be appearing on the other lines, the scanner is not actually ready in position over a pixel, so it should ignore these other lines. When the 'Send' button is pressed, its buffer output goes 'low'. Now the micro interprets the other lines as levels of brightness, depending on how it has been programmed (see later).

*Table 2.1 Data output from the Picture Digitiser*

'Send' button	Shade of pixel	Data lines 3 2 1 0	Decimal equivalent	Value displayed*
Not pressed	Any	1 X X X	more than 7	more than 247
Pressed	White	0 1 1 1	7	247
	Light grey	0 0 1 1	3	243
	Dark grey	0 0 0 1	1	241
	Black	0 0 0 0	0	240

0 = 'low'

1 = 'high'

X = 'low' or 'high'

\* on Spectrum; on the ZX81 add 65280; on the Ace, '31 IN 15 AND' gives the values shown in the 'Decimal Equivalent' column.

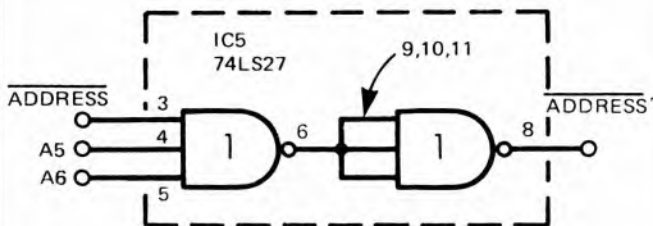
## Addressing

The addressing of this project is not completely provided for by the decoder (p. 158). The data inputs of the decoder are able to receive only one data line from each device (in other words, only one bit of data from each address). This project needs to be able to send 4 bits of data to the micro. As Fig.2.2 shows, it has four data outputs which are connected directly to lines D0 to D3 of the data bus. The wires from the Picture Digitiser go to 4 terminal pins on the board of the Decoder but, from there, the connection is made directly to the data bus of the micro, as shown in Fig.D.1.

Single-bit data inputs have complete address decoding on the decoder board; decoding of lines A0 to A4 is done by IC1 and IC2, while the decoding of lines A5 to A7 is done (for data inputs) by IC5. We can still use the decoding done by IC1 and IC2, but the Picture Digitiser needs to have its own IC to take care of decoding A5 to A7.

The decoding of lines A0 to A4 gives us a signal which is called ADDRESS. This goes 'low' whenever data is to be sent to any one of the addresses listed in Table D.2 (p. 173). If you want to have only the Picture Digitiser plugged on to the micro, and nothing else, there is no need to bother with any further decoding. Simply connect the ADDRESS output terminal of the decoder (Fig.D.1) to the ADDRESS output terminal of the Picture Digitiser (Fig.2.2). The Digitiser then has all 8 of the possible addresses of Table D.2. The micro reads the state of the pad, at *any one* of these addresses.

If you would rather be able to have several projects plugged into the micro at the same time, the Picture Digitizer needs its own address decoder to handle lines A5 to A7. Actually we adopt a compromise here and decode only two of these lines. This makes it possible to use just one IC for decoding. A suitable decoder circuit is shown in Fig. 2.3. Since it does not detect the state of line A7, which may be either 'high' or 'low', this decoder responds to *two* addresses, 1F and 9F. You can use either of these addresses for reading the Digitizer. Alternatively, use one of the decoder circuits shown in Fig.13.4.



To 0V: pin 7  
To +5V: pins 1,2,13,14

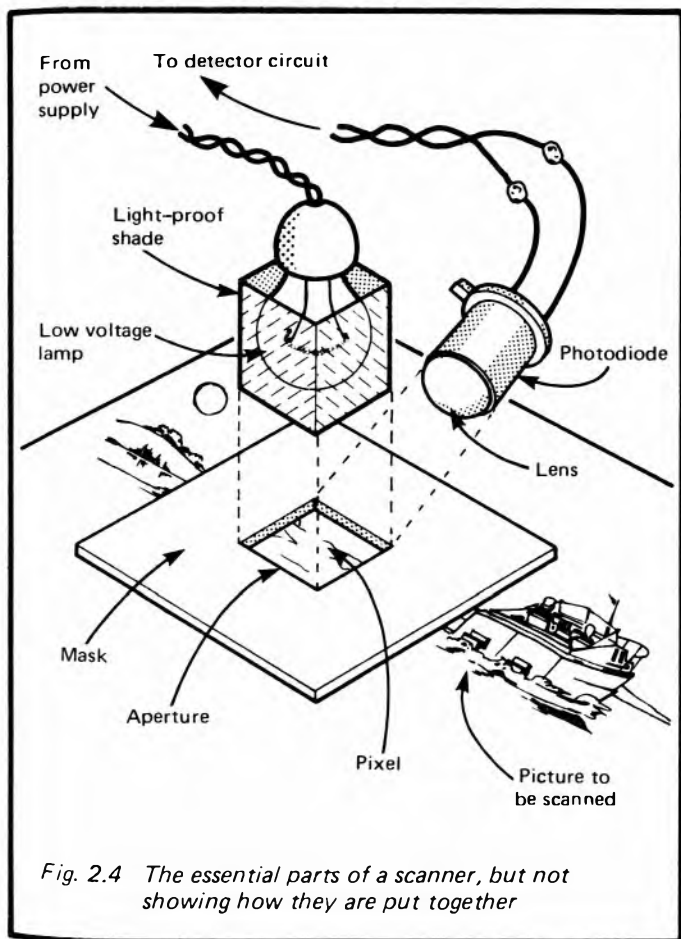
*Fig. 2.3 An address decoder for Project 2.  
This circuit gives it addresses 1F (decimal 31)  
and 9F (decimal 159)*

## Building it

The first thing to do is to make the scanner (Fig.2.4). There are several ways in which you can construct this, depending on the materials you have available and your skill at working with these materials. You may decide to base it on a small metal or plastic box, boring holes and fixing partitions to arrive at the design shown in the figure. Or you may decide to begin with a block of wood (or a large cork stopper, which is easier to carve) and drill out the various channels required. The essential points of the scanner are:

- i. It rests flat on the picture.
- ii. It has an aperture about 5mm square on the lower side;





*Fig. 2.4 The essential parts of a scanner, but not showing how they are put together*

- iii. this defines the area of the pixel.
- iii. A small lamp shines light on to the picture through this aperture.
- iv. It must be possible to raise or lower the lamp so as to adjust the amount of light reaching the picture. The

lamp should be a firm fit in its mounting, so that it does not readily slide out of its fixed position.

- v. The photodiode is to be fixed in position and aimed so that it catches light reflected from the picture, but does not receive light coming directly from the lamp.
- vi. It is best if all surfaces in the scanner are painted matt black to cut out unwanted reflections of light.

The prototype scanner was made from a large cork of the type used in the demijohns in home wine-making. Such a cork is cheaply obtainable from a store selling home wine-making equipment. Buy one which already has a central hole. Shape the hole at the lower end to make it square. The small filament lamp (on the end of flexible leads) fits tightly in the upper end of the hole and can be slid up and down. A slanting hole is cut from one side of the cork, sloping down toward the aperture. After soldering it to flexible leads, the photodiode is wrapped round with insulating tape (with tape between its wires too, to prevent short-circuiting) and wedged into the hole. Before putting the lamp and photodiode into their final positions, the insides of all holes are blackened using a felt-tip marker pen.

Black insulating tape is wrapped around the cork, except on the lower surface, to keep light from entering from the outside. Marks are made on the scanner to assist in registering its position during scanning (see later).

Before laying out the circuit board, decide whether you are going to include an extra IC for decoding lines A5 and A6. You will need to provide terminal pins on the board for the leads to the photodiode, and to the micro (7 leads: 0V, +5V, ADDRESS, and four data lines). You may also need to provide pins for the power supply to the lamp. The lamp needs to be a bright one which may take up to 0.3A of current. It is unlikely that your micro will be able to provide such a supply, especially if other devices are attached to it at the same time. Therefore the safest procedure is to power the lamp from dry cells. If your lamp is rated to run on 2.5V, use two 1.5V cells. A 3.5V lamp can run on 3 cells, and a 6V lamp on 4 cells. Adapt an old electric torch case to hold the cells, or buy one of the ready-made plastic battery-holders, which are inexpen-

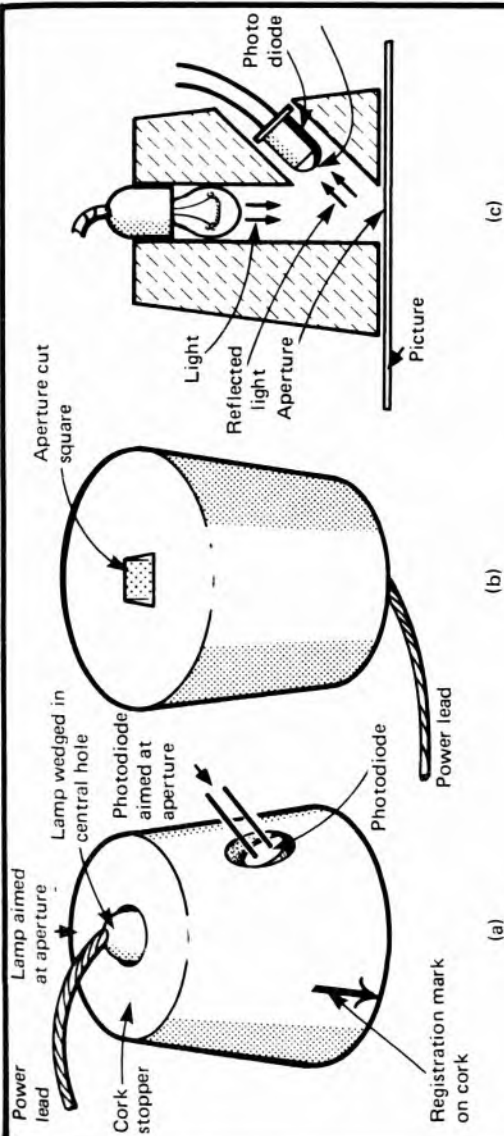


Fig. 2.5 An easy way of making a scanner from a cork stopper.  
 (a) in use, except that connections to the photodiode are not shown.  
 (b) upside down, to show aperture. (c) section

sive. The power supply to the battery is then completely separate from the main circuit of the Digitiser.

Wire up R1 and D1 first, and one of the comparators (say IC1 with RV1). Test the circuit with a voltmeter connected to point A. In fairly low room lighting the voltage at A is close to +5V but, when D1 is moved toward the lamp (switched on), the voltage drops almost to 0V. If you fail to obtain a changing voltage, is it possible that you have connected D1 the wrong way round. Set RV1 to about the middle of its track; the voltage at its wiper and at pin 2 of IC1 is then about 2.5V. Move D1 toward the lamp, the output of IC1 (pin 7) is close to +5V at first but suddenly swings to 0V when D1 is put closer to the lamp. Now wire up R2 and Q1. Check that the voltage at the collector of Q1 (i.e. where Q1 is linked to R2) changes from 0V to +5V as D1 is moved toward a lamp.

Repeat the above sequence of tests on the other two comparators as you assemble these sections of the circuit.

Next wire up the buffer (IC4). Temporarily connect the ADDRESS' input to the 0V line, to enable the buffers. You can then check that the outputs change as expected when D1 is moved toward the lamp (see Table 2.1). Adjust the settings of RV1, RV2 and RV3 so that the outputs change in order as D1 is moved toward and away from the lamp.

Finally, wire up the address decoder as in Fig.2.3, and connect its output to the ADDRESS' input of IC4.

## Final check

Before connecting the circuit to the micro it is important to test it thoroughly. Use a multimeter or a circuit-tester to check that there are no short-circuits between next-door data lines, or between data lines and the power lines. It is easy for a thin thread of solder to form a bridge between such lines. Also check that there is no short-circuit between the +5V power line and the 0V line. It is also worth testing each data line and other control line to make sure that there really is a connection between the plug at one end of the line and the IC or other component which is at the other. A 'dry joint' which causes a break in a line can cause all kinds of problems in

getting the circuit to work.

If it passes the tests above, the Digitiser should now be ready to plug into the Decoder, which itself is plugged into the micro. Switch on the power supply to the micro. If the display on the screen is not as expected, *switch off immediately* and repeat the checks.

## Test program

Below is a short program for the ZX-Spectrum which tests that the Digitiser is working properly.

```
10 FOR J=1 TO 100
20 PRINT IN 31 ;
30 PRINT " ";
40 NEXT J
50 PAUSE 200
60 CLS
70 GO TO 10
```

If you are using the ZX81, you will need to incorporate the machine code input routine into a REM statement on line 5, as described on p. 174. This routine is called by aUSR statement, so you will need to alter line 20 to LET X =USR(16514) and add another line, '25 PRINT X;'.

The interface may be tested on the Ace by using the word:

```
: TEST 100 0 DO 31 IN 7 AND . LOOP ;
```

This reads input, then ANDs it with 7 (=0111) to find what is present on the lower three data lines. The values obtained are those listed in the Decimal Equivalent column of Table 2.1.

The program reads the output from the Digitiser 100 times and displays the results on the screen. After a short pause, it clears the screen and takes the next set of 100 readings. Since the upper 4 data lines are unconnected, they read as 'highs', giving a total value of 240 (1111 0000) for those lines. On the ZX81, the value is 65520 (1111 1111 1111 0000) as explained on p. 9. The output from the Digitiser is added to this. Table 2.1 shows what values to expect. Place the scanner on white paper, and press the 'Send' button. The reading

should be '247' (65527 on the ZX81). Try it on black paper also and on various shades of grey. You can adjust the variable resistors slightly so that the readings change at the levels of grey which you decide on.

The only possible readings when the button is pressed are 240, 241, 243 and 247 (or their equivalent on the ZX81), depending on the brightness of the pixel. When the button is not pressed, you will get 248, 249, 251 and 255. If you obtain other readings, there is something wrong. Maybe the address decoder is not working properly. This is likely if you get the same result (especially '255') every time, no matter what the brightness of light or whether or not you press S1. You can check the address decoder by using the Pulse Detector (Project 1). Connect its input to the ADDRESS output of the Decoder, or to the ADDRESS' output of IC5 (Fig.2.3). Reset it, then run the test program. The LED should go off when the program is run. If it does not, check the decoding circuits.

Assuming the address is being decoded properly, but you are still getting unexpected numbers, write down the numbers you get in binary form, and also write down, in binary, the numbers you *expect* to get. By comparing these you may find that one of the data lines is 'stuck', always giving 'high' or 'low' when it should be changing. If so, examine the wiring of this line, looking for short circuits to next-door data lines or to the power lines. Look also for gaps and breaks in the line, dry solder joints and other possible bad connections.

The logic circuits are not 'tricky' in the sense that they need careful adjustment to get them to work. If you obtain your IC's *new* from well-known suppliers, they are very unlikely to be faulty, so if the circuit does not give the expected results, it is nearly always a fault in the construction.

## Programming

The program listed below shows how the Spectrum is used to read data from the scanner and plot it on the screen.

```
10 LET J=0 : BRIGHT 0 : CLS
20 LET scan=IN 31
```

```

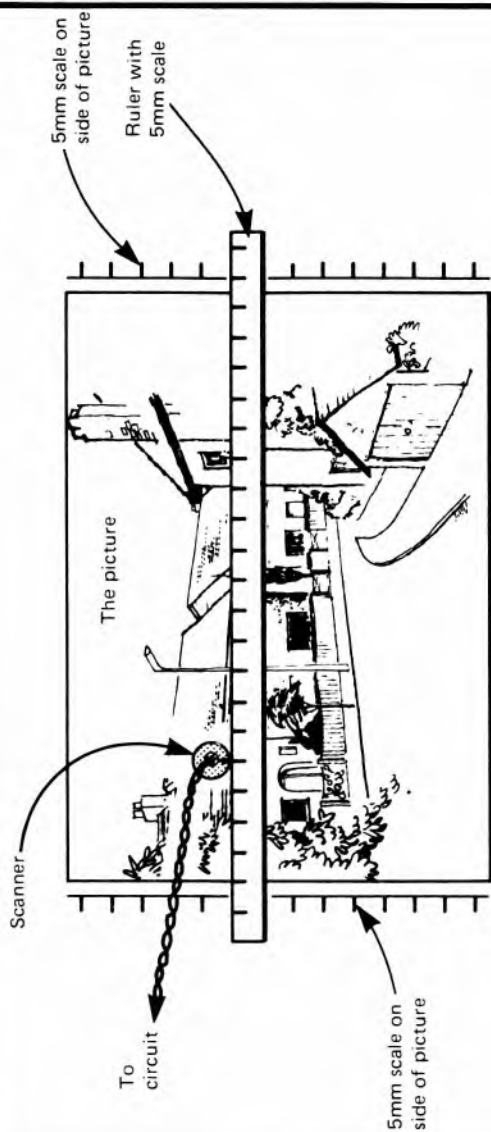
30 PAUSE 5
40 IF scan>247 THEN GO TO 20
50 IF scan=247 THEN BRIGHT 1 : LET pixel = 128
60 IF scan=243 THEN LET pixel=128
70 IF scan=241 THEN LET pixel=144
80 IF scan=240 THEN LET pixel=143
90 PRINT TAB J, CHR$(pixel),
100 LET J=J+1 : BRIGHT 0
110 IF J=704 THEN STOP
120 LET scan=IN 31
130 PAUSE 5
140 IF scan<248 THEN GO TO 120
150 GO TO 20

```

The scanner is moved across the picture as shown in Fig.2.6. First a ruler is laid across the picture, with its ends aligned with two scales drawn down the sides of the picture. The scanner is moved step by step from left to right along the ruler. The 'Send' button is pressed each time the scanner is put in a new position. At the end of the line, the ruler is moved down one step, and the scanner returned to the left-hand end. The next line is scanned.

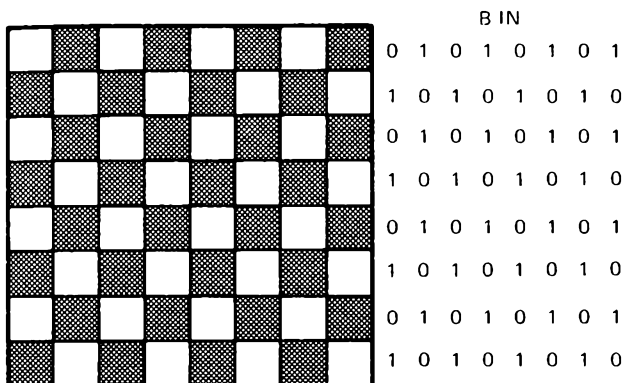
The program uses four character blocks to indicate the four levels of brightness. White is represented by a white block, with the screen brightness set at BRIGHT 1 (line 50). Otherwise screen brightness is normal. A white block at this level represents light grey. A black block (character 143), represents a black pixel. For dark grey we define a graphics block as described on p. 92 of the ZX Spectrum handbook. This must be done before you enter and run the display program below. When defining the character, poke USR"a" in line 20, so that the code number of the character is 144, as required for the display program. The character itself is a chequer-board of black and white dots (Fig.2.7), obtained by inputting the binary numbers 01010101 and 10101010 alternately when requested by the character defining program.

The 'dark grey' character is already available on key 'A' of the ZX81, but this micro does not have the BRIGHT command. This means that you can have only three shades –



*Fig. 2.6 Scanning a picture*





*Fig. 2.7 User-defined 'dark grey' graphics block for the ZX-Spectrum. Enter the binary codes listed on the right*

white, grey and black. The program for the ZX81 will include a line such as IF S=65523 OR S=65521 THEN LET P=8. Users of the ZX81 will note that the codes for the pixels are different (see p.181 of the ZX81 manual), as are the values read from the interface (see p. 27).

This program reads from the scanner and plots the results on the screen in 22 rows, each consisting of 32 pixels. Assuming the aperture of the scanner measures 5mm x 5mm, the maximum size of the picture scanned is 16cm wide and 11cm deep. If you want to scan a bigger picture, enlarge the aperture of the scanner.

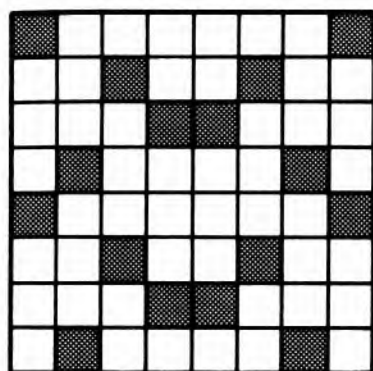
Here are some words for using the interface with the Ace. You need to define two characters to produce light grey and dark grey on the screen. Page 71 of the Ace Manual explains

how to do this. Fig.2.8 suggest two patterns. Define light grey as ASCII 1, and dark grey as ASCII 2. Next define WAIT, as explained on p.151 of the Ace manual. The other words needed are:

```
: SEND? BEGIN 31 IN DUP 8 AND WHILE  
  2000 WAIT REPEAT ;  
: NEXT? BEGIN 2000 WAIT DROP 31 IN 8  
  AND UNTIL ;  
: PIXEL I J AT DUP 4 AND IF 144 EMIT ELSE DUP 2  
  AND 1 EMIT ELSE DUP 1 AND IF 2 EMIT ELSE  
  SPACE THEN THEN THEN DROP ;  
: SCAN INVIS CLS 24 0 DO 32 0 DO SEND? PIXEL  
  NEXT? LOOP LOOP;
```

The word SEND? waits for the 'Send' button to be pressed. When the button is pressed, it leaves the input value on the top of the stack. PIXEL looks at the input value bit by bit (by successively ANDing it with 4, 2, and 1, and then prints one of the 4 possible pixel graphics. NEXT? waits until the button is released. SCAN repeats the operation of printing the pixel over the whole screen.

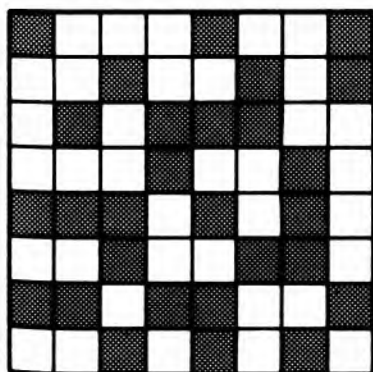
There are many other possible ways of using the output from the scanner. You can easily alter the program so that it displays a negative image of the picture. You can provide facilities for sending the completed display to the ZX Printer. Once the picture is stored in the micro's memory, it is possible for it to be processed in many different ways. In some applications it might be interesting for the computer to work out the proportion of light and dark areas, or the numbers of dark or white spots in the picture. The computer could scan its memory and locate the *edges* of each area of light and shade; then it could generate an 'edges-only' display, which would look rather more like a line drawing than a picture made of solid areas of various tones. On the Spectrum, you could write programs to create new images in colour based on the original picture.



```

1 0 0 0 0 0 0 1
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 1 0 0 0 0 1 0
1 0 0 0 0 0 0 1
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 1 0 0 0 0 1 0

```



```

1 0 0 0 1 0 0 1
0 0 1 0 0 1 0 1
0 1 0 1 1 1 0 0
0 0 0 1 0 0 1 0
1 1 1 0 1 0 1 0
0 0 1 0 0 1 1 0
1 1 0 1 1 0 0 1
0 0 1 0 1 0 1 0

```

Fig. 2.8 User-defined grey characters for the Ace. Enter the binary codes given on the right of each block

## *PARTS REQUIRED for the PICTURE DIGITISER*

*Resistors* (carbon, 0.25W, 5% tolerance)

R1                100k  
R2–R4          1k (3 off)

*Variable Resistors*

RV1–RV3    100k miniature horizontal preset (3 off)

*Semiconductors*

D1                BPX65 or similar photodiode  
Q1–Q3          ZTX300 or similar general-purpose npn  
                         transistors (3 off)

*Integrated Circuits*

IC1–IC3        311 comparator (3 off)  
IC4               74LS125 quad bus buffer gate with three-  
                         state output  
IC5               74LS27 triple 3-input NOR gate (optional, if  
                         required for decoding)

*Miscellaneous*

S1                Push-to-make push-button switch  
Circuit board  
8-pin IC sockets (3 off)  
14-pin IC sockets (1 or 2 off)  
10-way socket to fit 10-way plug of decoder board  
Lamp for scanner, low voltage (2.5V to 6V), 0.3A (the type  
which has a lens moulded into the end of the glass bulb  
is strongly recommended, as it focusses a bright spot of  
light on to the paper; available from most stores which  
sell electric torches.)  
Socket for lamp  
Materials for making scanner  
Connecting wire, including thin flexible wire for connec-  
tions to scanner

## Project 3

### FIVE-KEY PAD

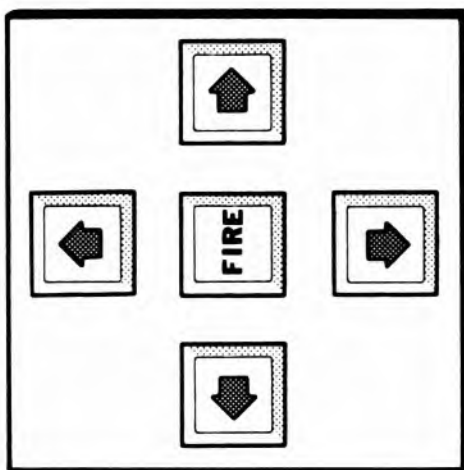
Although it is usual to control the movement of the cursor, or to aim and fire your space-gun against the aliens, by using the keys of the regular keyboard of the micro, it is very useful to have a separate keyboard specially designed for this purpose. This keyboard has 5 keys, which you may use in lots of different ways.

For instance, mark four of them with arrows pointing up, down, left and right, and use them for steering the cursor or anything else around the screen. The keys are arranged as in Fig.3.1 so that it is easy to remember which is which and to find the right key quickly. Another way of marking the four keys would be according to the point of the compass – N, S, E and W.

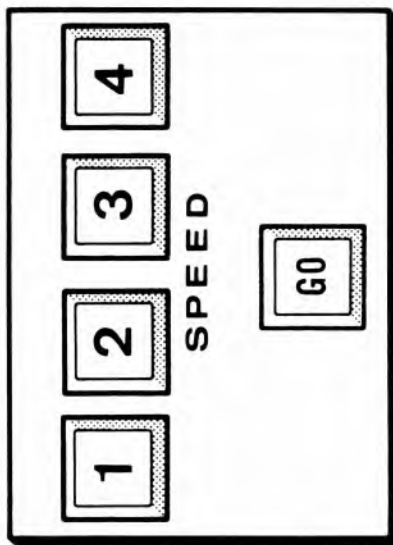
The fifth key is a control key which has several possible functions, depending on how you program the micro. In a games program you can use it to fire the space-gun, or perhaps to indicate "no move". In a program which draws pictures on the screen, under the control of the four "arrow" keys the fifth key can be used to indicate "pen down" (move the cursor to a different position and draw a line as it moves) or "pen up" (move the cursor to a different position, but do not draw a line).

The keyboard can also be used as a special-purpose controller for a model railway system or for slot-cars (see Project 4), with the keys being marked 'Go', 'Faster', 'Slower', and 'Reverse' (for locos at least!). The fifth key can be 'Emergency Stop'. A second keyboard can be used for controlling points or signal lamps.

Many two-player games are improved if each player has an individual keyboard. They are especially useful if it is the sort of game in which each player has to 'make a move' without letting the other player know which move is being made. Also, the computer can be programmed to read one keyboard or the other, depending on whose turn it is to play. Playing out of



(a)



(b)

Fig. 3.1 Two possible layouts for a 5-key pad.  
 (a) direction arrows for games control, with firing key. (b) slot car speed controller;  
 press "go" and one of the numbered speed buttons and then release "go" key.

turn is prevented.

The keyboard can be used as a remote-operating keyboard. You place the micro in one room and the keyboard in an adjacent room, sending instructions to the micro along a 6-wire cable.

## How it works

Each key is connected to one input of an exclusive-OR gate (Fig.3.2). An exclusive -OR gate has two inputs. When both inputs are alike (both 'high' or both 'low') the output of the gate is 'low'. The output is 'high' only when the inputs differ. The IC used here belongs to the TTL family. The inputs of this family have the property that if they are disconnected they count as a 'high' input. Thus, when a button is not pressed the corresponding input acts as if it were 'high'. Pressing any button makes its input 'low'. The effect of pressing various combinations of buttons are as shown in Table 3.1.

*Table 3.1 Data output from the Five-key Pad*

Keys pressed 5 4 3 2 1	Data lines 3 2 1 0	Decimal equivalent	Value displayed*
- - - - -	0 0 0 0	0	240
- - - - +	0 0 0 1	1	241
- - - + -	0 0 1 0	2	242
- - + - -	0 1 0 0	4	244
- + - - -	1 0 0 0	8	248
+ - - - -	1 1 1 1	15	255
+ - - - +	1 1 1 0	14	254
+ - - + -	1 1 0 1	13	253
+ - + - -	1 0 1 1	11	251
+ + - - -	0 1 1 1	7	247

- = key not pressed

0 = 'low'

+ = key pressed

1 = 'high'

\* on the Spectrum; on the ZX81 add 65280; on the Ace, '31 N 15 AND.' gives the values shown in the 'Decimal Equivalent' column

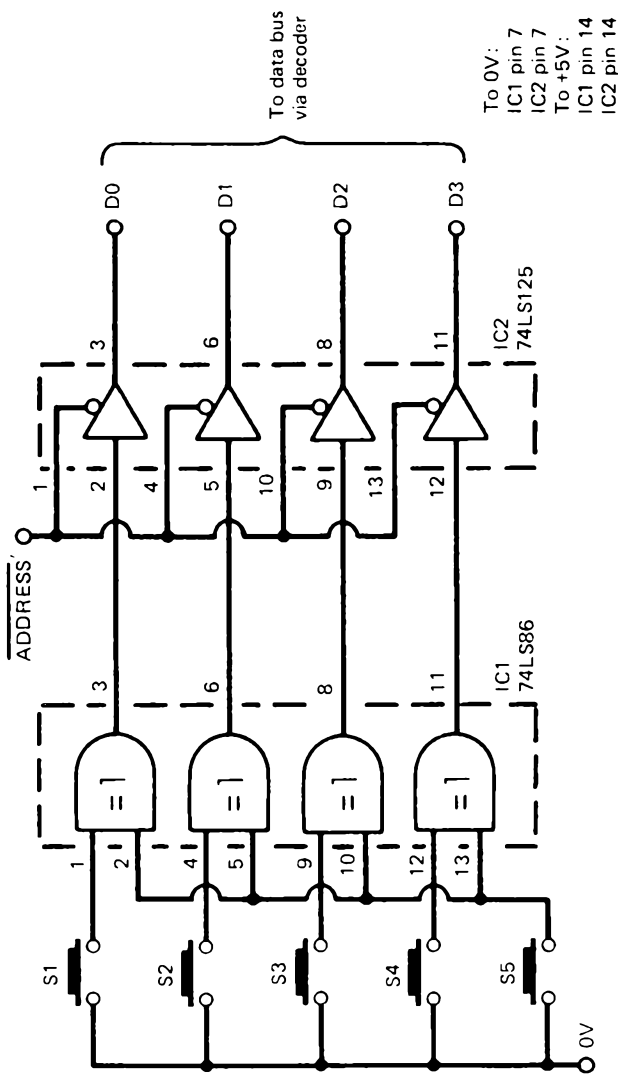


Fig. 3.2 Circuit diagram of the 5-key pad. S5 is the 'control' or 'fire' key



The output of each exclusive-OR gate goes to a buffer with three-state outputs (see p. 160). The outputs of these buffers are connected to data lines D0 to D3. The buffers are made to put data on the bus by bringing the enable line low. This is connected to an Addressed Output of the decoder (see Fig.D.1). When one of the addresses of the decoder is read from, the corresponding Addressed Output goes low. This enables the buffer, and the data from the buffers is put on to the data bus.

## Addressing

As explained for Project 2 (p. 21), the addressing of this project is not completely provided for by the decoder (p. 158). If you want to connect only this project to the micro, you can do as suggested on p. 21: wire the  $\overline{\text{ADDRESS}}$  output terminal of the decoder (Fig. D.1) to the  $\overline{\text{ADDRESS'}}$  input terminal of the 5-Key-Pad (Fig. 3.2). It can then be addressed using any one of the addresses in Table D.2.

If you would like to have other projects connected at the same time as this project, you need to add a decoder to this project. To keep the wiring as simple as possible, the decoder uses only lines A5 and A6 (Fig. 3.3). As explained on p. 21 it responds to two addresses, which are 3F and BF in this instance.

If you are building a second pad, it must have a different address. Use the same decoder circuit as in Project 2 (Fig. 2.3), which has addresses 1F and 9F. If you do this, you will not be able to have the Picture Digitiser and the second 5-Key Pad plugged on the micro at the same time. Other decoding circuits you could use are given in Fig. 13.4 (p. 120).

## Building it

Although it is possible to use any kind of switch for the keys, it is best to employ the type specially designed as keyboard keys. Usually the switch is sold separately from the key-top. Keytops fit on to the switch and are of various types. Some makes of key have tops in a range of bright colours. Others

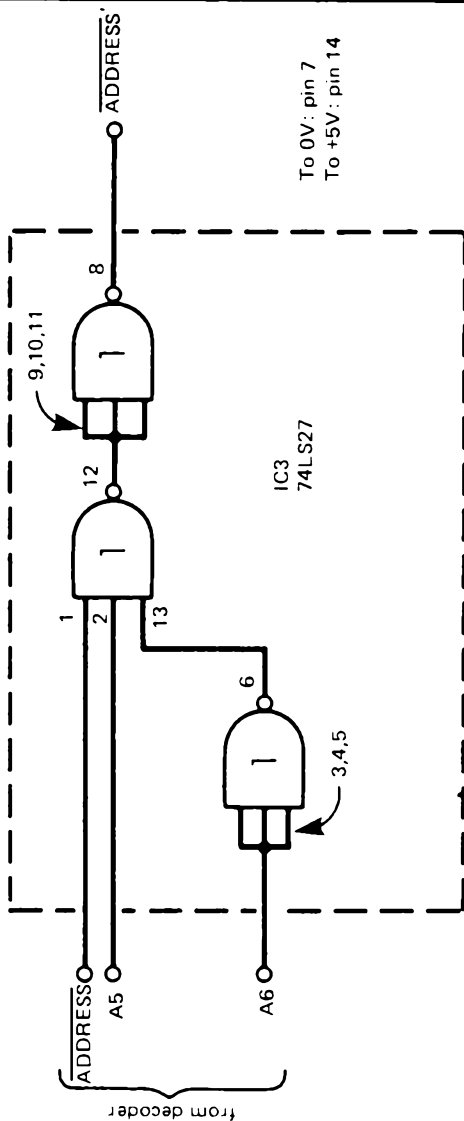


Fig. 3.3 An address decoder for Project 3. This circuit gives it addresses 3F (63 in decimal) and BF (191 in decimal)

have white tops with transparent covers. You write a letter or symbol on the top (or use rub-down lettering) then clip the transparent cover over the top to protect the lettering from wear. Alternatively, draw the symbol or letter on a small square of card and insert this before clipping on the cover.

The project is best mounted in a small plastic case. Cases can be obtained with a sloping top which makes them very suitable for use as a keyboard enclosure. Alternatively, a plastic box used for packing foods and other commodities is generally obtainable free and may be used for housing this project.

The layout and construction of the circuit presents no problems. The two ICs and the decoder IC can easily be accommodated on a small piece of strip-board which will fit into the case. Cut an aperture in the lid for the keys. There are nine leads between the pad and the decoder — 3 for addressing, 4 for data and 2 for power. These can be a metre or so in length to allow you to operate the pad in any convenient position. The leads end in a 10-way plug which fits the 10-way plug of the Decoder board.

## Testing

When construction is complete, test the circuit to make sure that there are no short-circuits between next-door data lines or address lines, or between these lines and the power lines. Also make sure that there is no short-circuit between the two power lines, 0V and +5V. Connect the circuit to a 5V power supply and temporarily connect the address inputs (ADDRESS and D5/D6 if used) to the 0V line. Then press each button in turn, while measuring the output from the corresponding buffer with a voltmeter. Table 3.1 shows what to expect.

The pad may now be plugged on to the decoder which is plugged in to the micro. Switch on the micro. If the usual display fails to appear on the screen, *switch off immediately* and repeat your checking of the circuit. Now run a simple test program. A program suitable for the Spectrum is given on p. 27, but change the address on line 20 to 63. Below is a similar program for use with the ZX81.

First, key in and RUN the input program listed on p. 174. Then delete lines 20 onward and replace them by:

```
20 FOR J=1 TO 100
30 LET X=USR 16514
40 PRINT X;
50 PRINT " ";
60 PAUSE 200
70 CLS
80 GO TO 20
```

This program displays values 65520 to 65535 depending on which key is pressed (see Table 3.1). The right hand column of Table 3.1 shows what numbers should appear on the screen. If you fail to get the expected values, read p. 28 to find out how to find the fault.

When the pad has been checked and found to be in working order, the next step is to make use of it on one of your own programs. Exactly how you do this depends on what sort of program it is. The principle is simple: read the pad, and then branch to different parts of the program depending on the value that is obtained. The program of p. 28 illustrates two points about reading from keys. A micro works fast, and it can read a key hundreds of times while it is being pressed just once! The program on p. 28 is controlled by pressing the 'Send' key of the Picture Digitiser. In lines 20 to 40 it waits in a loop until it detects that the key has been pressed. The pause in the loop is intended to overcome the contact-bounce of the keys. When a key is pressed it seldom changes straight from off to on, but is more likely to go on-off-on-off-on-off and so on several times until it finally settles at on. The pause allows it time to settle to fixed state before taking the next reading. The main part of the program next deals with the reading so obtained. Then, before taking the next reading, the program must check that the key has been released. It waits in another loop (lines 120 to 140) until it detects that the key is no longer being pressed. Then it jumps back to the beginning of the program to wait for the key to be pressed the next time.

You may need to use routines of this sort in your key-pad program if you want to interpret a series of key-presses as a

series of commands. Of course, if the keys are simply to be pressed and held down for as long as a particular action is to continue, you simply read the pad repeatedly, without pauses, until a change of input occurs.

### *PARTS REQUIRED for the FIVE-KEY PAD*

#### *Integrated Circuits*

- IC1      74LS86 quad exclusive-OR gate
- IC2      74LS125 quad bus buffer gate with three-state output
- IC3      74LS27 triple 3-input NOR gate (optional, if lines A5 and A6 are to be decoded)

#### *Miscellaneous*

- S1–S5   Key-switches with tops (5 off)
- 14-pin IC sockets (2 or 3 off)
- 10-way socket to fit 10-way plug of decoder board
- Key-pad case
- Connecting wire

## Project 4

### MODEL CONTROLLER

Putting a micro in command of model railway system adds greatly to the realism of its operation. The locomotive can be stopped, started, or reversed and its speed can be varied in a number of stages. The project can also be used with other electrically powered models such as slot-cars. If you are keen on building your own robot, this project provides a way to control its actions. In fact, any model which is driven by low-voltage DC motors, or is activated by electromagnets can be controlled through this interface. On the model railway scene, it may be used for switching points. The interface can also switch lamps, so is ideal for signal-switching.

Many of the circuits used for controlling the speed of a motor require that the power source should have a higher output voltage than is actually needed for driving the motor at top speed. This circuit employs relays to do the switching, so it is able to take power from the transformer or power pack that you normally use for your model railways or slot-cars. There is no need to obtain or build a special power supply. The relays do not cause loss of voltage, so the motors run at top speed when you want them to do so.

The project as described here is suitable for use with a model railway but you will find it easy to adapt it to slot-cars or other models. One point about the circuit is that you do not need to build it all at once. You can start with just one relay and expand it with further relays from time to time. As your model railway system grows, you can build a second or even a third version of the project, perhaps using one for controlling the locomotive and the other for controlling points and signal lamps.

### How it works

Fig.4.1 shows the relay-control side of the project. IC1 contains four latches. The D (for data) inputs of these are con-

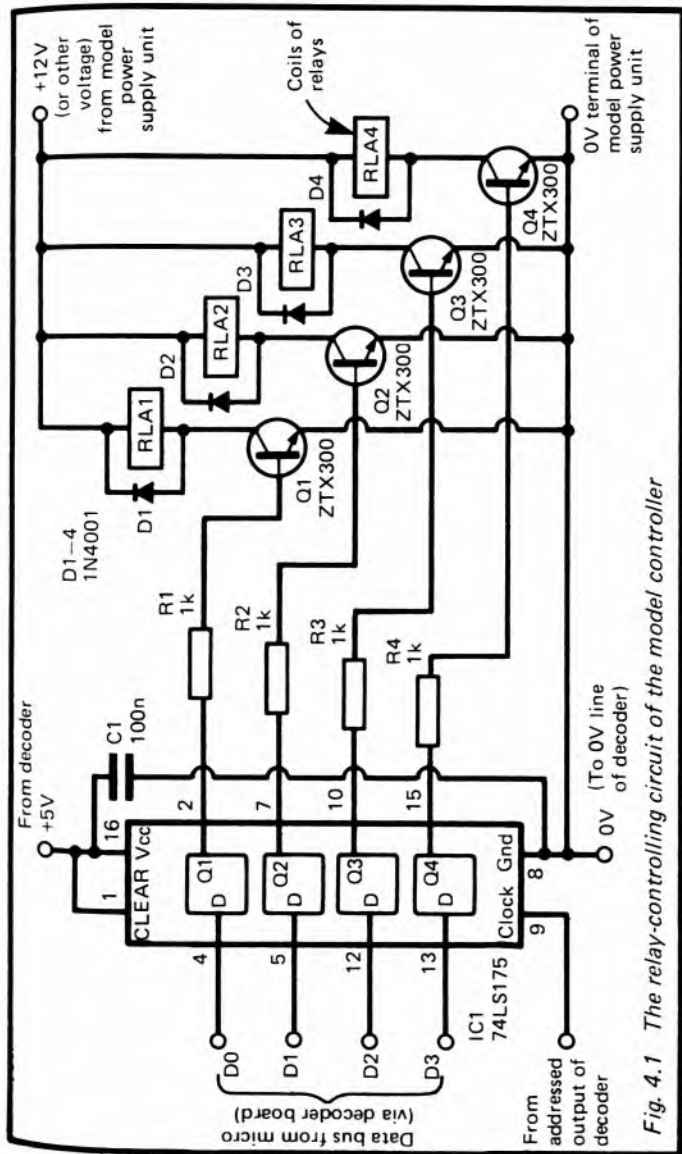


Fig. 4.1 The relay-controlling circuit of the model controller

nected to 4 lines of the data bus. The IC also has a 'clock' input (pin 9). This is connected to one of the Addressed Outputs of the decoder (Fig.D.1). When the micro wants to send data to the Controller, it puts data on the data bus and the Controller's address on the address bus. The address is decoded completely by the decoder, with the result that the corresponding Addressed Output goes low for an instant. This low pulse triggers the latches of IC1. The output (Q) of each latch becomes the same as its data input (D). Once the triggering pulse is over, Q does not change even though D changes. In fact, D changes immediately the trigger pulse is over, for the micro is busily sending data to its memory, to the printer or other devices, or is receiving data. But the data which was on the bus at the instant the triggering pulse occurred is held unchanged (latched) until the next time that the micro sends data to the Controller.

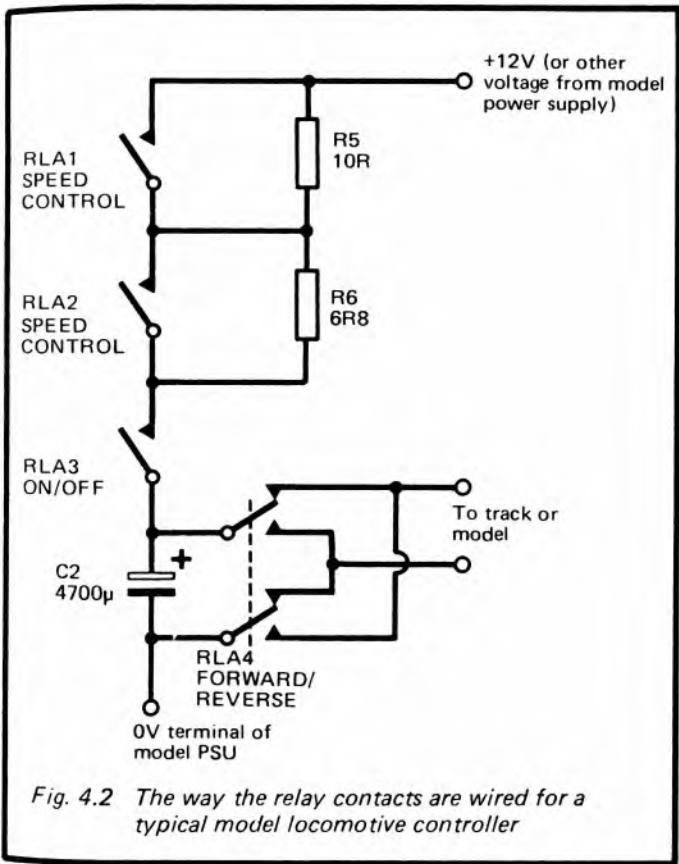
The output of each latch supplies base current to a transistor. If Q is 'low', there is no base current. If Q is 'high', base current flows and the transistor is turned on. This causes a collector current to flow, which activates the coil of a relay. What happens then depends on how the relay is wired.

Fig.4.2 shows a typical relay circuit for controlling a model locomotive. Note that the power supply for the relay coils and for the locomotive all come from the power supply unit which belongs to the model railway. This *must* be a DC supply and you must not use the circuit with voltages higher than 25V, for this would damage the transistors. The normal voltage for models is 12V or less, so this circuit is very likely to be suitable for all your models.

Relay 3 (RLA3) is a simple on-off switch. It is wired so that the circuit is closed (the loco moves) when the output of the latch is 'high' and the transistor is turned on. The current to the track may pass through two resistors (R5,R6), but either or both of these may be short-circuited by closing relays 1 and 2. These are wired so that they are open when the output of their latches is 'low'.

When relays 1 and 2 are both open, the current to the track has to pass through R5 and R6, putting a resistance of 16.8ohms in series with the motor of the locomotive. This





*Fig. 4.2 The way the relay contacts are wired for a typical model locomotive controller*











gives a slow starting speed. Incidentally, the values of resistors shown in Fig.4.2 are those which were used with a particular model locomotive when this circuit was being designed and tested. You will probably find that you need quite different values in your circuit, though those given should be good as a starting point for your own trials. When one of relays 1 and 2 is closed, the resistance in series with the motor becomes 10 or 6.8ohms. These reduced resistances give two higher speeds.

Finally, both relays may be closed, reducing the resistance to zero and putting the locomotive into top speed.

Relay 4 is a reversing relay. The relay has two pairs of contacts which are changed over simultaneously. As a result, the current flows one way or the other around the circuit and the loco runs forward or in reverse.

Table 4.1 shows the outputs required from the latches in order to produce differing speeds and direction.

*Table 4.1 Command values for the Controller*

Direction of motion	Speed	Latch outputs				Decimal equivalent	Symbol in REM (ZX81)
		Q4	Q3	Q2	Q1		
Forward	STOP	0	X	X	0	0, 2, 4, 6	space,  ,  , 
	Slow	0	0	0	1	1	
	Medium	0	0	1	1	3	
	Fast	0	1	0	1	5	
	Top	0	1	1	1	7	
Reverse	STOP	1	X	X	0	8,10,12,14	 ,  , £, :
	Slow	1	0	0	1	9	
	Medium	1	0	1	1	11	"
	Fast	1	1	0	1	13	\$
	Top	1	1	1	1	15	?

0 = 'low'

1 = 'high'

X = 'low' or 'high'

## Building it

Since all decoding is done by the decoder, there is no need for any address-decoding ICs on the board. The single IC takes up very little room, but allow plenty of space for the relays and for R5 and R6. C2 also takes up a lot of room. It is essential to instal this capacitor, for a locomotive produces sparks as it crosses the joins between one rail and the next, or if it encounters a part of the track where the rails are rough or slightly corroded. These sparks generate voltage 'spikes' which can be transmitted through the circuit and reach the micro. They are not likely to do any harm to the micro but may cause some of the memory locations to change state. This leads to rather odd error messages appearing which seem to

bear no relation to the program. If interference is worse the program may "crash". This capacitor completely eliminated all such interference in the prototype circuit. C2 should be wired as close as possible to the two terminals which supply power to the track. If a reversing relay is fitted, C2 must be as close as possible to this, on the side nearer the other relays (see Fig.4.2).

There are similar reasons for C1. Latches have the habit of becoming 'unlatched' if there is interference around. They are sensitive even to the voltage surges caused by switching the relays on and off. Position C1 so that its wires are soldered as close as possible to the power terminals of IC1 (pins 8 and 16).

The final protective feature is the diodes. When relays switch off, a large reverse current is generated. This could eventually damage the transistors. The diodes conduct this current safely away to the +5V line. The diodes should be wired as close as possible to the terminals of the coils of the relays.

The controller is connected to the decoder board at 2 points. One connection is to the data lines; this cable requires 4 wires and may be plugged on to one of the 10-way plugs. The other connection is to one of the 3-way plugs with an Addressed Output; this needs 3 wires, one for the Addressed Output, the others being the 0V and +5V lines.

As mentioned earlier, there is no need to build the whole circuit at once. If you want to try it out on a small scale first, wire up just IC1, R1, Q1, D1, and RLA1. You also need to include C1 and C2. This will allow you to experiment with controlling one relay which you might use, for example, simply to start and stop the locomotive. Speed control and reversing can be added later. At that stage you may decide to use only one relay for speed control (giving just 3 speeds — slow, medium, top-speed), so freeing a relay for use in switching points or operating signal lamps. The main point is that, if you start with a small system, build it on a board large enough to accommodate future expansion.

As mentioned above, the most suitable values for R5 and R6 depend on the characteristics of the motor of the locomotive or other model. The best course is to try connecting

the resistors in series with the locomotive *before* wiring them permanently into the circuit. The amount of current taken by the motor may be 1 ampere or more, so it is essential to use resistors rated at 2.5W at least. Fixed-value wire-wound resistors are available cheaply. It is possible to obtain variable wire-wound resistors (3 watts), but these are relatively expensive so the best course is to buy a selection of fixed-value resistors and experiment with these. If you are following the two-relay/two-resistor scheme of Fig.4.2, first find out what resistance is enough to let the locomotive run at its lowest steady speed. The speed must be such that the motor does not stall when the loco crosses a gap in the track or when running round sharp bends. Also it must allow a stationary locomotive to begin moving while pulling the heaviest train it is likely to have to pull. Having established the correct value (which is the total value of R5 and R6), divide this value into two parts, one rather larger than the other. In Fig.4.2, the total value is 16R8 ohms, broken into 10R and 6R8 ohms. These values are, of course, the nearest standard values obtainable. If you need a value which is non-standard, it is often possible to join two or more resistors in series; for example, you can make up a resistance of 7R5 by joining 1R, 1R, 2R2 and 3R3 ohms in series. Often an easier method is to put two larger resistors in parallel. In this instance, wire two 15R resistors in parallel to obtain 7R5 ohms. When wired in parallel, and providing the resistors are more-or-less equal in value, they *share* the current. Resistors of lower rating may be used, such as 1W or 2W carbon resistors.

Before connecting the controller to the micro, test it to make sure that there are no short-circuits between adjacent data lines, between the data lines and the power lines, and between the two power lines. The model's +12V (or other) power line does *not* connect directly to any line going to and from the micro, but its 0V line *must* be connected to 0V line of the project. To test the operation of the circuit, connect the controller to the track, and to the power lines of the decoder (plugged into the micro), but do not connect the data lines or the Addressed Output line yet. Use leads with crocodile clips to connect the data line terminals and Addressed Output

terminal of the controller to the 0V line.

Switch on the power supply to the controller and the railway power supply unit. Connect the lead of terminal D0 to the +5V line. Then connect the Addressed Output lead (i.e. the one which connects to the 'clock' terminal of IC1) to the +5V line. As soon as it makes contact with the +5V line, the output of latch 1 changes from 'low' to 'high', because of the 'high' level on its input. Often it changes *before* you actually touch the lead against the +5V line, for the slight rise in voltage caused by taking it away from the 0V line is enough to trigger the latches. Putting the lead back on to the 0V line has no further effect. As the output of latch 1 goes 'high', the locomotive should start moving slowly. Now put the D0 lead back to the 0V line, touch the Addressed Output ('clock') lead to the +5V line and the locomotive stops. This is because the output from latch 1 has now returned to 'low'. In a similar way check the action of the other latches and relays.

## Programming

Programming the controller to perform a required action is extremely easy. Just use the OUT command on the ZX-Spectrum or Ace, or the equivalent machine-code subroutine on the ZX81 (p. 174), with one of the values of Table 4.1.

There is one point to be considered when starting the locomotive (or other motor) from rest. Some model power supply units have an automatic cut-out which shuts down the supply if it becomes overloaded. If you try to accelerate the locomotive too rapidly by starting it off at top speed (Relays 1, 2 and 3 on together), the sudden surge of current may trip the cut-out. You will then have to reset the power unit by hand. If your unit is of this type, always start off the locomotive at its slowest speed. Once it has started moving, it can be put into higher speeds almost immediately without risk of triggering the cut-out.

On the other hand, certain types of motor need an initial burst of power to get them running, after which they can be run at relatively low speed. Slot-cars are often like this, requiring a quick 'kick' on the control lever to start them. If

your model is of this kind, you may find that the best procedure for starting it from rest is to begin with relays 2 and 3 closed, so that maximum power is delivered. Follow this *immediately* with a command to open both relays. The micro is programmed to deliver these two commands in very quick succession, so that the initial high acceleration is over before you have had time to notice it. The car apparently accelerates smoothly away from its starting point.

With railway systems and models of some other kinds, especially robots, it is possible (though quite a challenge!) to write a program by which you can control the model from the keyboard of the micro by pressing certain keys. While this is happening, the computer is storing in its memory a list of all the commands you have issued and the length of time for which each command was in effect. When the sequence is over, the micro repeats all the commands with the same timing, so the model repeats the whole sequence automatically. This is the way in which an industrial robot is 'taught' to perform a complicated sequence of actions by an experienced instructor.

One way in which the controller may be helped to do its job is to let the micro know what the model has actually done. With a railway system, for example, it is helpful if the micro can be told exactly which part of the track the train has reached, when it is approaching a station, or when the end of the train has moved fully into the siding. One way of locating the train is to arrange beams of light across the track to be broken by the train as it passes. The Lap Sensor described in Project 10 is ideal for this purpose. One or more of these placed at strategic positions on the railway system will make practicable many kinds of automatic manoeuvre. A sophisticated program would allow the operator to type in the names of the departure and destination stations, whereupon the micro would work out the route, set the points and signal lamps and drive the train from the one station to the other, without any further guidance for the operation.

## *PARTS REQUIRED for the MODEL CONTROLLER*

### *Resistors*

R1–R4	1k carbon, 0.25W, 5% tolerance
R5, R6	Wire-wound resistors, 2.5W, of suitable values (see p. 50)

### *Capacitors*

C1	100n polyester
C2	4700 $\mu$ electrolytic, working voltage greater than that of the model power supply unit

### *Semiconductors*

D1–D4	1N4001 (4 off)
Q1–Q4	ZTX300 or similar npn transistors

### *Integrated Circuit*

IC1	74LS175 quad D-type flip-flop
-----	-------------------------------

### *Miscellaneous*

RLA1–RLA3	Miniature relays; coils rated to voltage of model power supply unit; single-pole single-throw contacts or change-over contacts; contacts rated to voltage of power supply to 2A minimum (3 off)
RLA4	Miniature relay, specification as above, except that contacts are to be double-pole change-over

### *Circuit board*

14-pin IC socket

10-way socket to fit 10-way plug of decoder board (a 4-way or 5-way plug may do, as only the 4 data lines are needed)

3-way socket to fit 3-way plug of decoder board

Plug to fit outlet of model power supply unit

Connecting wire

## Project 5

### BLEEPER

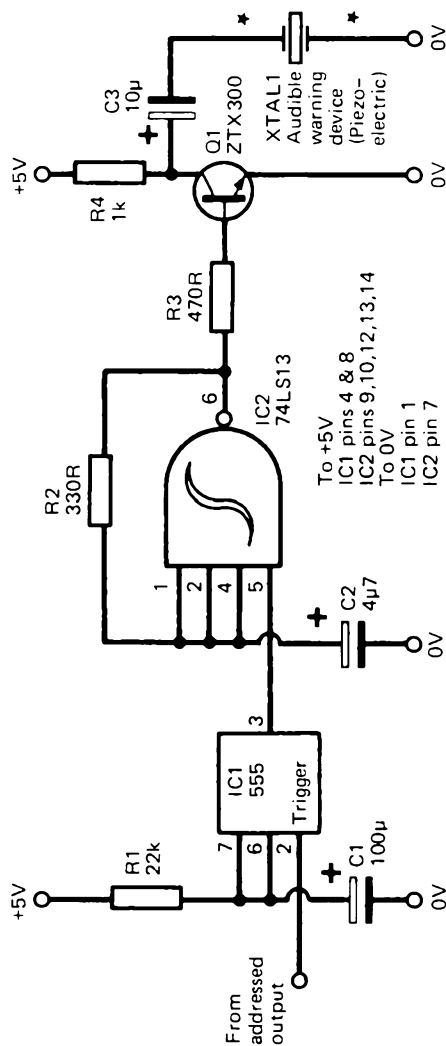
A device which emits a short 'bleep' when triggered by the micro has many uses in connection with games programs and in various applications around the home. Although the Spectrum and Ace already have a built-in loudspeaker which can be programmed to 'bleep', the loudspeaker is firmly fixed inside the micro and can not be placed elsewhere. With this project, the micro can be in one room and the bleeper in another. For example, with the bleeper in the kitchen, you could use the micro as an egg-timer, 'bleeping' at the ends of the cooking times which each member of the family prefers. If you have the ZX81, then the bleeper is a valuable addition which will greatly enhance many of your programs.

#### How it works

The note emitted by the bleeper is produced by an oscillator built from a single NAND gate (IC2, Fig.5.1). This gate has Schmitt trigger inputs, which means that the output of the gate swings sharply when the input voltage reaches a certain value. The inputs of the gate are wired together (except for one, which we shall discuss later), so it acts as an *inverter*. When the inputs are 'high' the output is 'low', and when the inputs are 'low' the output is 'high'. Suppose the input is low to begin with, and the output is high. Current flows through R2 and gradually charges the capacitor C2. When the voltage across C2 reaches a certain level, this counts as a 'high' input. The output swings low immediately. Now current flows from C2 toward the output, which is at 0V. The capacitor gradually discharges and the voltage across it falls. When it falls below a certain level it counts as a 'low' input and so the output swings 'high' again. In this way the output swings 'low' and 'high' continuously.

The rate at which the circuit oscillates depends on the values of C2 and R2. For the circuit to work, the value of R2





\* = where the leads may be extended when mounting the AWD remotely

Fig. 5.1 Circuit diagram of the bleeper

must lie between 330 ohms and 470 ohms, but we can alter the value of C2 over a reasonably wide range to give a note of the chosen pitch. With the values shown in Fig.5.1, the pitch is about 500Hz.

The duration of the 'bleep' is controlled by a 555 timer IC, (IC1). This is wired so as to produce a single 'high' pulse (at its output, pin 3), whenever a short 'low' pulse is delivered to its trigger input (pin 2). The input pulse may be very short indeed and in this circuit we use the pulse from an Addressed Output of the decoder (Fig.D.1). The length of the output pulse can be anything we choose, within reason. Its length depends on the values of C1 and R1. The equation for calculating the duration is:

$$t = 1.1RC$$

where t is the time in seconds, R is the resistance of R1 in ohms and C is the capacitance of C1 in farads. With the values given in Fig.5.1, the duration is about 2.4 seconds.

The output from the timer is normally 'low'. One input of the NAND gate is held 'low', so the gate is prevented from changing state. It can not oscillate and no sound is heard. When the timer is triggered, its output goes high for 2.4 seconds, during which time the NAND gate is able to oscillate and the 'bleep' is heard.

The sound is made by a piezo-electric audible warning device. This is a thin slice of crystalline material which vibrates when a pulsing signal is passed through it. It is rather like a crystal microphone or record-player cartridge, but working in reverse. The output from the oscillator is insufficient to drive the crystal directly, so we use a transistor (Q1) which is switched on and off by the output from the oscillator. This provides enough power to make a suitably loud noise come from the crystal. If you prefer, a small loudspeaker may be wired in place of the crystal.

## Building it

The project may be housed in any plastic case big enough to hold the scrap of circuit board on which it is assembled. It

needs only three wires from the decoder: 0V, +5V and one of the address outputs. If you intend to use the bleeper in another room at some distance from the micro, it is best for the main part of the circuit to be in its case close to the micro with the crystal (or loudspeaker) on the end of a long pair of wires leading to the other room.

One thing to think about before beginning construction is the mounting of the audible warning device. The volume of sound obtained is much greater if it is mounted on a firm (but not too firm) surface. The surface acts as a sounding-board, helping to transfer the energy from the crystal to the air around. Most crystals are already in a light metal case with metal lugs attached (Fig.5.2). These lugs are pushed through holes bored in the plastic case containing the circuit board (or a separate case, if the crystal is to be located in another room). Then the lugs may be bent flat to hold the device firmly against the wall of the case. If you are using a loudspeaker instead of a crystal, mount it on the inside of the wall of the case, with a few holes bored in the case to allow the sound to escape. An old transistor radio set could be adapted for this purpose, provided that its loudspeaker is in working order. There is no need to remove the 'works' for there is sure to be enough room to spare for the Bleeper board. Disconnect the loudspeaker from the radio circuit and connect it to the bleeper instead.

Apart from the points mentioned above, building the circuit is very straightforward. The device needs no address decoding other than that already done by the decoder. Simply run a 3-way cable from one of the Addressed Output plugs of the board.

The circuit may be tested by connecting it to a +5V or +6V supply. Usually it 'bleeps' as soon as it is switched on. Temporarily connecting pin 3 of IC1 to the 0V line will make it 'bleep' again. If it does not, check the wiring carefully. If the pitch of the note is not as required, substitute a different capacitor for C2. The larger its value the lower the pitch. If the length of the 'bleep' is not correct, alter C1 or R1. Changing either or both of these to greater values, lengthens the 'bleep'. It is possible to have the note sounding for several tens of minutes.

Hole for leads

Metal lug

AWD

(a)

(b)

Fig. 5.2 Mounting the audible warning device on the case.

(a) showing holes required, (b) lugs inserted in holes and bent around the panel

## Programming

This could not be simpler. With the Spectrum, the command OUT 31,0 causes the device to 'bleep'. This supposes you are using the first Addressed Output, if you are using another one, use its address in place of the 31. The 0 is just a dummy value, for we are not actually sending any data to the bleeper. You could use any value in the range 0 to 255.

Programming is a little more complicated with the ZX81, for you first have to put the machine-code output program into memory, as explained on p. 174. Having done this, use the command LET X =USR 16514. If you have other devices connected at the same time, you will have to precede this by POKE 16517, 31, or whatever address you have chosen for the bleeper.

With the Ace, a word to trigger the bleeper is defined like this:

```
: BLEEP 0 31 OUT ;
```

The word BLEEP will then trigger the bleeper into action. As above, you should substitute the address you actually use for the '31' in the definition.

### *PARTS REQUIRED for the BLEEPER*

*Resistors* (carbon, 0.25W, 5% tolerance)

R1	22k
R2	330R
R3	470R
R4	1k

*Capacitors* (electrolytic)

C1	100 $\mu$
C2	4 $\mu$ 7
C3	10 $\mu$

*Semiconductor*

Q1	ZTX300 or similar npn transistor
----	----------------------------------

*Integrated Circuits*

IC1	555 timer
IC2	74LS13 dual 4-input Schmitt trigger NAND gate

### *Miscellaneous*

XTAL1 Piezo-electric audible warning device, PCB mounting (loudspeaker, 3 ohm to 15 ohm may be substituted)

Circuit board

8-pin IC socket

14-pin IC socket

1mm terminal pins (5 off)

3-way socket to fit 3-way plug of decoder board

Connecting wire

## Project 6

### LAMP FLASHER

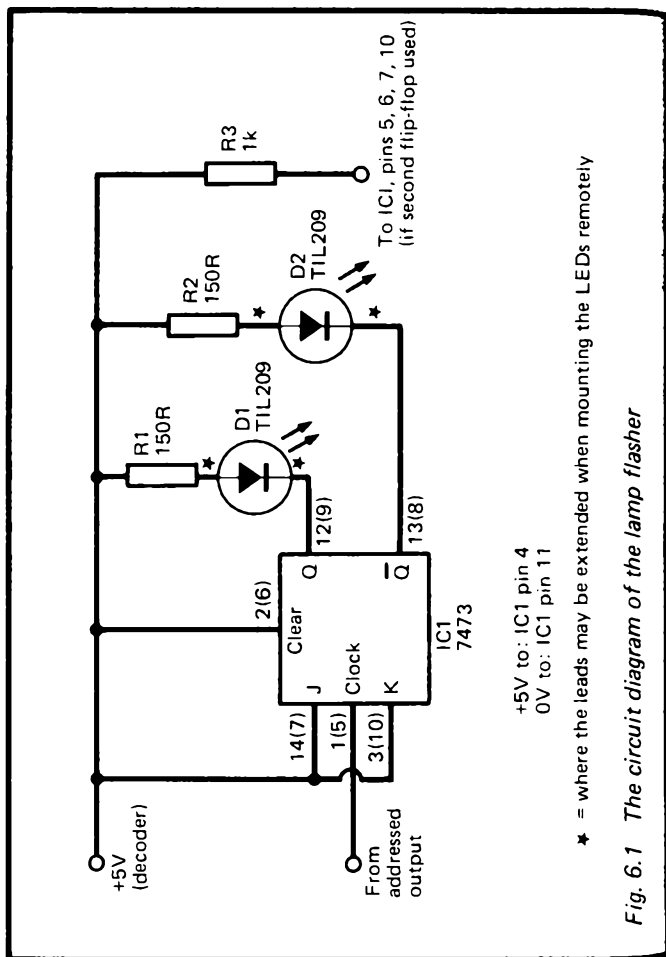
This circuit flashes one or two lamps (possibly more) on and off, whenever commanded to do so by the micro. It has lots of applications in models, from flashing the 'eyes' of a robot, to controlling signal lamps in model railway systems. It has uses in games and more serious uses around the house for operating warning or other indicating lamps. The lamps used are light-emitting diodes, which are obtainable in several colours – red, orange, yellow or green. There is also the possibility of using infra-red LEDs, which could have applications for remote control of devices not actually wired to the computer. It is also possible to modify the circuit to control small filament lamps.

#### How it works

The basic circuit consists of a J-K flip-flop. LEDs are connected to its two outputs, Q and  $\bar{Q}$  (Fig.6.1). These outputs have opposite states, so one LED is on when the other is off. This action can be made good use of for a pair of warning lamps. For a model railway, with red LEDs placed side by side, it imitates the typical pair of flashing lamps found beside the road at the approach to a level crossing. If you want to flash a single lamp on or off, you need connect only one to the flip-flop.

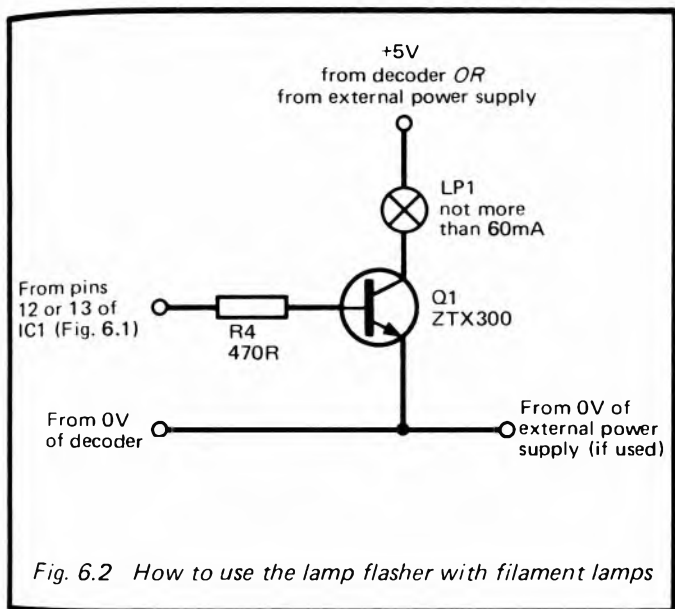
The J and K inputs of the flip-flop are wired to the +5V line. The effect of this is to make the flip-flop change state whenever a pulse is applied to the 'clock' input. This is known as a toggle action. If a given LED is off and is to be switched on, the micro addresses the flasher. This makes the Addressed Output of the decoder go low for an instant, causing the LED to come on. The LED stays on until the next time the lamp flasher is addressed, when it goes off. Note that it is when an output from the flip-flop goes low that the LED lights.

Several modifications of the circuit are possible. The out-



puts from the flip-flop are insufficient to drive a filament lamp directly, but with the addition of a transistor (Fig.6.2) it is easy to switch such a lamp. Here the lamp is on when the output from the flip-flop is *high*. If the lamp has low current requirements (not more than, say 60mA) and if it is rated to





*Fig. 6.2 How to use the lamp flasher with filament lamps*

operate on +5V or a voltage close to this, it is reasonable to use the +5V power supply of the micro. However, if you are operating other interfaces at the same time there may not be current to spare for this. In this event, or if the lamp requires higher voltage, the lamp must be powered from some external source, such as a battery or a power supply. The power supply *must* be direct current. The voltage required depends on the rating of the lamp, but should not exceed 25V; voltages greater than this would damage the transistor. The gain of the transistor limits the lamp current to about 40mA, so a 60mA lamp may not light at full brilliance.

The IC contains two flip-flops, only one of which is used. The pin connections for the second IC are shown in brackets in Fig.6.1. Both 'clock' inputs can be connected to the same Addressed Output, allowing two pairs of lamps to be controlled. Alternatively, you can use a separate Addressed Output for each, controlling the flip-flops independently.

## Building it

The circuit takes up so little room that, when used in connection with a model, it may be hidden away inside the model itself. Otherwise it needs a small plastic box to contain it. Wiring up the circuit presents no problems. No address decoding is required, since this is done on the decoder board (Fig.D.1). The only connections to the decoder board are the two power lines and the Addressed Output line. If you intend to place the lamps at some distance from the micro (for example in another room, or on a mobile model) it is better to have the circuit itself close to the micro and run pairs of wires to the LEDs. The asterisks in Fig.6.1 show where the leads should be extended.

When you have assembled the circuit, check that there are no short-circuits between the three wires which connect it to the decoder. Then plug it into the decoder at one of the Addressed Output plugs, and switch on the power. One of the LEDs comes on, though it is not possible to say in advance which one this will be. Run a short program to write to the device. On the Spectrum, use 'OUT 31,0'. If necessary, change the '31' to whatever address you have chosen for the lamp flasher. The '0' is a dummy value, for we are not actually sending any data to the circuit. The mere act of addressing it is enough to trigger it.

The program given on p. 176 for the ZX81 will trigger the LEDs to change when run. On the Ace, the command is '0 31 OUT'.

## Programming

As explained earlier, the lamps are triggered to change state every time the lamp flasher is addressed. This program for the Spectrum will make the two lamps flash on and off 10 times:

```
10 FOR J=1 TO 20
20 OUT 31,0
30 PAUSE 50
40 NEXT J
```

An equivalent effect can be obtained on the Ace by defining the word WAIT (as on p.151 of the manual) and then this word:

```
: FLASH 20 1 DO 0 31 OUT WAIT LOOP ;
```

Entering FLASH causes the LEDs to flash on and off 10 times.

Of course there is a lot more to programming than simply making the LEDs flash. Commands are inserted in a wide variety of programs whenever the flip-flop is to change state. When using the LEDs as indicators, the rate of flashing can be varied according to what information it is intended to convey. An interesting programming project would be to devise a Morse Code sender. This would be a helpful practice device. The user types in a message at the key-board of the micro. When 'Enter' is pressed, the micro converts the message into the equivalent symbols of the Morse Code and then flashes a single LED to transmit the message.

### *PARTS REQUIRED for the LAMP FLASHER*

#### *Resistors*

R1, R2 150R (2 off)

#### *Semiconductors*

D1, D2 TIL209 or similar light-emitting diodes (any colour, 2 off)

#### *Integrated Circuit*

IC1 7473 dual J—K flip-flop, with clear. (Note: this project requires the standard TTL version not Low-Power Schottky (LS) version.)

#### *Miscellaneous*

Circuit board

14-pin IC socket

3-way socket to fit 3-way plug on decoder

1mm terminal pins (3 off; 7 off if LEDs are mounted remotely)

## Project 7

### LIGHT PEN

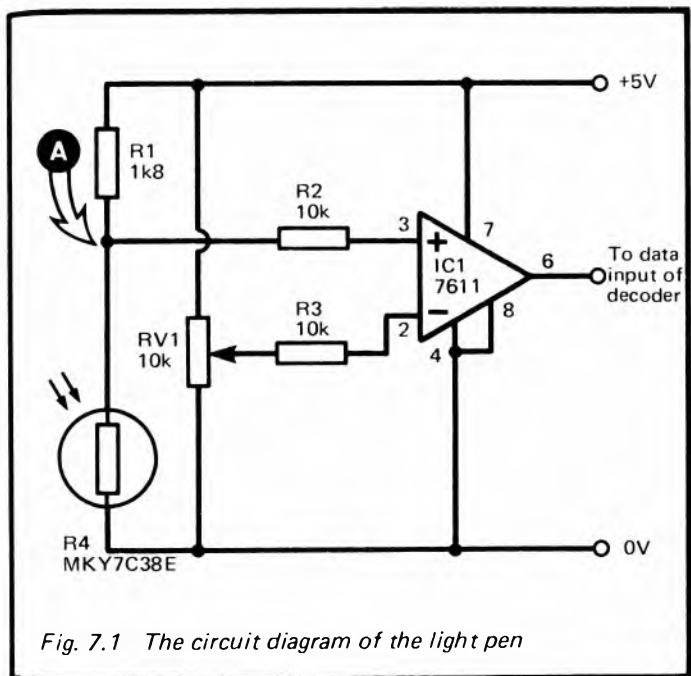
You may already have seen a light-pen being used with a computer and know some of the ways in which it can be used. Often it is used to 'draw' on the screen. The user moves the pen across the screen, just as if drawing, and a line appears on the screen, following the tip of the pen. This Light Pen can be used in the same way, though rather more slowly than the commercially made pens, but it has the advantage of being extremely easy and cheap to build.

There are several other ways in which the light pen is used. For example, it is used in selecting choices from a 'menu' displayed on the screen. Beside each item there is a flashing patch. You point the pen at the patch beside the item of your choice and the computer then does as requested. The same idea can be applied to learning programs that have multiple-choice questions. A flashing patch appears next to each of the answers. You point the pen at the patch beside the answer which you think is correct. Many people are unfamiliar with the layout of a typewriter keyboard and perhaps slightly apprehensive about trying to use one. For such people it is easier to point with a light pen than to look at a keyboard and try to find and press one of several possible keys.

There are applications for the light-pen in games too. Instead of making your moves by using the key-board or a joystick, use the light-pen.

### How it works

A light pen works by sensing light coming from the screen. In this design, the sensor is a *light-dependent resistor* or LDR, often called a *cadmium sulphide photoconductive cell* (PCC). The resistance of an LDR is inversely proportional to the amount of light falling on it. Its resistance is very high in darkness (a megohm or more), but decreases to only a few hundred ohms when it is in the light. In Fig.7.1, R4 is the LDR. It is in



*Fig. 7.1 The circuit diagram of the light pen*

series with a fixed resistor, R1. Together, they make up a potential divider. In darkness or dim light, the potential at point A is high, since the resistance of the LDR is much greater than that of R1. As the amount of light falling on the LDR is increased, the potential at A falls. In very bright light it is only a fraction of a volt.

The potential at A is compared by an operational amplifier (IC1) with a reference potential which is set by adjusting the variable resistor RV1. The amplifier is very sensitive to small differences of potential. If the potential at A is a little higher than that at the wiper of RV1, the output of the amplifier falls very sharply, almost to 0V. On the other hand, if the potential at A falls slightly so that it becomes just a little lower than that at the wiper of RV1, the output voltage swings sharply

upward, almost to +5V. In this way the circuit gives a low output in darkness or dim light, and high output in brighter light. There is a rapid swing at a fixed level of brightness. We can choose this *threshold* level by the position in which RV1 is set.

The output from the amplifier is a clear cut signal which is entirely suitable for feeding to a TTL logic gate. The output is taken directly to one of the Data Input terminals on the decoder (Fig.D.1).

## Building it

The circuit is connected to the decoder board by only three wires: the two power lines, and a wire to one of the Data Input plugs. There are two wires going to the LDR and two to the variable resistor, which is best mounted on the case of the project. It is possible to use a miniature preset resistor for RV1, and to set the level once and for all. However, it is convenient to be able to adjust the threshold level occasionally, especially if you are using displays with backgrounds of differing colours, so a 'volume control' type is preferred.

Almost any type of LDR can be used with this project. The one specified is a relatively small one, measuring 8mm in diameter. One of the larger types, such as the ORP12, will work just as well. The resistance ranges of different types vary; the resistance of R1 should be about the same as that of the LDR in average room lighting conditions. If necessary, substitute a resistor of different value for R1.

There is scope for experimentation and ingenuity in making the pen. In its simplest form (Fig.7.2) it consists of a tube into which the LRD fits fairly tightly. The LRD needs to be set back about 3cm from the open end so as to confine its field of view to a small area of the screen. With the ZX computers the size required is a little smaller than one of the graphics blocks. The actual dimensions of this depend on the size of the screen of your TV. With the tube held close to the screen the LDR should receive most of the light from a single graphics block, but none from adjacent blocks. It will help if the inside of the tube is painted dull black. Use a felt-tip spirit marker for this. If you use a metal tube, take care to cover its end with a softer

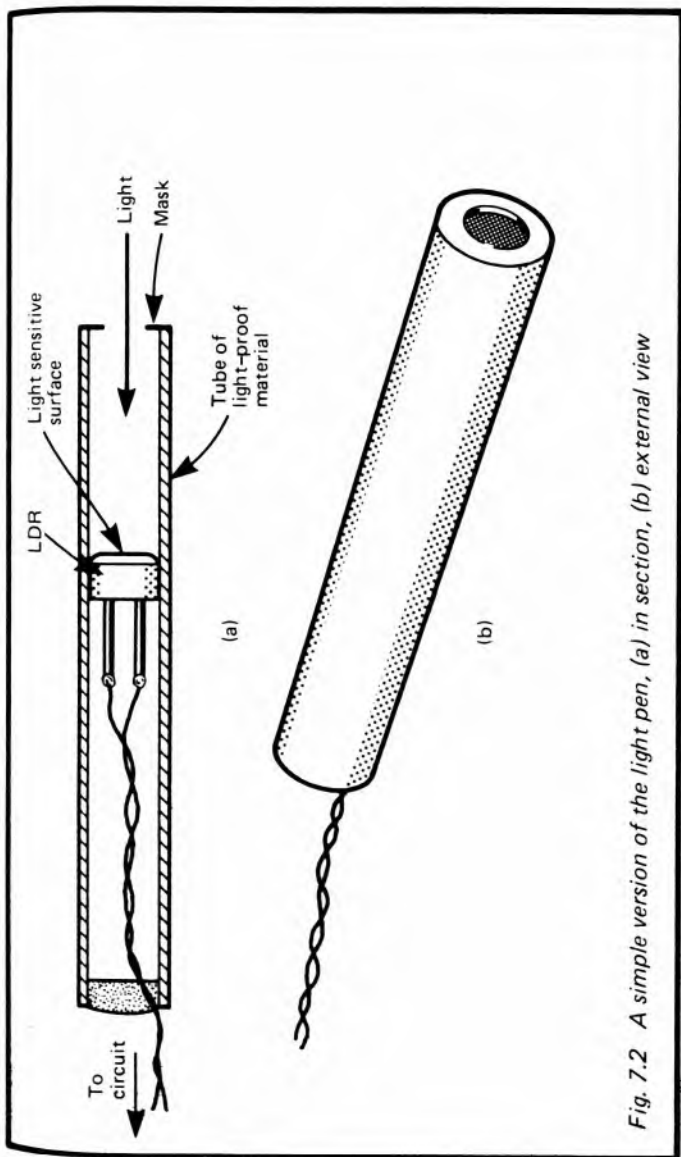


Fig. 7.2 A simple version of the light pen, (a) in section, (b) external view

or more flexible material so that there is no danger of scratching the front surface of the TV tube.

If you are using one of the larger LDRs, the tube into which it fits needs masking at one end to reduce the view to a single graphics block. A wider tube may make it more difficult to see the screen when you are drawing, so a better plan is to make a tube which tapers toward the open end.

A simple tube must be held close to the screen, almost touching it, to sense light from the selected area only. If you fit a small lens to the end of the tube and adjust the position of the LDR so that light from a small area of the screen is concentrated on it, you will be able to hold the pen further away. The only difficulty then is that it becomes harder to know exactly where the pen is pointing.

When the circuit is complete, connect it to a power supply (+5V or +6V) and connect a voltmeter to the output of IC1. Switch on the micro and the TV. Print a black graphics block on the screen, if the background is white, or print a white block on a black ground. Point the pen squarely and accurately at a white area of screen. Adjust R1, if necessary, to make the voltmeter read +5V (or a reading very close to this). Now move the pen to point to a black area. The voltage falls almost to zero. If it does not, RV1 needs further adjustment. You will soon find the position of R1 in which the voltage swings sharply to 0V or +5V as the pen is pointed at white or black areas.

The Light Pen may now be plugged into the decoder and tested on the micro. Here is a simple test program for the ZX Spectrum:

```
10 PRINT AT 20,20;"■■"  
20 PRINT AT 21,20;"■■"  
30 PAUSE 5  
40 PRINT AT 10,10; IN 31  
50 PRINT AT 20,20;"  "  
60 PRINT AT 21,20;"  "  
70 PAUSE 5  
80 PRINT AT 10,10; IN 31  
90 GO TO 10
```

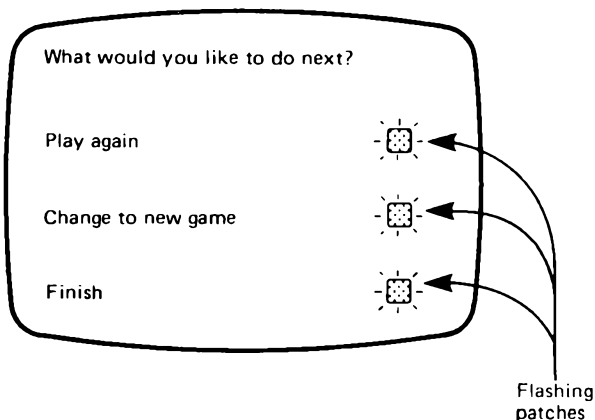


Users of ZX81 will need to modify this program as explained on p. 174. If your pen is attached to one of the other Data Input plugs, alter the address in lines 40 and 80 accordingly. This program puts a flashing square of four graphics blocks on the screen. Point the pen at the square. As it flashes, you will see the number (which indicates the input) alternating between 255 (white) and 254 (black). This is because unused data lines count as 'high' while the used line (D0) is alternating between 'high' and 'low'. This gives binary numbers 1111 1111 (=255) and 1111 1110 (=254) respectively. When you move the pen away from the square to point at a white area of screen, the number stays at 255. If you cover the end of the pen to exclude the light, the number stays at 254. If you do not get the changes described here, it is likely that RV1 may need further adjustment to put the threshold point in the range of brightness of your TV.

The pauses in this program give the pen time to respond to the changes in brightness, and also gives you a chance to see that the pen is responding correctly. The lengths of the pauses can be reduced to PAUSE 3 and, with more precise adjustment of RV1, eliminated altogether.

## Programming

Although this is such an easy project to build, you need to think out the programming very thoroughly. The two main ways of using it are (i) to select a choice from a menu or other display on the screen, and (ii) to draw on the screen. Let us consider choice selection first. The choices (items of the menu, answers to a question etc.) are printed on the screen with a flashing patch beside each one (Fig. 7.3). The best approach is to print the choices first then to print the patches using PRINT AT. The patches need be only a *single* graphics block in size, so you alternately print a white block (a space) and a black block (an inverse video space) in quick succession. The trick is to flash only one patch at a time. We begin with a white screen. Then (supposing there are three choices and three patches) patch no. 1 is made black. This is followed by a pause (say, PAUSE 3) after which the input from the pen is read. If the pen is already pointing at that patch its output



*Fig. 7.3 A menu displayed with flashing patches for use with the light pen*

goes low, so you need a program line which finds out what input is coming from the pen at that instant, and acts accordingly. It might be something such as:

```
50 IF IN 31=254 THEN GO TO 200
```

On the ZX81 you will need to use the input routine given on p. 174 and the value 65534 instead of 254. If the input is not 254, this means that the pen is not pointing at that patch, but

possibly at one of the other patches. So patch no 1 is made white again and patch no 2 is made black. After a pause, comes another line to read the input from the pen, perhaps like this:

```
80 IF IN 31=254 THEN GO TO 450
```

If this test fails, the pen is not pointing at patch no 2 — perhaps it is at patch no 3, or possibly it is just pointing at some other part of the screen. The next step is to flash patch no 3. If the pen is not there either, the program goes round in a loop to flash the patches in order again. Sooner or later it finds the pen in this way and the micro is sent to one of a number of sub-routines according to which patch the pen is pointing.

You may have noticed the slight complication that if the pen is not pointing at the screen at all, its output is 'low' all the time and the test at line 50 will succeed, sending the program to line 200. One way of dealing with this is to keep the pen pointed at the screen when it is not being used. Another is to make the program wait until the pen is certainly pointing at the screen:

```
40 IF IN 31=254 THEN GO TO 40
```

A line like this is needed before all the lines which make the patch black. This ensures that the input really *is* 255 before the micro makes the patch black and tests if it changes to 254. There are various ways of programming for drawing on the screen. The commercial systems find out where the pen is pointing by scanning the whole screen, row by row, with a moving flashing patch. The input from the pen is read at every point. When the patch reaches a point at which its flash is followed immediately by a change in output from the pen, the micro knows that it has 'found' the pen. It plots a black graphics block there (or white if the background is already black). This kind of program is not too hard to write, but unless you write it in machine code, it is extremely slow. Also the time required for this simple pen to respond to the flash would still make it a very slow program.

There are two ways of speeding things up. One way is to

make the drawing start at a fixed point on the screen. The program scans only the 8 blocks adjacent to this area, trying to find out where the pen has moved to. When it finds the pen it plots a mark at the new position of the pen. Then it scans the 8 areas adjacent to this area, and so on.

Here is the program, to run on a Spectrum:

```
10 DIM s(21,31)
20 LET y=10
30 LET x=15
40 PRINT AT y,x;"■"
50 LET s(y,x)=1
60 PAUSE 5
70 IF IN 31=255 THEN GO TO 60
80 LET j=y-1
90 LET k=x-1
100 PAUSE 5
110 IF IN 31=254 THEN GO TO 100
120 IF s(j,k)=1 THEN GO TO 170
130 PRINT AT j,k;"■"
140 PAUSE 5
150 IF IN 31=254 THEN LET y=j:
    LET x=k: LET s(j,k)=1: GO TO 80
160 PRINT AT j,k;" "
170 LET k=k+1
180 IF k<=x+2 THEN GO TO 100
190 LET j=j+1
200 IF j<=y+2 THEN GO TO 90
210 STOP
```

This program is easily adapted for the ZX81 (see p. 174). It is worth while studying this program in some detail as it illustrates several points of importance in operating the light pen successfully. First of all the program sets up an array (line 10) which holds a 'map' of the screen. The idea of this is to keep a record of which squares have been drawn on. Though it is possible to PEEK the video RAM for this information, there are three equations to be used in different areas of the screen. This is too complicated. Unfortunately, having to set aside memory for the array means that this program needs more

memory than the unextended ZX81 possesses. After printing a black block in the centre of the screen the program waits (lines 60–70) for the input from the pen to change to 254, which is taken to indicate that the pen has been placed on the block. As soon as the pen is placed there the variables  $j$  and  $k$  are initialised ready to begin scanning (Fig. 7.4). Nothing else happens now until the pen is moved off the first block (lines 100–110) and the input becomes 255. As soon as this happens, it is assumed that the pen has been moved on to one of the adjacent white blocks. Note the pause at line 100, which gives the pen time to *move away* from one block and *arrive* at the next. The program then scans the adjacent blocks, making each one black in turn (line 130) and reading input to see if it

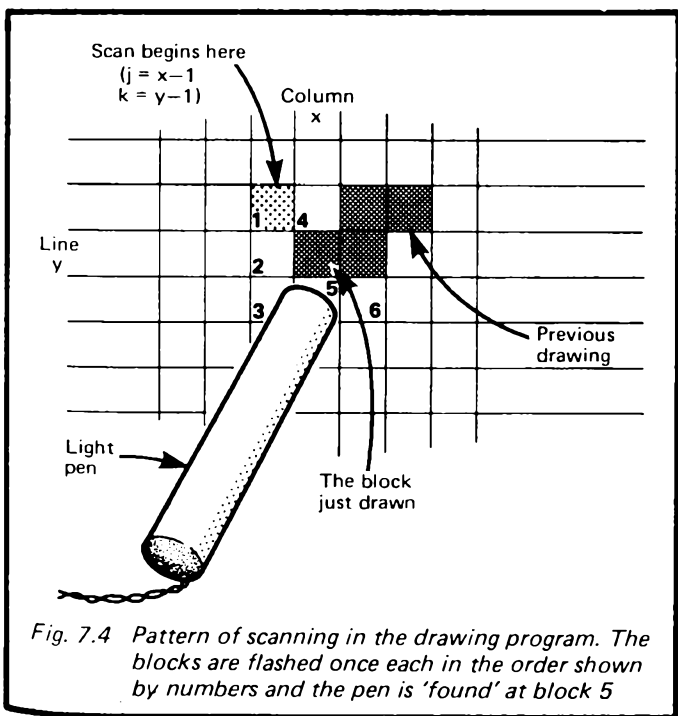


Fig. 7.4 Pattern of scanning in the drawing program. The blocks are flashed once each in the order shown by numbers and the pen is 'found' at block 5

has fallen to 254. However, before leaving this it looks to see if this block has already been drawn on and, if so, (line 120) skips the flashing routine. This prevents the program from erasing a block it has already drawn. If the pen happens to be pointing at one of these blocks by mistake, odd things may happen. Even this fairly elaborate program needs improving!

If it 'finds' the pen (line 150), it updates the values of x and y, (representing the current position of the pen), to the values of k and j, (the coordinates of the square being flashed). A black block is printed there and is also recorded in array s. Having done this, the program begins again from line 80, basing its scan on the new values of x and y.

When using this program it is important to move the pen to *exactly* where you want the next block to be. If it is half-way between one block and the next, flashing either block may fail to reduce the input to 254, because of light reaching the pen from the unflashed block. In this event the scan fails to 'find' the pen and the program runs through to the end. Rather than lose a drawing by such an accident, position the pen correctly, then type in the direct command GO TO 80.

Another approach to programming relies on the DRAW command of the Spectrum and Ace. Scan the whole screen, row by row with a flashing patch. This takes longer than merely scanning adjacent squares but time is saved when the pen is 'found', for the program can be made to DRAW a line directly from the last location of the pen to the present location. Another variation is to use the CIRCLE command. Point the pen where the centre of the circle is to be. The screen is scanned and this point 'found'. Then point the pen at any point location on the circumference of the circle to be drawn. The micro 'finds' this, computes the radius and plots the circle. You could work out similar programs to plot squares, triangles or other shapes.

### *PARTS REQUIRED for the LIGHT PEN*

*Resistors (carbon, 0.25W, 5% tolerance)*

R1        1K8 (depending on average resistance of R4)

R2, R3 10k (2 off)

#### *Other Resistors*

MKY7C38E light-dependent resistor: dark resistance 300k,  
sunlight resistance 100 ohms: ORP12 is  
possible alternative but larger

RV1 10k variable potentiometer

#### *Integrated Circuit*

IC1 7611 CMOS operational amplifier

#### *Miscellaneous*

Circuit board

8-pin IC socket

3-way socket to fit 3-way pin on decoder

Knob for RV1

1mm terminal pins (7 off)

Materials for making the body of the pen

Connecting wire, including light flexible wire for the lead to  
the pen

## Project 8

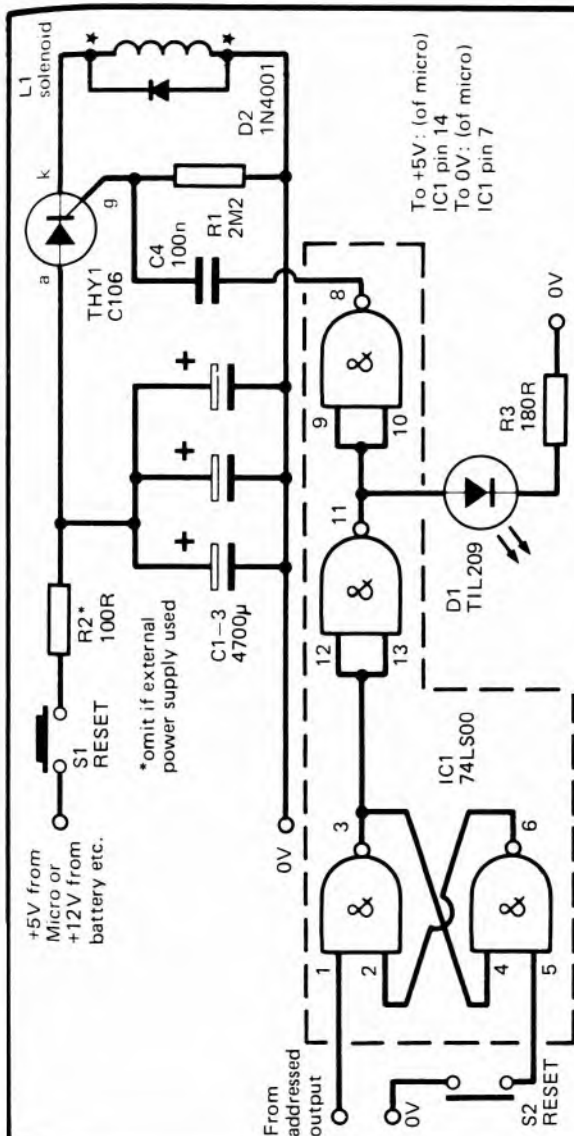
### MAGNETIC CATCH

This circuit has a once-for-all action. It is reset manually and remains reset until triggered by the micro. Triggering leads to some kind of mechanical action. After this, it must be reset manually again. Exactly what action occurs when the catch is triggered depends on the application to which it is put. It could be triggered to release the bolt on a door, so, for example, letting the cat out early in the morning. Or maybe the bolt is on the door of a cupboard which is to be opened only when the correct secret password is typed into the computer. It could be used to discharge a quantity of food into an aquarium, so overcoming the problem of feeding the fish on Saturday evening when you are away for the weekend. If you are away for the day, and if the thermometer of Project 15 is used to monitor the temperature in your greenhouse, the magnetic catch can be triggered by the micro when the temperature gets too high. It opens the window or door or lets the sun-blinds roll down over the roof.

#### How it works

The action of the catch depends upon briefly energising a coil, or solenoid. There is a sliding soft-iron core partly inside the solenoid. When a burst of current flows, the magnetic field so created pulls the core swiftly into the solenoid. This motion is used to perform whatever mechanical action is required. One of the problems with solenoids is that the more powerful ones have coils with a relatively low resistance. They would draw too much current from the micro. If a battery supply was provided, the batteries would soon be exhausted if the coil were to remain energised for periods of a few hours. This circuit uses capacitors to store the energy needed to activate the solenoid (Fig.8.1). When the circuit is being reset, button S1 is pressed and charges the capacitor up to maximum voltage. If the supply is taken from the micro this is +5V. If the solenoid





★ = the leads to the solenoid may be extended at these points

Fig. 8.1 The circuit diagram of the magnetic catch

requires a higher voltage for satisfactory operation, it is charged from a battery or other external supply.

The capacitors are prevented from discharging through the solenoid immediately because there is a thyristor in the circuit. This does not conduct until a high pulse is sent to its gate electrode (g). Otherwise the capacitors remain charged for many hours, and during this time the only power required to operate the catch is that needed by IC1 (about 8mA).

The gate electrode is held at low voltage by resistor R1. The gate is also connected, through a coupling capacitor C4 and two inverting gates, to the output of a flip-flop. When this flip-flop is reset by pressing S2, its output goes low. The inverted output goes 'high', putting the LED on to indicate that the circuit is now reset. The doubly inverted output goes 'low'.

The flip-flop is triggered by a low-going pulse from an Addressed Output of the decoder (p. 158). When the micro addresses the catch, the flip-flop changes state. The LED is turned off and a high pulse passes through C4 to the gate of the thyristor. This causes the thyristor to begin conducting. It conducts very easily and the capacitors are discharged almost instantaneously through the solenoid. This creates the strong magnetic field required to operate the catch. Once the capacitors have been discharged, no further action occurs until the flip-flop has been reset and the capacitors have been recharged.

If the capacitors are to be charged from the +5V supply of the micro, it is essential to include a resistor (R2) in the charging circuit. Otherwise, the sudden drain on the power supply of the micro is too great and its program is lost. The resistor reduces the charging current to a manageable amount, though it then takes about 10 seconds to recharge the capacitors.

The mechanical side of the project is left to the ingenuity of the reader, for so much depends on the application for which it is intended, the strength of the particular solenoid used, and the skill of the reader in constructing mechanical devices. Fig.8.2 shows how a pull from the core can be made to release a prop, so causing a window (e.g. of a greenhouse) to fall shut under the action of gravity. Fig.8.3 shows how

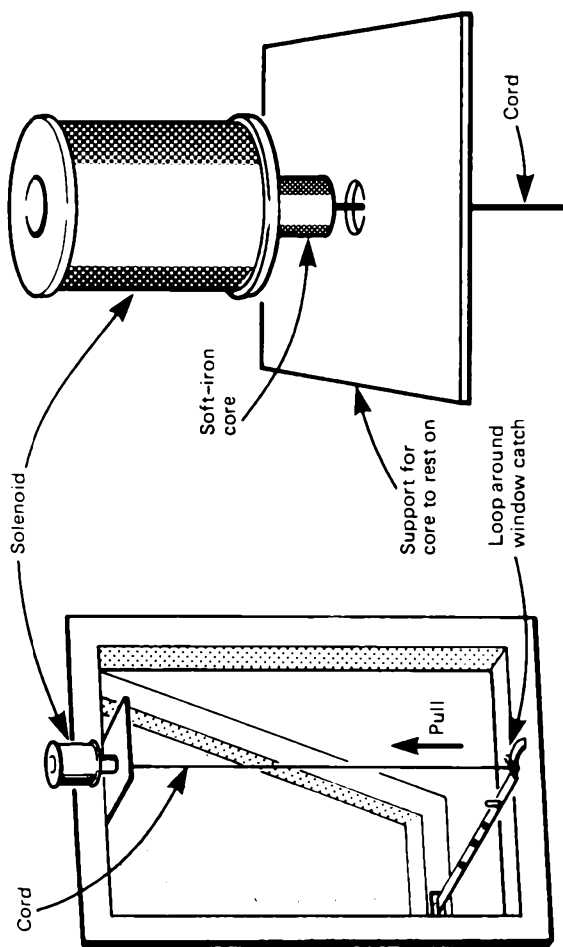


Fig. 8.2 A simple mechanism to close a window

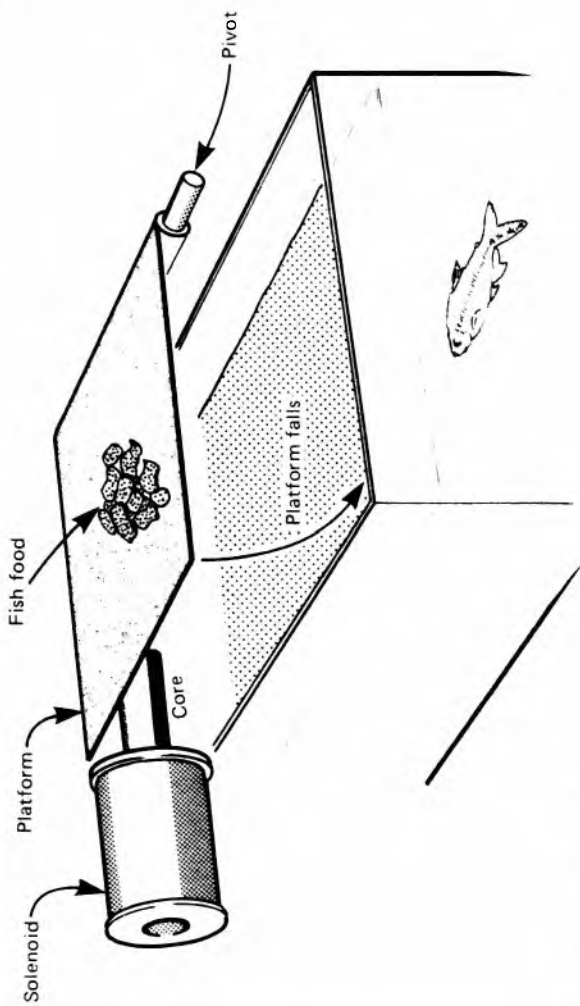
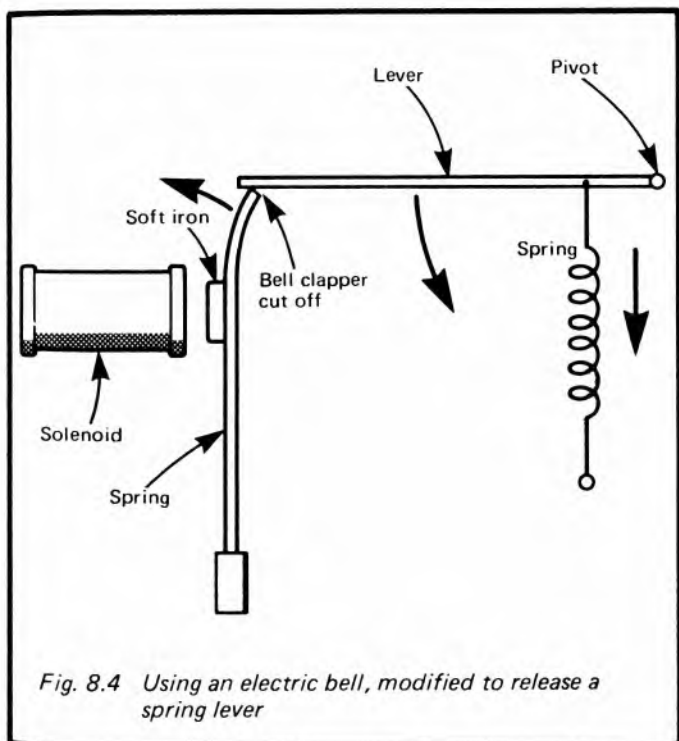


Fig. 8.3 A mechanism for feeding aquarium fish

platform carrying the required daily quantity of fish food may be released, so scattering the food into the aquarium below.

As an alternative to a solenoid with a sliding core, the reader may prefer to use a solenoid with a fixed core which attracts a moving armature. An old electric bell or buzzer could be adapted for this purpose. Fig.8.4 shows how the motion of an armature can release a sprung lever, so initiating a relatively more forceful operation.



## Building it

The catch needs no address decoding other than that already provided for on the decoder board (Fig. D.1). It requires three wires to connect it to the board; the two power lines, and the wire connecting it to an Addressed Output. The circuit should be connected to the micro by fairly short wires. If the solenoid is to be located at some distance from the micro its leads may be extended at the points marked \* in Fig.8.1. The circuit is best housed in a case, on which are mounted the two push-buttons and the LED. If you are using a battery for charging, the case needs to be large enough to hold this too.

It is best to design, build and test the mechanical side of the catch before constructing the electronic side. It is essential to make certain that the solenoid you intend to use will develop enough power to actuate the mechanism. You can connect the capacitors together on a bread-board or by leads ending in crocodile clips. Then charge them from a battery and try discharging them through the solenoid. If the force developed is insufficient, there are several ways of increasing it:

- i. Begin with the moving core further inside the coil (though this also reduces its length of travel)
- ii. Use a solenoid with more turns of wire
- iii. Add more capacitors in series with C1—C3
- iv. Charge the capacitors to a higher voltage (assuming this is within the rating of the solenoid)

It is also worth while checking that the mechanical design is sound. For example, parts that are supposed to slide should do so smoothly, without undue friction. A transverse force applied to the core by the weight of an attached lever may prevent it from sliding easily into the solenoid.

Once assembled, the electronics are tested for short-circuits between the lines which connect it to the micro. It is then ready for testing on the micro. Plug it on to the decoder board. Key in this short program.

```
10 OUT 31, 0
```

This simply addresses Addressed Output 0. If you are using

another address, as in Table D.2, substitute this in the line above. Running the output program on p. 176 has the same action with the ZX81. On the Ace, use 0 31 OUT.

Press S2 to reset the flip-flop; the LED comes on. Next press S1 to recharge the capacitors. Hold the button pressed down for about 10 seconds if you are charging from the micro's own power supply. Reset the mechanism so that it is ready to be triggered. When all is reset, RUN the program. The solenoid is energised, the mechanism is operated and the LED goes out.

## Programming

The essential program line is the one given above. Apart from that, the rest depends on the application.

### *PARTS REQUIRED for the MAGNETIC CATCH*

*Resistors* (carbon, 0.25W, 20% tolerance)

- R1 2M2
- R2 100R (required if micro power supply used for charging)
- R3 180R

*Capacitors*

- C1–C3 4700 $\mu$  electrolytic (3 off)
- C4 100nF polyester

*Semiconductors*

- D1 TIL209 or similar light-emitting diode
- D2 1N4001
- THY1 C106 or similar thyristor

*Integrated Circuit*

- IC1 74LS00 quadruple 2-input NAND gate

*Miscellaneous*

- L1 Solenoid with soft-iron moving core, rated to operate at 6V, 12V etc. (alternatively an old electric bell, buzzer or relay)
- S1, S2 Push-to-make Push-button switches (2 off)
- Circuit board
- 14-pin IC socket
- 3-way socket to fit 3-way plug of decoder

1mm terminal pins (9 off)  
Parts for making mechanism  
Connecting wire



## **Project 9**

### **LAP SENSOR**

Although this Project is called 'Lap Sensor', it has many other applications. In its original application as a lap sensor for slot-car racing, it detects when a car passes a given point on the track. Depending on how the micro is programmed, it can keep count of the number of laps, it can measure the lap time, or it can do both things at once. From this information it can calculate and display the lap speed and the race speed. Since the device works by the breaking of a beam of light passing across the track it does not matter what it is that breaks the beam. It could equally well be used for bicycle races, horse races or in many kinds of athletic track events.

Likewise, it does not have to count how many times the same object breaks the light beam, but could be counting how many times different objects break the beam. It can count people going into a room, cars going into a car park, or objects on a conveyor belt. It works so fast that it can count objects (or other causes of interruption of the beam) which come so quickly in succession that it would be difficult, if not impossible, to count them by eye.

Another class of application is that in which we want the computer to wait for and act on a single interruption of the beam. An obvious example is in connection with intruder detection. A beam of light across a corridor is broken by the intruder. The micro detects this and sounds the alarm. It could even be programmed to detect the setting of the Sun and then switch on the porch light!

### **How it works**

The light is detected by a photodiode (D1, Fig.9.1). This receives light from a source some distance away. A special lamp may be provided to focus a well-defined beam across the track or corridor, but this is not always essential. If a table-lamp with a 60W bulb is placed on one side of a corridor and

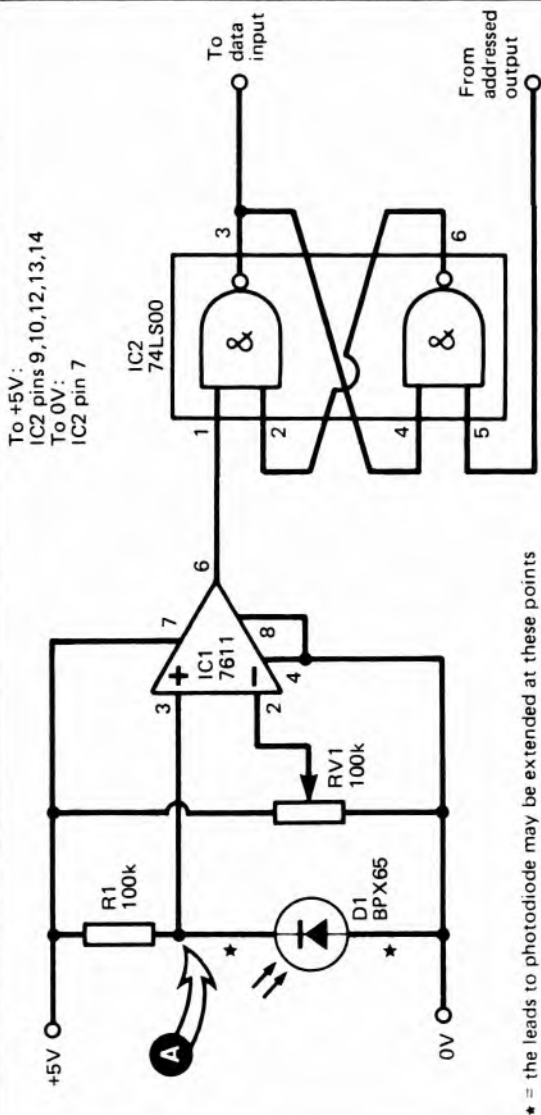


Fig. 9.1 The circuit diagram of the lap sensor

the photodiode is on the opposite side, pointing at the lamp, the shadow of a person passing between them is sufficient to trigger the sensor. On the smaller scale, (for example, on the slot-car track or model railway), an unshaded 6V 60mA torch bulb placed 10cm or slightly more from the lamp is entirely suitable as a source of light. The only point to look out for is that other lamps in the vicinity do not shine on the photodiode when the light from the source lamp is supposed to be interrupted.

The photodiode works best with light from filament lamps, but is relatively insensitive to light from fluorescent tubes. It works with daylight but, since the intensity of daylight varies so much with cloudiness and time of day, this is not a reliable source.

The photodiode is connected so that it is reverse-biased. The amount of leakage current increases in proportion to the amount of light falling on the photodiode. As the current increases the PD across the diode increases, so the potential at point A rises. The operational amplifier IC1 compares the potential at point A with a standard potential at the wiper of the variable resistor RV1. We adjust the standard potential to be very slightly *less* than the potential at A when the beam of light is unbroken. The tiny difference of potential is amplified and the output of IC1 swings sharply to +5V ('high'). When the beam is broken, the current through D1 decreases, the potential across D1 decreases, and the potential at A falls below that of the wiper of RV1. The potential difference between the inputs may still be tiny, but it is now a difference in the opposite direction. Consequently, the output of IC1 swings sharply to 0V ('low'). We can adjust RV1 to allow for the brightness and distance of the light source.

The output of IC1 goes to a flip-flop (IC2). The flip-flop is reset by a low pulse from an Addressed Output of the decoder (p. 158). In this state, its own output goes 'low'. The output of the flip-flop is connected to a Data Input of the decoder. In the 'reset' state the computer reads this output as a 0 on data line D0. When the beam of light is broken, even for a few milliseconds, the flip-flop is triggered to change state. Its output goes 'high'. When the micro next reads from the Lap Sensor's

address, it finds a 1 on line D0. The flip-flop remains in this state indefinitely, so it does not matter if the micro does not read the lap sensor *immediately* after the flip-flop has been triggered. This means that the micro does not miss brief interruption of the beam if it happens to be engaged in some other part of its program at the time it occurs.

When the micro has read the lap sensor and found that it has been triggered, it resets it by writing to its address. This causes a low pulse to appear at the Addressed Output, and this triggers the flip-flop to reset. Now it is ready to detect the next interruption of the beam.

## Building it

The circuit is housed in a small case with the potentiometer RV1 mounted on it. The photodiode can be hidden inside the case, with an aperture cut in the wall to allow light to enter. This arrangement helps screen off unwanted light from other sources. In certain applications it may be more convenient to have the light source and photodiode at some distance from the micro, perhaps in another room. In this even, the circuit should be close to the micro, but the leads to the photodiode may be extended where marked \* in Fig.9.1.

The circuit does not need any special address decoding, so it requires only 4 wires to connect it to the decoder board: the two power lines, and connections to an Addressed Output and a Data Input, preferably having the same address.

There are no difficulties in wiring up the circuit. When it is complete, connect it to a power supply (+5V or +6V) and connect a voltmeter to point A. Point the photodiode toward a nearby light source (e.g. a table lamp). As you place your hand between the lamp and the photodiode, the voltage at A should rise. Then connect the voltmeter to the output of IC1. Adjust RV1 until the output *just* swings to +5V. Interrupting the beam makes it swing sharply to 0V. Finally connect the voltmeter to the output of the flip-flop (pin 3, IC2). Briefly connect pin 5 to the 0v line to reset the flip-flop. Its output goes high (+2.5V or slightly more). When the light beam is interrupted, the output falls to 'low' (close to 0V). The circuit

is now ready to be tested on the micro.

Here is a simple test program for the ZX81. First of all type in and run the combined input-output routine given on p. 177. Remember to include the correct value for the address (Table D.2). Then delete lines 20 to 70 and type in the following:

```
20 LET X = USR 16514
30 PRINT X
40 LET X = USR 16521
```

Line 20 reads the output from the Sensor. If it has been triggered, line D0 is high, and so X takes the value 65535 (see p. 9). If the device is not triggered, the value is 65534. Line 30 prints the reading and the sensor is reset by the write command given in line 40. Run the program once just to reset the sensor. Then run it again; since you have not broken the beam before running it, '65534' is displayed. Now break the beam by placing your hand in it. The next time the program is run, the figure '65535' is displayed.

On the Spectrum the corresponding program is:

```
10 PRINT IN 31
20 OUT 31,0
```

Use another address if appropriate. The values 254 and 255 are obtained.

## Programming

The commands to read and to reset the lap sensor are easy enough to understand, but there are one or two points to consider when using them. Take as an example this program for the Spectrum:

```
10 LET n=0
20 OUT 31,0
30 IF IN 31=254 THEN GO TO 30
40 LET n=n+1: PRINT n
50 OUT 31,0
60 IF IN 31=255 THEN GO TO 50
70 GO TO 30
```

This is a program to count how many times the light beam is broken. It could be used for purposes such as those described on p. 87. Line 10 initialises the counter variable, n. Line 20 resets the sensor, just in case it goes into the 'set' condition when the power was first switched on. Line 30 waits for the flip flop to become 'set'. As soon as the beam is broken the input becomes 255 and the program drops through to line 40. Here the counter n is incremented and n is displayed. At line 50 the micro attempts to reset the sensor ready to count the next interruption. But micros work more quickly than most objects move, so it is likely that the beam is still broken. If this is so, the sensor will not reset. The micro waits in a loop (lines 50 and 60) continually attempting to reset the sensor and then reading it to find if it has been successful. Eventually, when the object has moved out of the beam and the light level is restored to normal, the sensor resets and line 60 detects that this has happened. The program then goes to line 70 and back to line 30 to wait for the next interruption. This example demonstrates that it is essential to reset the sensor at the beginning, and to allow the beam to be restored and confirm that the sensor has been reset after each interruption.

Lap timing programs need two routines similar to lines 30, 50 and 60 above, to detect an interruption and then reset the sensor. At the first interruption the time is recorded using the micro's internal clock. At the second interruption the time is taken again. The difference is the lap time. You read the micro's internal clock by PEEKing certain addresses. For the Spectrum the expression is:

```
LET time=(65536* PEEK 23674+256* PEEK 23673 +
PEEK 23672)/50
```

In the USA the divisor is 60. Details of how to use this expression are given on p.130 of the manual.

For the ZX81 the expression is:

```
LET T=(256* PEEK 16437 + PEEK 16436)/50
```

Again, the divisor 60 is used in the USA. On the ZX81, the expression allows timing of periods up to about 9 hours, but do not use the PAUSE command during this time, for it alters

the values contained in the two addresses. Indeed, for timing periods of more than a few seconds, it is easier to make the computer wait for, say PAUSE 100 (one second) and count the number of PAUSEs between the first time the beam is broken and the second time it is broken. This gives the Lap time to the nearest second.

Page 142 of the Jupiter Ace handbook describes some words for timing on this machine.

### *PARTS REQUIRED for the LAP SENSOR*

#### *Resistors*

- R1     100k carbon, 0.25W
- RV1    100k variable potentiometer

#### *Semiconductor*

- BPX65 photodiode (or almost any type)

#### *Integrated Circuits*

- IC1     7611 CMOS operational amplifier
- IC2     74LS00 quadruple 2-input NAND gate

#### *Miscellaneous*

- Circuit board and case
- Knob for RV1
- 8-pin IC socket
- 14-pin IC socket
- 3-way sockets to fit 3-way plugs on decoder board (2 off)
- 1mm terminal pins (9 off)
- Connecting wire
- A source of light

## Project 10

### PHOTO-FLASH

There are really two sections to the project, and you may use either one of them or both together. One section allows the micro to fire a photographic flash-gun. The other section is a sound sensor. The two are intended to be used together to fire the flash-gun when a sound is heard. One application of this is in high-speed photography. The camera is aimed at an inflated balloon (or even a glass bottle). The lights are turned out and the shutter is opened. Prick the balloon with a pin (or strike the bottle with a hammer) and the resulting noise triggers the flash. The result is a photograph of the bursting balloon or breaking bottle. Of course, a sound sensor could be connected *directly* to a circuit to fire a flash-gun, but having these circuits interfaced to a micro means that you can program the micro to delay the firing until a fixed period of time after the sound occurs. In this way you can obtain photographs taken at different intervals after the initial impact. With more than one flash circuit, the micro could fire a succession of flashes, producing multiple images on one frame.

Another application is to take photographs of nocturnal animals. A noise made by the animal triggers the flash. The sound detection circuit can be used on its own in many ways. It is particularly sensitive to sharp noises (clapping, snapping fingers) and to the higher-pitched whistles, so it provides a way of controlling the micro by sound. This section of the project could be used in conjunction with Project 4 to control the action of a model locomotive or slot-car by blowing a whistle, by clapping your hands, or by calling out "Go!" or "Stop".

Project 8, the Magnetic Catch is another one which can be controlled by sound, through the agency of the micro.

The photo-flash section can also be used on its own, perhaps simply to give a timed delay in taking a photograph of a group when you want to be included in the picture. Project 9 can be useful with the photo-flash. Instead of triggering the flash by sound, why not trigger it when a beam of light is broken?

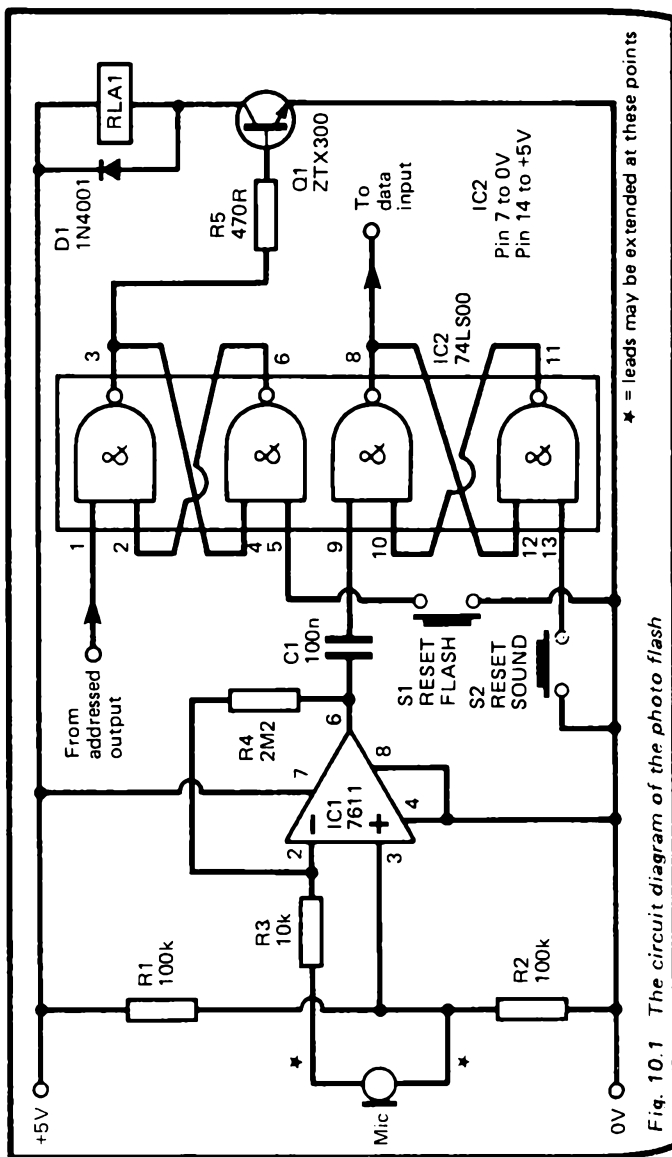


## How it works

The photo-flash section is controlled by an Addressed Output which is connected to a flip-flop (Fig.10.1). The flip-flop is reset by pressing S1. Its output is then 'low'. When the device is addressed, the low pulse from the Addressed Output sets the flip-flop. Its output goes 'high' switching on transistor Q1. The coil of RLA1 becomes energised and the contacts of the relay are closed. The relay contacts are connected across the terminals of a flash gun, which then fires. Note that power to operate the flash comes from the flash-gun, not from this circuit. The relay contacts do the same job as the contacts inside the camera which are normally used for firing the flash. Once the flash has been fired, the flip-flop must be reset manually before the flash can be fired again. It is possible to dispense with S1 and wire pin 5 of IC2 directly to another Addressed Output. This would allow the micro to reset the flip-flop ready for another photograph. If you are using the kind of flash-gun which requires flash-bulbs to be renewed there is little point in doing this. If you have an electronic flash-gun, it is possible to program the micro to make multiple exposures by firing the gun again after a short interval.

The sound sensor takes the signal from a crystal microphone and amplifies it by the operational amplifier IC1. In the absence of sound the output from the amplifier is about 2.5V. When sound is detected it oscillates above and below this level. Low-going voltage changes act as a low pulse to the other flip-flop, causing it to set. Its output, which is normally 'low', changes to 'high'. This can then be read by the micro. Note that the flip-flop stays high once it has been set, so it does not matter if the micro is not actually reading data from the sensor at the instant the sound occurs.

The flip-flop is reset by pressing S2. For the original application of this project, manual resetting is all that is required. However, in some applications it might be convenient to have the micro do the resetting. This is done by replacing S2 with a wire from pin 13 of IC2 direct to an Addressed Output of the decoder. This allows the micro to reset the flip-flop after a sound has been detected, ready to detect the next sound.



The device needs a minimum of four lines to connect it to the micro: the power lines, plus connections to an Addressed Output and a Data Input. It needs more lines if either of the flip-flops are to be reset by the micro, instead of being reset manually. Assuming either that the flip-flops are to be reset manually or, that they are to be reset by the micro using *separate* Addressed Outputs, all address decoding is done by the decoder, and there is no need to provide for this in the project itself.

The photo-flash should be housed in a case, on the outside of which the microphone can be mounted. The two push-buttons are mounted on the panel of the case. The switch contacts of the relay are wired to a socket, on the panel of the case. You need a plug to fit this socket, with wires leading to the flash-gun.

As explained earlier, this project consists of two parts, and there is no need to build both if you are interested in only one aspect of it. If you want to operate several flash-guns, it is easy to add a second 74LS00 IC from which two more flip-flops may be constructed.

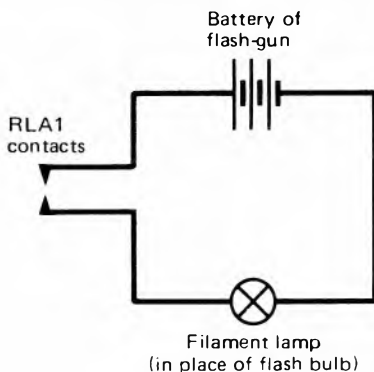
When assembly is complete and the project has been tested for short-circuits between the various lines which link it to the micro, it may be given its preliminary test. Connect it to a +5V or +6V supply. If you alternately press S1, and touch a grounded (0V) wire to pin 1 of IC2, you should be able to see or hear the relay switching on and off. Then connect a voltmeter to the data output (pin 8 of IC2). Press S2 and the output goes low. Make a sound near the microphone and the output abruptly goes 'high' (+2.5V or more). It responds best to sharp clicking or clapping sounds. Try snapping your fingers about 50cm from the microphone. Even a gentle 'snap' should be enough to trigger it. If this fails, try tapping the microphone gently. If this fails, the circuit is at fault. Failure to respond to sound of reasonable loudness may be due to lack of output from your microphone. To improve amplification, try substituting a resistor of higher value for R4. A resistor of 3M3 or 4M7 will increase the gain.

The photo-flash may now be tested on the micro. Plug its leads in the decoder and switch on. Press both reset buttons. If

you have an electronic flash, you can use this while testing. If you need to use expendable flashbulbs, this would be too expensive a session. Instead, wire up an ordinary low-voltage bulb, in place of the flashbulb shown in Fig.10.2. On the Spectrum, the command for flashing a bulb is OUT 31,0. Substitute another address in this line if appropriate. On the ZX81, simply run the output program given on p. 176. On the Ace, the word FLASH is defined like this:

```
: FLASH 0 31 OUT ;
```

When the sound-sensor flip-flop is reset its output is low, so reading from its address gives 254 on the ZX computers. When a sound has been detected, the reading changes to 255. The reading command is PRINT IN 31 on the Spectrum. On the ZX81 use the program on p. 175, and X takes the value '65534' or '65535'.



*Fig. 10.2 How to use a filament lamp while testing the circuit*

## Programming

Controlling the photo-flash is relatively simple, and several examples of possible uses for it have already been described. Here are programs which fire the flash a short period after a sound has been detected. This one runs on the Spectrum:

```
10 IF IN 31=254 THEN GO TO 10
20 PAUSE 5
30 OUT 31,0
40 PRINT "Photograph taken"
50 STOP
```

The PAUSE gives a delay of one tenth of a second (a twelfth of a second in USA). With the ZX81 type in and run the combined input-output program given on p. 177. Then delete lines 20 to 70 and type in:

```
20 LET X = USR 16514
30 IF X = 254 THEN GO TO 20
40 PAUSE 5
50 LET X = USR 16521
60 PRINT "PHOTOGRAPH TAKEN"
70 STOP
```

To program the Ace you need the word PAUSE, defined on p.147 of the Ace manual, and these three words:

```
: FIRE 0 31 OUT ;
: LISTEN BEGIN 31 IN 1 AND UNTIL ;
: PHOTO-FLASH LISTEN 10 PAUSE FIRE ;
```

The figure in the definition of PHOTO-FLASH sets the length of the interval between detecting the sound and firing the flash.

In the programs above the micro waits in a loop until it finds that the sensor has been triggered. Then, after a pause, the length of which you can adjust, it fires the flash. It would be a useful precaution to begin the program with a routine which reads the sensor to make sure that you have remembered to reset it first. If it finds that it is not reset, it displays a message to that effect. Then when it detects that you have reset it, it

goes on to the routine listed above, waiting for the sound to occur. Something of this sort will eliminate the risk of the micro firing the flash as soon as you run the program. Another refinement, especially useful if you are using the device to photograph any animals which might come close to the camera, is to insert a long pause in the program to give you time to get away before it becomes able to respond to sound. In this event it would be a good idea to have the sensor reset by the micro, so that the sounds of your departure do not permanently trigger the sensor.

If you are using the micro to reset the sensor after a sound is detected, the program must test that it has reset successfully. If it attempts to reset whilst the sound is still continuing, it will be unsuccessful (see p. 92).

As explained on p. 10, unconnected data lines in the Ace are 'floating' and may give either 0 or 1 when read. To read the state of line D0 alone, we simply perform the logical AND operation with the data reading. If it is ANDed with 0000 0001, the result gives the state of line D0, irrespective of the state of the other data lines. Thus we can read the output of the sound sensor by using the word:

: SOUND? 31 IN 1 AND . ;

Note that the photo-flash can have the same address for *both* of its sections, for the operations of reading and writing are independent.

### *PARTS REQUIRED for the PHOTO-FLASH*

*Resistors* (carbon, 0.25W, 5% tolerance)

R1, R2 100k (2 off)

R3 10k

R4 2M2

R5 470R

*Capacitor*

C1 100n polyester

*Semiconductor*

Q1 ZTX300 or similar npn transistor

D1 1N4001 100

### *Integrated Circuits*

- IC1      7611 CMOS operational amplifier
- IC2      74LS00 quadruple 2-input NAND gate

### *Miscellaneous*

- RLA1    Reed relay or other single-pole single-throw relay  
          operating on 6V
- S1, S2   Push-to-make push-button switches (2 off)
- MIC     Crystal microphone or microphone insert
- 8-pin IC socket
- 14-pin IC socket
- Socket for connection to flash-gun
- 3-way plugs to fit 3-way sockets on decoder board (2 off)
- Circuit board
- 1mm terminal pins (8 off)
- Connecting wire

## Project 11

### GAMES CONTROL

A games controls adds a whole new aspect to computer fun. It consists of a small box with a control knob on top. As you turn the knob one way or the other, you move a 'bat' round the screen, aim a 'laser gun', steer a 'racing car', or in some other way take a very active part in the game. You can build and operate 2 or more of these if you wish. Another possibility is to house this project in the same case as the 5-Key Pad (Project 3).

#### How it works

The control knob alters the setting of a variable resistor (RV1, Fig.11.1). This is wired between two 56k resistors, and the chain of resistors is connected between 0V and +5V. As the wiper of RV1 is moved from one end of its track to the other, the voltage at the wiper ranges between a little over 1.25V and a little under 3.75V. The voltage varies *smoothly* over this range. We say it is an *analogue* quantity. But micros are not able to accept such an input. A micro understands only two kinds of input, 'high' and 'low'. So, IC2 converts the smoothly varying analogue voltage into one which alternates between 'high' and 'low'. This IC is driven by a clock, built up from two NAND gates of IC1. The clock pulses cause a voltage (Vramp) inside IC2 to ramp down from about 3.75V to 1.25V. This actually happens in a series of 128 steps, but for all practical purposes it is a smooth ramp, as shown in Fig.11.2. Whenever the input analogue voltage is higher than Vramp, the output of the IC is 'high'. Otherwise it is 'low'. The result is that we get a waveform which has a fixed frequency (one 128th of the clock frequency), but has a varying mark-space ratio. As input voltage rises the 'mark' (or 'high' level) decreases, and the 'space' (or 'low' level) decreases. The micro is programmed to find out how much time is spent in the 'high' or 'low' states and, from this information it can work



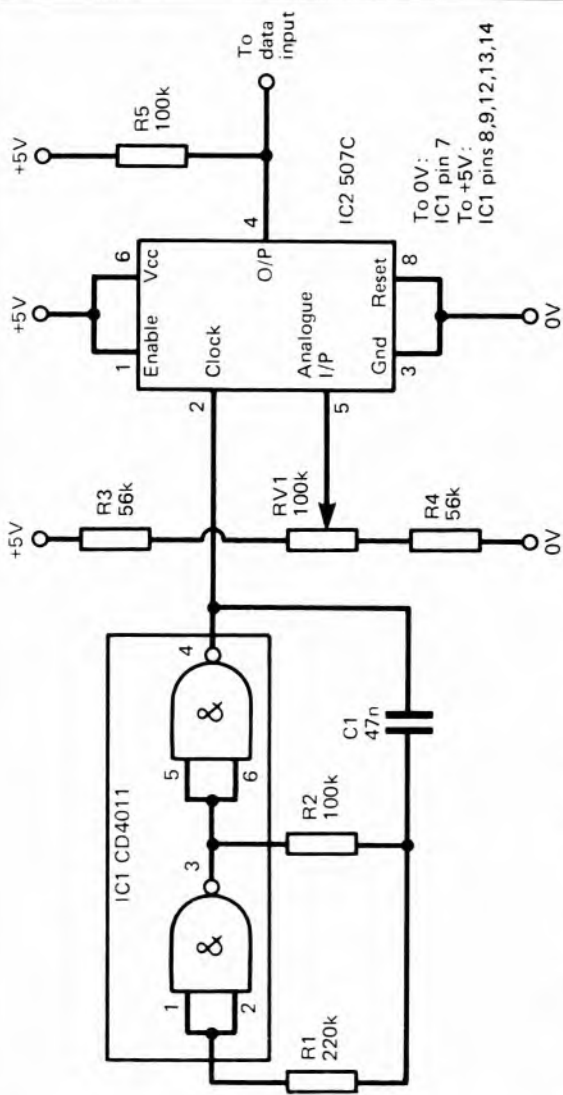


Fig. 11.1 The circuit diagram of the games control

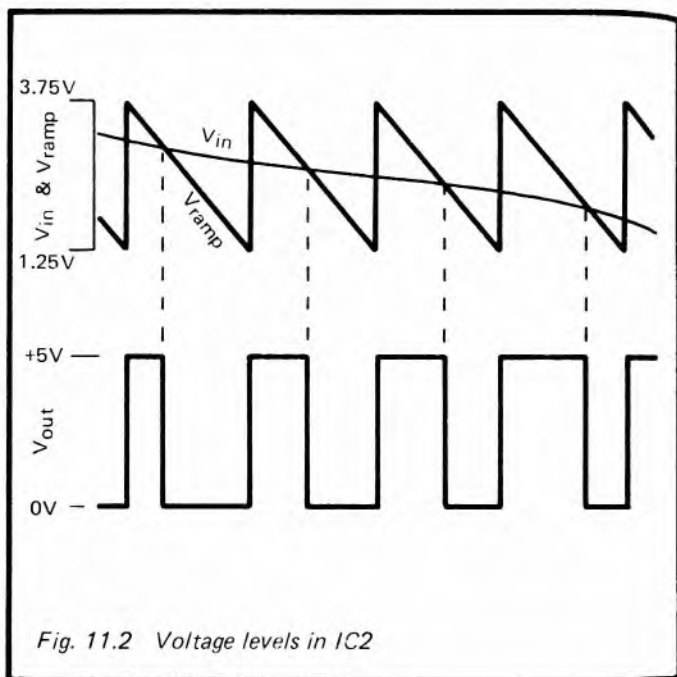


Fig. 11.2 Voltage levels in IC2

out the level of the input analogue voltage.

We call IC2 a *voltage-to-time converter*. It is not a true analogue-to-digital converter, for time, like voltage, is an analogue quantity. But time can be represented by 'highs' and 'lows' and in this form can be understood and measured by the micro.

## Building it

The device needs three lines to connect it to the decoder: the two power lines and a line to a Data Input. It is best housed in a small plastic box, preferably with a sloping top. The variable resistor is mounted on top, and the remainder of the circuit is assembled on a small circuit board placed inside.

If you have an oscilloscope, you can test the clock circuit and the output from IC2 after assembly. The clock runs at about 1kHz. Its exact rate is not important. The output from IC2 has a frequency of about 8Hz. As you alter the setting of RV1 the mark-space ratio varies from about 1:20 to about 20:1. If the output is continuously low when RV1 is at one end of the track, the input voltage has fallen below 200mV, which is the minimum required. This may happen if resistors are not quite of their specified values. To compensate, increase R4. If the output of IC2 stays 'high' at the other end of the track, the analogue voltage is exceeding all values of  $V_{ramp}$ . Increase the value of R3. Without an oscilloscope, the outputs can be detected by connecting a crystal earphone (e.g. from a tape-recorder) to the output pins and the 0v line, with a 100nF (approx.) capacitor in series. You should hear a high-pitched note when it is connected to IC1 and a low buzzing sound when connected to IC2.

After completing construction, test for short-circuits between the lines which are to be connected to the decoder. Then plug the control into the decoder. To test its output on the micro, you need a program which reads input repeatedly and displays it. This program is for the Spectrum:

```
10 LET x = IN 31
20 PRINT x
30 GO TO 10
```

As the program runs, a succession of values appear on the screen, changing regularly from 254 to 255. With an equivalent program on the Ace, you may get 32 and 33. If the unused data lines are floating, other values may appear. It is better to eliminate these by the AND operation:

```
: TEST 31 IN 1 AND . ;
```

TEST gives 1 when the output from the circuit is 'high' and 0 when it is 'low'. If you place this in a loop:

```
: WAVE 200 0 DO TEST LOOP;
```

You will see a series of 1s and 0s changing regularly in response to the waveform from IC2. The proportion of 1s to 0s varies

with the setting of RV1. If you ever get all 1s or all 0s, the input voltage is swinging out of range and the value of R3 or R4 should be altered, as explained earlier.

## Programming

The most frequent use for the control is to move 'bats' or other objects around the screen. There are two ways of assessing the output from the circuit. One method is for the micro to measure the length of time for which the output is 'high'. This is the *timing* method. To do this accurately requires that the micro should be working quickly, so a machine-code program is virtually essential. We shall return to a discussion of this method later.

The other method is to read the output a fixed number of times and count how many of these readings are 1. This is the *sampling* method. Its result is related directly to the mark-space ratio. It is necessary to take at least 100 readings to get a reasonably reliable result. Readings must extend over several waveforms in order to get a fair sampling of the relative times spent in the 'high' state and in the 'low' state. This means that the method tends to be rather slow. This BASIC program for the Spectrum uses the sampling method:

```
10 LET x=0
20 FOR j=1 TO 100
30 IF IN 31=255 THEN LET x=x+1
40 NEXT j
50 LET x=x/3
60 CLS
70 PRINT TAB x;"*""
80 GO TO 10
```

This may be easily adapted to the ZX81, using the routine described on p. 175. If necessary, alter the address 31 to that used by your control. The program takes 100 samples in rapid succession. The number found to be 'high' varies from about 5 to about 95, depending on the setting of the control. This is too big a value to be used directly, and in any event is subject to a certain amount of sampling error. This arises

mainly because we do not begin to sample at exactly the same stage in the output sequence of the circuit on each occasion the program is run. Dividing the result by 3 (line 50) gives a value suitable for use with TAB, and at the same time averages out almost all the error. When the program is run the 'bat' (the asterisk) moves across the top of the screen, as you turn the knob of the control.

This program is just a simple one to get you started on using the control. Your next step is to incorporate versions of this into your games programs.

The program given above is slow. If you try to speed it up by taking fewer samples, the sampling is less reliable and the 'bat' jumps around like a tennis-player waiting to receive a fast service. Perhaps this adds realism to the game! If you want a faster yet still reliable action, use this machine-code program, which is based on the timing method:

06 00	LD B, 0	reset register B
0E 00	LD C, 0	reset register C
DB 1F	A:INA,(1F)	read address 1F to acc.
3C	INC A	increment accumulator*
20 FB	JR NZ	jump back to A, if not zero
DB 1F	B:INA, (1F)	read address 1F to acc.
3C	INC A	increment accumulator*
28 FB	JR Z	jump back to B, if zero
0C	C:INC C	counting in register C
20 01	JR NZ	jump to D if C is not zero
04	INC B	carry to register B
DB 1F	D:INA, (1F)	read address 1F to acc.
3C	INC A	increment accumulator*
20 F7	JR NZ	jump to C, if not zero
C9	RET	jump back to BASIC program

The program times the length of a 'high' period by building up a count in registers B and C of the MPU. First these registers are reset to zero. Next, the input is read. As explained above, it is 254 or 255, so the accumulator register of the MPU (microprocessor) now contains one of these values. The accumulator is next incremented (at \*) by 1:

254 (1111 1110) is incremented to 255 (1111 1111)  
255 (1111 1111) is incremented to 0 (0000 0000)

It is just like the mileometer of an old car, which changes from 99999 miles to 00000 miles when it exceeds the maximum that it is designed for. The point about this operation is that when any register changes to zero, the 'zero flag' is set. The next step of the program is to jump back to A if this flag is *not* set, that is, if the input is 'low'. Thus the MPU waits in a loop for as long as the input is 'low'. If you start the program running when output is 'low', it waits until it goes 'high'. If it is 'high' when the program begins, it goes straight on to the next step.

The next step is like the previous one, except that it is now waiting for the input to go 'high', jumping back to B if it is not. The two loops prevent the MPU from beginning counting until the input changes from 'low' to 'high', no matter what state it is in when the program is started. Now it goes around a third loop for as long as input is 'high' but in this loop it increments register C each time round. Register B is incremented each time C changes from 255 to 0. The MPU jumps back to the BASIC program when input finally goes 'low'. It now holds a count in its B and C registers, the size of the count depending on the length of the 'high' period. The USR routine of ZX computers allows you to transfer the contents of the B and C registers to a variable by using a statement such as LET X= USR 23760 with the Spectrum, or LET X= USR 16514 with the ZX81.

As explained on p. 175, a machine-code program may be loaded into the memory previously occupied by a REM line. The program below can be used either with the Spectrum or the ZX81, except that you have to substitute 16514 for 23760 when using the ZX81. Since the machine-code program is fairly long, we simply put it all into a DATA statement and then use a loop to POKE it into memory:

```
10 REM this is for the machine-code
20 FOR j=0 TO 23
30 READ x
```

```

40 POKE 23760+j,x
50 NEXT j
60 DATA 6, 0, 14, 0, 219, 31, 60, 32, 251, 219, 31,
    60, 40, 251, 12, 32, 1, 4, 219, 31, 60, 32, 247,
    201
70 LET x=USR 23760
80 IF x<10 THEN GO TO 70
90 LET x= INT (x/250)
100 CLS
110 PRINT TAB x;"*"
120 GO TO 70

```

If you are not using address 31, substitute the correct value three times in line 60, in place of all the 31s. When you have run this program once, the whole of the text in the REM will have been replaced by various symbols, for that section of memory now holds the machine-code program. You can then delete lines 20 to 60 in order to make space for your games program. Running this program as it stands produces the same result as the earlier program, but it is much faster. If you find that the 'bat' does not move far enough to the right, reduce the value of the divisor in line 90. This fast program occasionally picks up very brief and spurious 'high' pulses coming from the IC, and returns with a count of 1 or 2. Line 80 is there so that such pulses are ignored.

A machine-code program similar to the above would work on the Jupiter Ace too, for this also has a Z80 as its MPU. You need to change the INC A codes to AND 1. The code for this operation is E6 01. Enter this in place of 60 at three points in the program. The program may be placed in the parameter field of a word, as described on p.147 on the Ace's manual.

FORTH runs so much faster than BASIC that there is little to gain by using machine-code. Here are some FORTH words which will display a moving 'bat' on the Ace, using the timing method:

```

: HIGH? BEGIN 31 IN 1 AND UNTIL ;
: LOW? BEGIN 31 IN 1 AND WHILE 1+ REPEAT ;
: BAT INVIS 4 HIGH? LOW? SWAP / 0 SWAP CLS
  AT ." *" ;

```

: PLAY 200 0 DO BAT LOOP ;

HIGH uses BEGIN . . . UNTIL, so waits until the input condition is true (=1). By contrast, LOW? uses BEGIN . . . WHILE, so waits until input is false (=0). In the word BAT, HIGH? waits for input to become 'high'. Then LOW? waits for it to become 'low' again, and increments the top of stack while doing so. Then the count is divided by 4, and the 'bat' is displayed on the top line of the screen at the corresponding position. Note that a low result is obtained if input is 'high' when the program is started. After that, the program keeps pace with the changing input. If the 'bat' moves too little, change the 4 of the BAT definition to 3. If it moves too much, change it to 5. The definition could be adapted to floating-point numbers to obtain the required range of the 'bat' with greater precision.

### *PARTS REQUIRED for the GAMES CONTROL*

#### *Resistors (carbon, 0.25W, 5% tolerance)*

R1        220k  
R2, R5   100k (2 off)  
R3, R4   56k (2 off)  
RV1      100k variable potentiometer

#### *Capacitor*

C1        47n polyester

#### *Integrated Circuits*

IC1       CD4011 CMOS quadruple 2-input NAND gate  
IC2       507C voltage-to-timer converter

#### *Miscellaneous*

Case suitable for mounting control potentiometer  
Knob for RV1, suitable for use in games  
Circuit board  
1mm terminal pins (6 off)  
8-pin IC socket  
14-pin IC socket  
3-way socket to fit 3-way plug of decoder board



## **Project 12**

### **RAIN DETECTOR**

The remaining projects in the book make up a weather station series. They can all be plugged into the decoder and operated together to record certain aspects of the weather. These are easy projects and are not precision instruments, but they give enough information to allow the micro to try to predict what the future weather is likely to be. Although these last six projects are concerned with meteorology, many of them have several other useful applications.

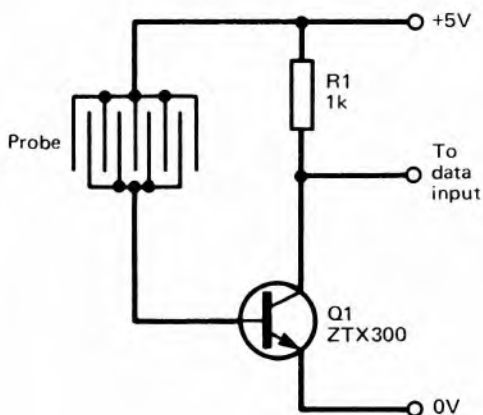
This project is the simplest in the whole book. It tells you when it is raining. Not only does this allow the micro to keep a record of the number of rainy days, or the number and duration of rain showers, but it has other uses around the home. On washdays it is handy for detecting when it starts to rain. If the Bleeper (Project 5) is connected at the same time, and is located in the kitchen (or beside the TV set?), the micro can sound a warning that the washing must be taken in. The project also detects when water rises above a given level. In conjunction with the micro it can warn you when the bath is overflowing, or when the well is running dry.

#### **How it works**

The base current to the transistor (Q1, Fig.12.1) comes by way of a probe. This consists of strips of conducting material placed close together. Their distance apart is such that raindrops are able to bridge the gap. A single raindrop will allow sufficient conduction to occur to turn on the transistor. The result of this is that the level at the Data Input falls from 'high' to 'low'. Fig.12.2 shows an easy way to make the probe from a scrap of strip-board.

#### **Building it**

Two small scraps of strip-board are required, one for the

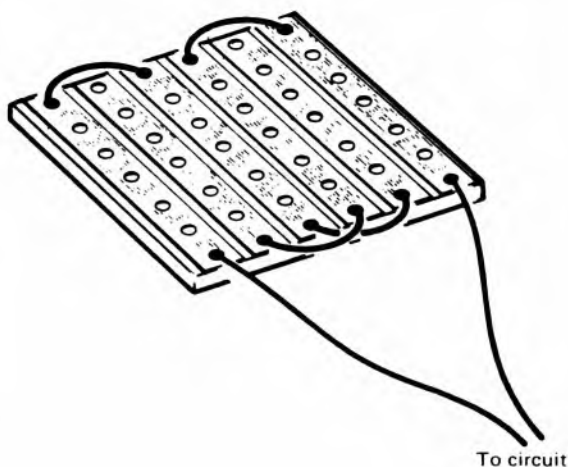


*Fig. 12.1 The circuit diagram of the rain detector*

circuit and one for the probe. It is in fact possible to build the whole device on one piece of board, covering the area on which the components are placed and leaving the rain-detecting area exposed. This circuit gives a DC output, so it is in order to have a long wire connecting the output to the Data Input of the decoder board.

Test the circuit by connecting a voltmeter to its output. It should read +5V. When you let a drop of water fall on the probe, the voltage should fall close to 0V. The same effect can be obtained by touching the probe with a moistened finger, so this device could also have applications as a touch-control switch.

The probe should be mounted in an exposed position outdoors, so that buildings or trees do not protect it from driving rain coming from certain directions. It is best to slope it



*Fig. 12.2 The probe for the rain detector*

slightly, so that rain drains away when the shower is finished, and the probe dries quickly.

## Programming

All that is needed is a program line which reads the input from the project at regular intervals. If it is raining, the input on the Spectrum is 254 and if it is dry the input is 255. On the ZX81, it is 65534 and 65535. With the Ace you will obtain 32 or 33, though you may prefer to AND the input reading with 1 (see p. 10) so as to obtain 0 for rain and 1 for dry.

## *PARTS REQUIRED for the RAIN DETECTOR*

### *Resistor*

R1    1k

### *Semiconductor*

Q1    ZTX300 or almost any npn transistor

### *Miscellaneous*

Stripboard for circuit and probe (small scrap)

3-way socket to fit 3-way plug on decoder board

Connecting wire

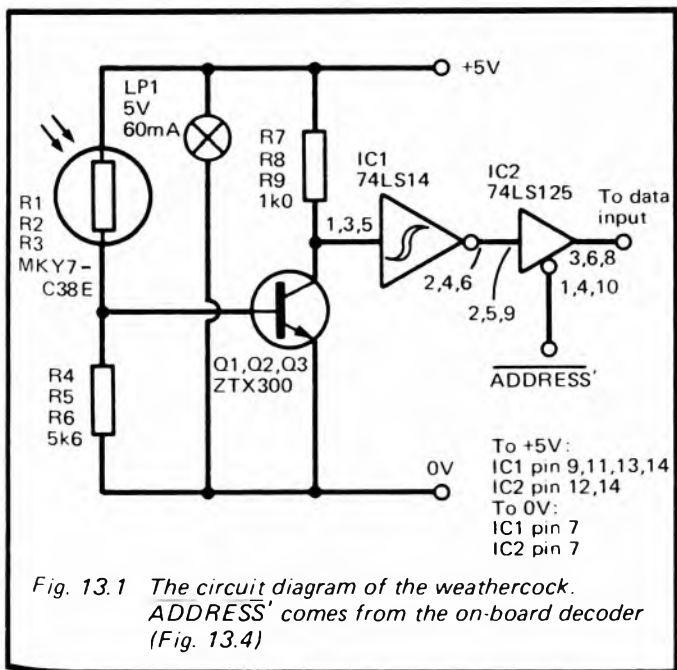
## Project 13

### WEATHERCOCK

Knowing the direction of the wind is an essential requirement for weather forecasting. This project reads wind direction and passes the information to the micro. It is also interesting as an example of the way in which the micro can be informed of a *position* which is being measured as an *angle*.

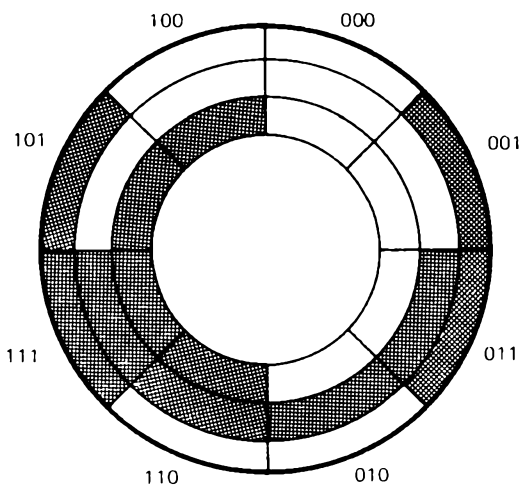
#### How it works

The position of the weather vane is sensed by three light-dependent resistors (LDRs). Fig.13.1 shows the circuit for one

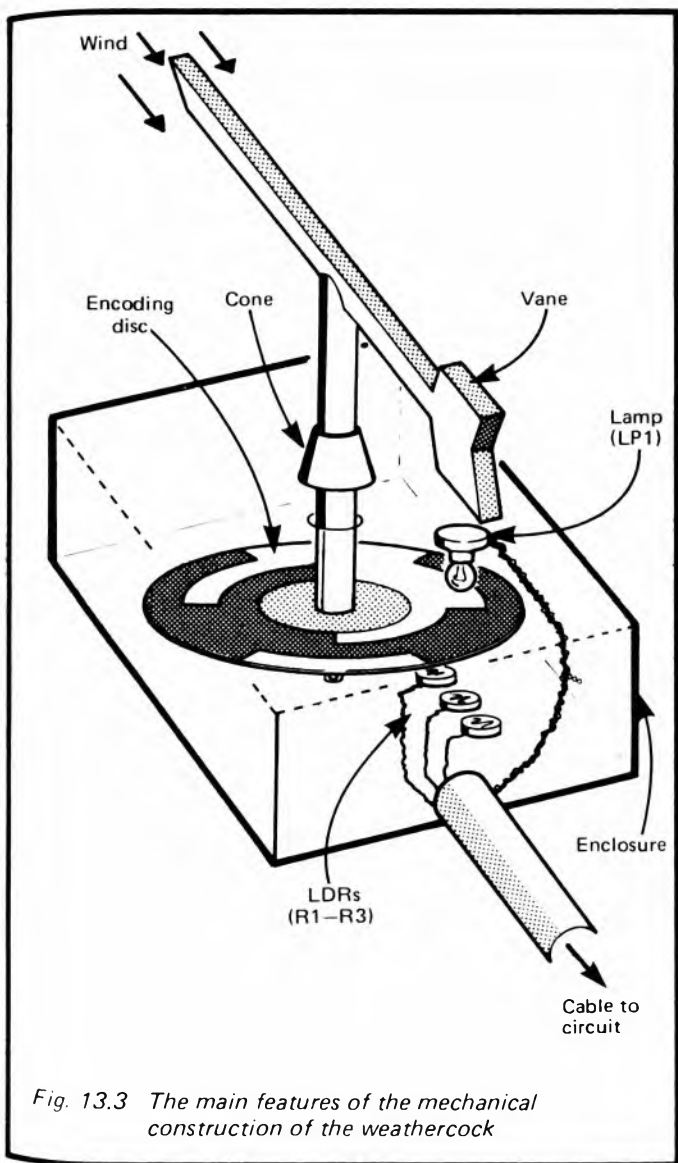


of these. The LDRs receive light which comes from a lamp and passed through an encoder disk (Figs.13.2 and 13.3). This disk is fixed to the shaft of the weather vane. The disk is transparent but has a pattern of black sectors on it. In any given position of the vane (and hence of the disk), each one of the LDRs is either shaded or unshaded. As the wind direction changes, the vane and the disk rotate. Some LDRs become exposed to the light from the lamp and some become cut off.

A closer look at the disk (Fig.13.2) shows that the pattern of the black sectors does *NOT* represent a sequence of binary numbers. Reading clockwise and converting to decimal, we find: 0, 1, 3, 2, 6, 7, 5, 4. This is what is known as "Gray Code". The feature of this sequence is that as we go from one number to the next (or return to the first number from the last number) only one digit changes each time. To see why this



*Fig. 13.2 Marking out the encoding disc for the weathercock*



*Fig. 13.3 The main features of the mechanical construction of the weathercock*

is important let us look at what would happen if we began the sequence with 0, 1, 2, . . . . The first change would be:

0 0 0  
changing to 0 0 1

This creates no problems and this change is found in the Gray sequence. But the second change could be troublesome:

0 0 1  
changing to 0 1 0

This would be all right if both digits changed at exactly the same instant. In a piece of equipment such as this it is very unlikely that the LDRs are so precisely arranged and their resistances are so exactly matched that each will respond in an identical manner. They will *not* all change at the same instant. If the middle digit changes before the right-hand digit, we get an intermediate stage showing 011. We get the same if the right-hand digit changes first. Instead of getting 0, 1, 2, . . . as the disk rotates, we get 0, 1, 3, 2 . . . . The 3 is only there briefly, but there would be time for the micro to take a reading and obtain a false result. The Gray code overcomes this problem, since only one digit changes each time.

## Building the vane

Fig.13.3 shows the general features of a typical Weathercock. Exactly how you construct yours depends on what materials and tools you have available and on your skill in making such a device. It is often possible to adapt a ready-made weathercock. Such instruments are obtainable cheaply from stores selling educational equipment. Often they are made from plastic and while these cheaper models are generally less durable than those intended for serious use, they are usually easier to adapt. The disk is made from clear transparent plastic. A sheet of thick acetate film can be used, and may be painted by using a black marker pen (felt-tip spirit pen). Alternatively, the sectors may be cut out from black PVC insulating tape and stuck on the disk. In order that the readings indicate the major points of the compass (N, NE, E, SE etc), the LDRs are placed so



that the *centre* of each sector lies over the line of LDRs when the vane is aligned with the main compass points.

For clarity, Fig.13.3 shows the disk several centimetres above the LDRs, but it is much better if the gap is small and the disk almost touches them. The lamp should be mounted a few centimetres above the disk, so that its light spreads fairly equally to all three LDRs. Leads are taken from each LDR back to the main circuit which is to be housed in a small case near to the micro.

Fig.13.3 shows a way of preventing rainwater from running down the shaft into the mechanism. The cone deflects it on to the top surface of the light-proof box. It is not essential for the hole where the shaft enters the enclosure to be absolutely light-proofed, though a collar around the shaft helps to prevent light from scattering toward the LDRs.

The circuit uses three lines of the data bus (lines D0 to D2). IC1 has three-state outputs (p. 160) which are enabled by a low from the address decoder circuit.

## Addressing

As explained for Project 2 (p. 21), the addressing of this project is not completely provided for by the decoder (p. 158). If you want to connect only this project to the micro, you can do as suggested on p. 21: wire the ADDRESS output terminal of the decoder (Fig.D.1) to the ADDRESS' input terminal of the Weathercock detector (Fig.13.1). It can then be addressed using *any one* of the addresses in Table D.2.

If you would like to have other projects connected at the same time as this project, you need to add a decoder to this project. You will probably want to have several weather-station projects working at once, so make sure they all are at different addresses. To keep the wiring as simple as possible, the decoder uses only lines A5 and A6 (Fig.13.4). As explained on p. 21, it responds to two addresses, which are 5F and DF or 7F and FF.

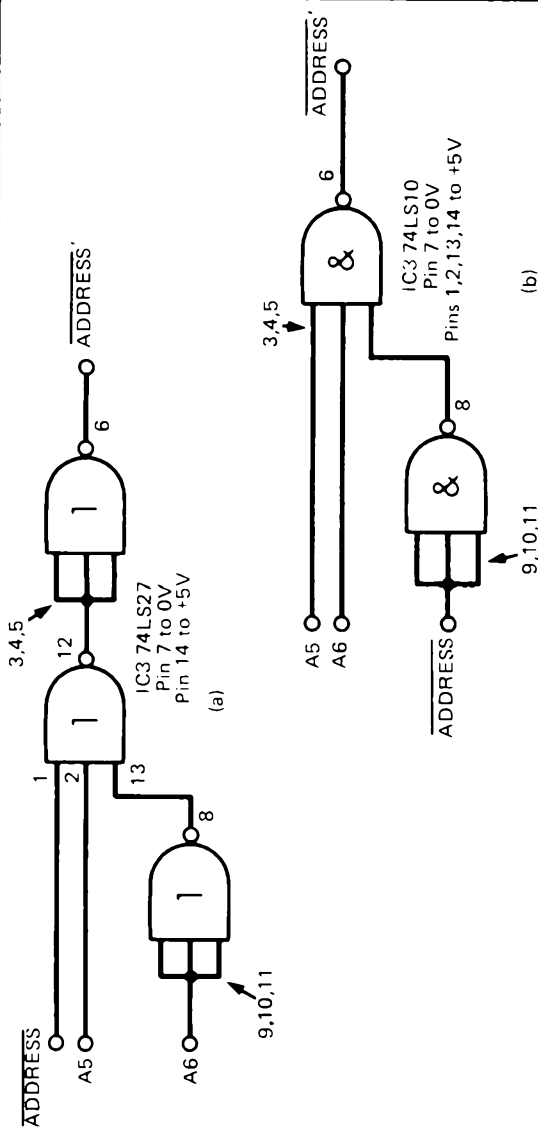


Fig. 13.4 Alternative address decoders,  
(a) uses addresses 5F or DF, (b) uses addresses 7F or FF

## Building the circuit

The device is to be plugged on to one of the 10-way plugs of the decoder (p. 172). It requires 9 lines: the two power lines, three data lines, 2 address lines and the  $\overline{\text{ADDRESS}}$  line. It also wires from the negative ends of the three LDRs. The type of LDR recommended is small, but any LDR of small diameter can be used instead. Larger types *can* be used if these are the only ones available, but to use these requires that the disk must also be made larger. If the LDRs are not of the type specified and have a higher or lower resistance range, it may be necessary to substitute resistors of a different value for R4–R6.

It is best to build and test the circuit for one LDR first. It is then easy to check that the correct resistance is being used for R4–R6, before proceeding with the remainder of the construction. When all is finished, temporarily wire the buffer enable input (IC2, pins 1, 4 and 10) to 0V to enable the outputs. Slowly rotate the disk to measure the output of each buffer with a voltmeter. Fig.13.2 shows what results to expect, where 0 represents a 'low' output (less than 0.8V) and 1 represents a 'high' output (more than 2V). Assuming that all is correct, test all wires going to the decoder board to check that there are no short-circuits between any pair of them.

## Programming

All that is required is to read the data input. This circuit leaves the top 5 data lines unconnected, so they each take the value 1. With zero input from the Weathercock, the data bus reads 1111 1000, which gives 248 in decimal on the Spectrum. A simple program such as:

```
10 LET x= IN 95
20 PRINT x
```

results in a number between 248 and 255 being displayed. You then have to write 8 program lines to interpret this reading. This is one of them:

```
30 IF x=248 THEN LET wind$="North"
```

On the Ace, it is best to AND the input with 7, so as to obtain a number between 0 and 7. A suitable word definition is:

```
: WIND? 95 IN 7 AND . ;
```

In the listings above we are assuming that the weathercock is addressed at 5F (hexadecimal, or 95 in decimal). If you are using another address modify the lines accordingly.

You may be able to use the wind direction to predict the weather. What the prediction will be depends very much on your locality. It may also depend on other features of the current weather measured by some of the other projects. As well as reading direction, you can use the weathercock to detect changes in the wind direction over a period of several hours. If the wind is veering, that is to say, changing direction in a clockwise direction, possibly indicates an approaching depression, with the likelihood of low cloud and rain.

Another aspect is the steadiness of wind direction. If the weathercock is read every minute and it is found that the direction is changing frequently, this can be another clue to what future weather is likely to be. There are several books on weather which will help you work out your predictions. You may also be able to program the micro to work these out for you.

### *PARTS REQUIRED for the WEATHERCOCK*

*Resistors* (carbon, 0.25W, 5% tolerance, except R1-R3)

R1-R3 MKY7C38E or similar light-dependent resistor  
(preferably small) (3 off)

R4-R6 5k6 (but see text) (3 off)

R7-R9 1k0 (3 off)

*Semiconductors*

Q1-Q3 ZTX300 or similar npn transistor (3 off)

*Integrated Circuit*

IC1 74LS14 Hex Schmitt trigger

IC2 74LS125 quadruple bus buffer gate with three  
state output

### *Miscellaneous*

LP1        5V 60mA filament lamp

Socket for LP1

14-pin IC sockets (2 off, if on-board decoder used)

10-way socket to fit 10-way plug on decoder

Circuit board

1mm terminal pins (14 off)

Materials for the mechanical parts

### *Parts for Alternative Address Decoders*

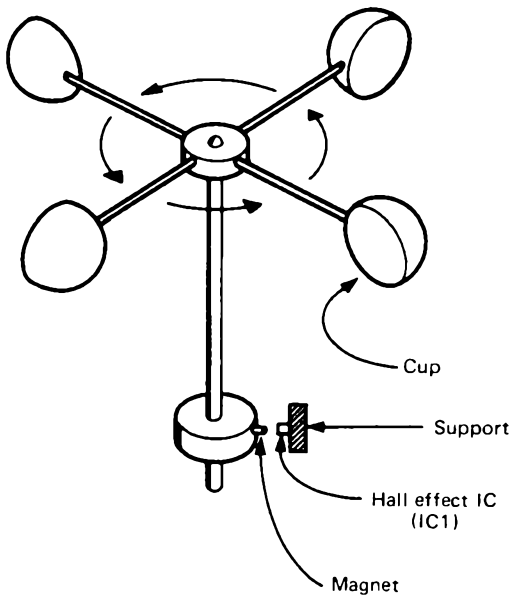
(a) IC3    74LS27 triple three input NOR gate

(b) IC3    74LS10 triple three input NAND gate

## Project 14

### ANEMOMETER

The anemometer measures wind speed. It has a set of three or four cups arranged on top of a shaft (Fig.14.1) so that they spin round when the wind blows. It is assumed that the cups travel at a speed close to that of the wind itself so, knowing



*Fig. 14.1 The main features of the mechanical construction of the anemometer*

the radius of the assembly and the rate of rotation, we estimate wind speed. In effect, the circuit of this project is a tachometer, a circuit to measure *rate of rotation*. The circuit could be adapted to measure the rate of rotation of other machinery.

## How it works

Figure 14.1 shows that there is a wheel at the base of the shaft of the anemometer. This rotates at the same rate as the cups. The wheel is made from non-magnetic material such as plastic or wood. It carries a small permanent bar magnet at one point on its circumference. The magnet is arranged with one of its poles protruding slightly. As the wheel rotates, the magnet is carried past an IC which is mounted to one side of the rotating assembly. This is a Hall Effect IC. The Hall Effect occurs when a piece of semiconducting material is placed in a magnetic field. Those who remember learning Fleming's Left Hand Rule in physics lessons will know that there is an interaction between a current and a magnetic field. In Fig.14.3 a current is passed along a slice of semiconductor. The moving electrons are deflected to one side when a strong magnetic field is present. The result is a potential difference between the two sides of the conductor. This is detected and amplified by circuits within the IC. The outputs of IC1 are normally at about 2V. When a magnet is held close to the IC1 the voltage at one output rises slightly and the other falls. The operational amplifier IC2 compares these voltages. When they are equal (or nearly equal) its output is 'low'. When the voltages are unequal (i.e. when a magnet is near), the output of IC2 rises sharply to +5V. This quickly switches on Q1. The voltage at the input of the counter (IC3) falls sharply, triggering the counter. Thus the counter is incremented every time the magnet passes close to IC1.

IC3 is wired as a divide-by-16 counter, so its D output changes at one-sixteenth of the rate at which the anemometer is rotating. This rate is measured by the micro, which then calculates the wind speed.

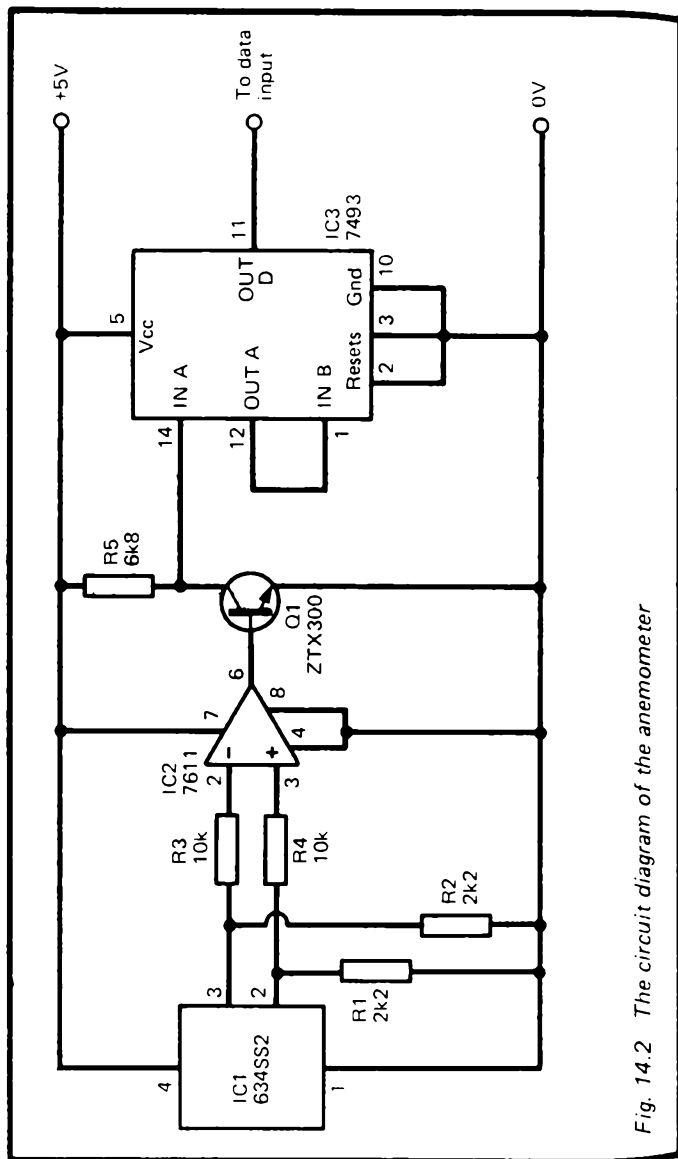


Fig. 14.2 The circuit diagram of the anemometer



## Building it

Unlike most of the interfaces in this book, the whole circuit is best located close to the sensor. The output from IC3 is a slowly changing one, making it suitable for transmission along a relatively lengthy line to a Data Input of the decoder. The circuit requires three connections to the decoder: the two power lines and a wire to a Data Input.

The circuit is to be housed in a weatherproof case. It may be convenient to mount IC1 on the outside of the case. Leads to this should be covered with melted paraffin wax or a weather-proofing and water-proofing compound to eliminate the possibility of short circuits.

The anemometer assembly is easy to build. Empty food cartons may be used for the cups and the cross-pieces can be made from stiff wire. The main essential is that the assembly rotates freely. Its diameter should be about 30cm or possibly a little more. The wheel which carries the magnet can be cut from wood, plastic or a large cork stopper. A suitable magnet is usually sold with the Hall Effect IC. Take care when mounting the magnet and the IC that there is no danger of the two coming into contact as the assembly rotates, particularly in high winds. On the other hand, the end of the magnet needs to pass within about 1mm of the centre of the IC in order to generate a sufficiently great voltage difference.

Build the entire circuit, then connect it to a +5V power supply for testing. Connect a voltmeter to one of the outputs of IC1. Rotate the anemometer slowly and note the rise or fall of voltage as the magnet passes the IC. Now transfer the voltmeter to the output of IC2. The output should be 0V, but rises sharply to +5V whenever the magnet passes IC1. If this does not happen, the polarity of the magnet is wrong; remove it from the wheel and replace it the other way round. Connect the voltmeter to the D output of IC3. The output should rise to 'high', fall to 'low' and rise to 'high' once for every 16 rotations of the anemometer.

The output from D changes at a slow rate so the cable between the interface and the decoder board can be several metres long. This allows the device to be mounted as high as

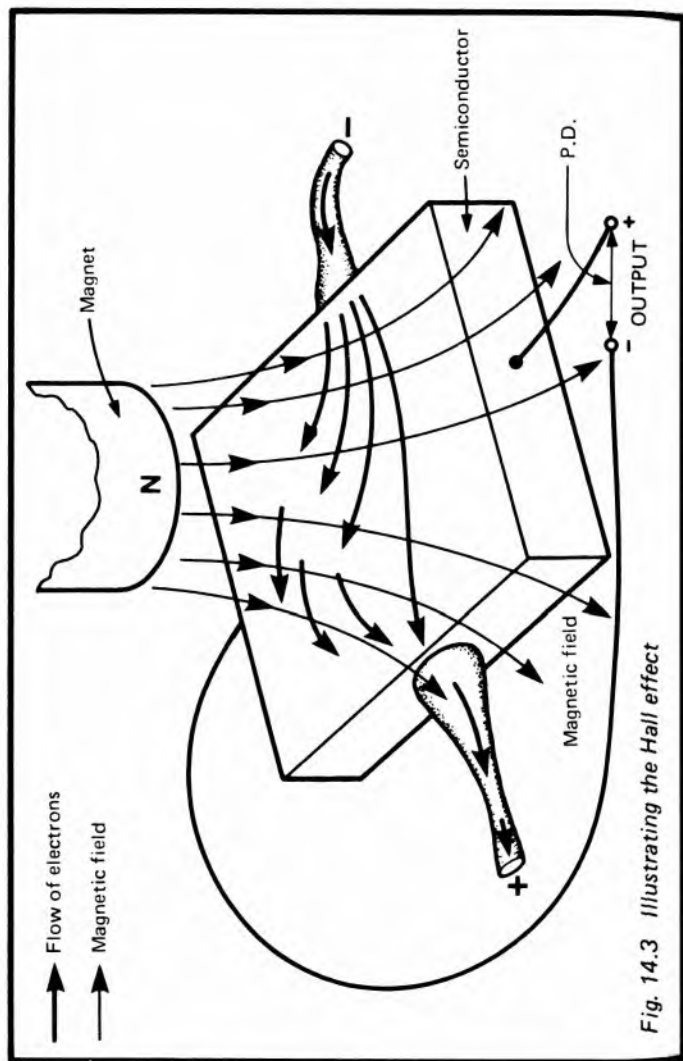


Fig. 14.3 Illustrating the Hall effect

possible above ground so as to expose it to the full speed of the wind.

## Programming

We require a program which measures the length of pulses coming from the D output. The machine-code program on p. 107 is suitable for use with the ZX81 and Spectrum. However, the pulse rate is slow, so a BASIC program will generally be adequate. With the Ace, use the word definitions on p. 109.

By turning the anemometer by hand at a known rate, it is possible to relate the value obtained to a given number of revolutions per second. Turn the anemometer by hand (about 1 revolution per second) for 20 or so revolutions. Count the number of revolutions exactly and also measure the time taken. Calculate the rate in revolutions per second. As the same time run the timing program to see what value is obtained. This gives the figure which corresponds to the rate of turning. Repeat this several times to obtain an average result for use in calculations later. Remember that the figure obtained is inversely related to the rate of rotation. For example, doubling the rotation rate halves the figures.

Assuming that the cups of the anemometer are moving with the same speed as the wind, the wind speed is:

$$S = 2 \times \pi r \times R \times 3600 \div 100000$$

Where  $S$  is the wind speed in kilometres per hour,  $R$  is the rate of rotation in revolutions per second, and  $r$  is the radius of the anemometer assembly in centimetres. The symbol  $\pi$  has its usual value 3.1412, so the equation can be simplified to:

$$S = 0.226 \times r \times R$$

The corresponding equation for British units is:

$$S = 0.357 \times r \times R$$

where  $S$  is in miles per hour,  $R$  is revolutions per second and  $r$  is in inches.

Either of these equations can be used in a program to calculate the wind speed, given the rate of rotation. For

example, if the interface is giving 1 count per second, this corresponds to a rotation rate (R) of 16 revolutions per second. If the radius of the anemometer is 15cm, the wind speed is:

$$S = 0.226 \times 15 \times 16 = 54 \text{ km/h}$$

With a similar anemometer with radius of 6 inches, the wind speed is:

$$S = 0.357 \times 6 \times 16 = 34 \text{ mph}$$

The anemometer circuit can easily cope with rotations of double this rate so can register winds of 100km/h or more.

### *PARTS REQUIRED for the ANEMOMETER*

#### *Resistors (carbon, 0.25W, 5% tolerance)*

R1, R2 2k2 (2 off)

R3, R4 10k (2 off)

R5 6k8

#### *Semiconductors*

Q1 ZTX300 or similar npn transistor

#### *Integrated Circuits*

IC1 634SS2 Hall Effect IC (complete with magnet)

IC2 7611 CMOS operational amplifier

IC3 7493 4-bit binary counter/divider

#### *Miscellaneous*

Circuit board

8-pin IC sockets (2 off, including one for IC1)

14-pin IC socket

1mm terminal pins (7 off)

3-way socket to fit 3-way plug of decoder board

Materials for making anemometer assembly

Connecting wire

## **Project 15**

### **THERMOMETER**

Although this project is part of the weather-station series, it has many other applications. It may be used indoors to measure room temperatures, or perhaps the temperature of photographic solutions. It can be used to detect excessive temperature as part of a domestic fire alarm system.

#### **How it works**

The circuit works in much the same way as that of the Games Control (Project 11). Instead of the variable potentiometer used in that circuit we have a variable resistor of another kind (fig.15.1). This is a thermistor, the resistance of which varies significantly with temperature. The thermistor used in this circuit (Fig.15.1) has what is known as negative temperature coefficient, which means that, over its usual working range, its resistance decreases as its temperature increases. An increasing temperature causes the resistance of the thermistor to fall, resulting in a falling potential at point A and a falling analogue voltage at the input of IC2. As explained under project 11, the waveform of the output of IC changes (see Fig.11.2), and the change can be measured by the micro.

#### **Building it**

Normally the thermistor is located outdoors. For use in weather recording, the best place is inside a meteorological screen. This shades it from direct sunlight yet allows air to circulate freely around it. If you do not have such a screen, mount it in a place where direct sunlight can not reach it and where air circulation is good. It should be several centimetres away from any surface such as a wall or fence, otherwise it may take the temperature of the wall or fence, instead of that of the air. If it is in the open, protect it from rain, for this may cause a partial short-circuit, and readings will be incorrect. One

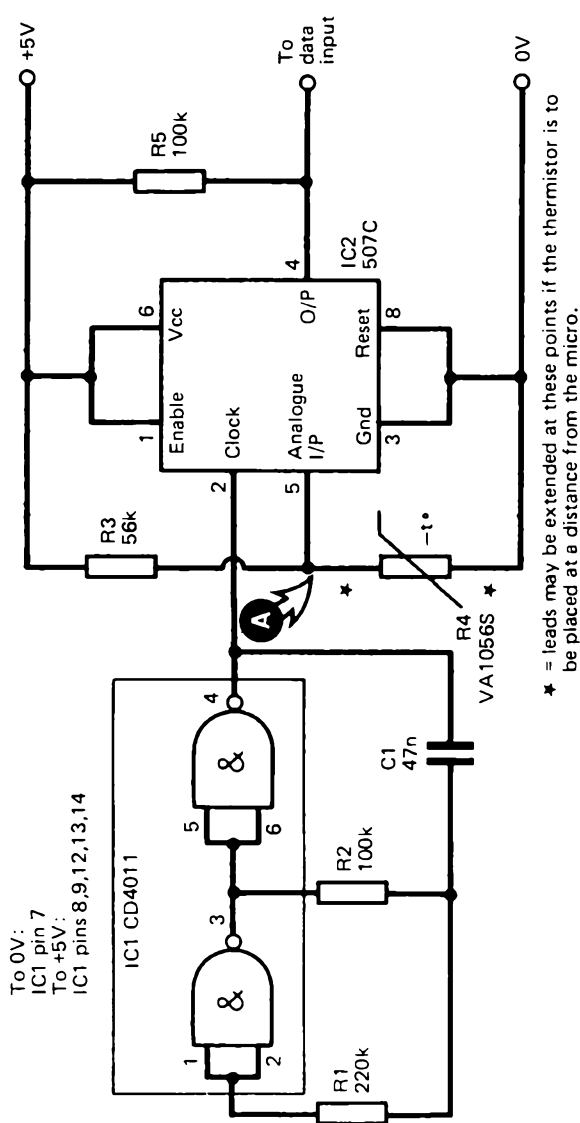


Fig. 15.1 The circuit diagram of the thermometer

method of waterproofing is to melt some paraffin wax and to dip the thermometer in the wax to coat it. Another method is to mix a quantity of epoxy resin adhesive (e.g. Araldite) and coat it with this.

The rest of the circuit is built in a small case, located close to the micro. It needs three wires to connect it to the micro: the two power lines and a wire to a Data Input. The circuit does not require any address decoding except that provided by the Decoder.

When the circuit is ready, it may be tested in the same way as the Games Control (see pp. 104–6). You can try the effects of various temperatures by immersing the thermistor in warm water or in water containing melting ice cubes. In such extremes the output of IC2 should never be continuously 'high' or 'low'. If this condition is found, replace R3 by a resistor of different value, or possibly wire an additional resistor between R4 (the thermistor) and the 0V line. Exactly what you may need to do depends on the characteristics of the thermistor you are using and the range of temperature over which you require it to operate.

## Calibration and programming

The simplest method of programming is to read the output from the circuit many times and count how many 'high's are obtained. This is the sampling method referred to on p. 106. With the ZX81 and Spectrum we may use a very similar program to that given there, except that it is feasible to take many more samples to obtain a more reliable result. Line 20 might be amended to take, say, 500 samples instead of only 100. The additional time required for this does not matter in this application. Lines 50–70 are not applicable, though something similar could be employed to plot a graphics 'thermometer' on the screen. Instead, type in this line: 50 PRINT x.

Having obtained a value x which varies in proportion to temperature, the next step is to calibrate the thermometer. Place the thermistor in a glass of water containing melting ice-cubes. Stir occasionally, and allow at least 5 minutes for

the thermistor to take up the temperature of the water. This is close enough to 0°C (32°F).

Run the program several times and find an average value for  $x$ . Results should be in the region of 20 to 50. If the count still seems to be decreasing, wait a while longer, for the thermistor has not cooled fully. If the ice cubes melt completely, replace them. Now repeat the procedure, but using a glass full of warm water at say 30°C (or say 90°F). Choose a temperature which is a little above the upper end of the range which you want to measure. You will need a thermometer to find out what the exact temperature is. Readings should now be at the upper end of the range, over 300 at least.

If the difference between the lowest and highest readings is too small you will not be able to obtain an accurate result. In this event you may need to replace R3, or wire in an additional resistor, as already mentioned.

Let us suppose that you obtained a reading of  $x_1$  at 0°C and  $x_2$  at  $T$ °C. The formula for converting the reading  $x$  at any temperature  $t$  is:

$$t = (x - x_1) \times \frac{T}{x_2 - x_1}$$

For example, if the count is 40 at 0°C and 420 at 30°C, then a count of 150 is equivalent to a temperature of:

$$\begin{aligned} t &= (150 - 40) \times \frac{30}{420 - 40} \\ &= 90 \times 30/380 = 7.1^\circ\text{C} \end{aligned}$$

The same formula may be used on the Fahrenheit scale, except that having calculated  $t$ , you add 32 degrees to it.

As a program line, given the values mentioned above, the formula becomes:

$$60 \text{ LET } t = (x - 40) * 30/380$$

On the Ace, we can use a set of words similar to those used with the Game Control:



```

: HIGH? BEGIN 31 IN 1 AND UNTIL ;
: LOW? BEGIN 31 IN 1 AND WHILE REPEAT ;
: COUNT HIGH? LOW? 0 500 1 DO 31 IN 1 AND
  IF 1+ THEN LOOP ,
: TEMP 30 COUNT 33 - * 302 / . ;

```

'HIGH' is as given before, but LOW? does not act as a counter. The word COUNT does this but waits for output to go 'high' and then 'low' before it starts. It then takes 500 sample readings and counts those which are high. This extends over several output pulses from the circuit. It finishes with this count on top of stack. TEMP performs the calculation. This definition is based on a calibration count of 33 at 0°C and 335 at 30°C. (N.B.  $302 = 335 - 33$ ). This word prints out the temperature as an integer. You could adapt the definition to work in floating-point so as to obtain a more precise result.

As well as reading temperature at a given instant, the micro can be programmed to take readings at regular intervals and record them in memory. They could then be displayed daily or on demand and a table could be printed out. If the readings are taken frequently enough it is easy to program the micro to print out the maximum and minimum daily temperatures too.

## *PARTS REQUIRED for the THERMOMETER*

*Resistors* (carbon, 0.25W, 5% tolerance)

R1	220k
R2, R5	100k (2 off)
R3	56k
R4	VA1056S rod thermistor, or any negative temperature coefficient thermistor having resistance about 47k at 25°C (disc and bead types equally suitable)

*Capacitor*

C1	47n polyester
----	---------------

*Integrated Circuits*

IC1	CD4011 CMOS quadruple 2-input NAND gate
IC2	507C voltage-to-time converter

### *Miscellaneous*

Suitable housing for thermistor

Case for circuit

Circuit board

1mm terminal pins (5 off)

8-pin IC socket

14-pin IC socket

3-way socket to fit 3-way plug of decoder board

Connecting wire

## Project 16

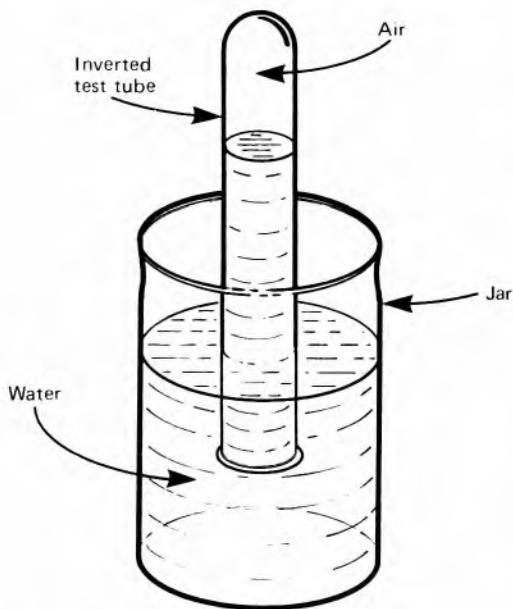
### BAROMETER

A barometer measures the pressure of the atmosphere. It is almost essential for forecasting the weather. The approach of depressions or anticyclones is announced by a fall or rise in pressure. This gives us advance warning of what kind of weather to expect. The rate of change of pressure, tells us how soon we may expect the weather to change. In addition, rapid changes of pressure mean strong winds, while slow changes or no changes at all are associated with calm weather.

Devices for measuring atmospheric pressure electronically are usually very expensive. Indeed, an IC which does this may cost about as much as a ZX81! This project is far cheaper and although it does not give an accurate reading, it is the *changes* which are important. It is much less important to know what the exact pressure is. So this device is a useful one for the fore-caster.

#### How it works

The barometer is based on an extremely simple idea which is often featured in books of science projects for young people (Fig.16.1). The inverted tube contains air. As atmospheric pressure increases, the air inside the tube becomes more compressed and the water rises up inside the tube. When pressure falls again, the air expands and the water level falls. Pressure is measured by measuring the level of the water. Unfortunately this apparatus has one big snag as a barometer. Air expands when heated. If the temperature increases, the water level falls, even though the atmospheric pressure may not have changed. It is possible to allow for the change in temperature by working out how much it would alter the volume of the air. This is too much trouble to be bothered with for the simple barometer of Fig.16.1, but if we make a similar barometer and attach it to a micro, the micro can do all the working out for us!



*Fig. 16.1 A simple 'water barometer'*

In order to compensate for changes in temperature, the micro must know what the temperature is in the region of the barometer. It can find out this if we have the thermometer (project 15) operating at the same time. There is an equation which brings volume, temperature and pressure together:

$$\frac{\text{Pressure} \times \text{Volume}}{\text{Temperature}} = \text{constant}$$

Pressure and volume can be measured in any units we like (provided that we always keep to the same units), while temp

perature must be measured in kelvin. To work out any temperature in kelvin, just take the temperature on the Celsius scale (sometimes wrongly called the Centigrade scale) and add 273.

If we measure pressure, volume and temperature of the air inside the barometer on two separate occasions, and since the calculated result is a constant we can say:

$$\frac{\text{1st pressure} \times \text{1st volume}}{\text{1st temperature}} = \frac{\text{2nd pressure} \times \text{2nd volume}}{\text{2nd temperature}}$$

or, rearranging the above:

$$\begin{aligned} \text{2nd pressure} = \\ \text{1st pressure} \times \frac{\text{1st volume}}{\text{2nd volume}} \times \frac{\text{2nd temperature}}{\text{1st temperature}} \end{aligned}$$

We use an ordinary barometer to measure the 1st pressure when we first take a reading with our home-made barometer. At the same time we measure 1st temperature and the 1st volume. Some time later we (or the micro) measure 2nd volume and 2nd temperature and can then calculate the 2nd pressure. Project 15 measures temperature, while this project measures volume. The micro then calculates the pressure.

In this project, the air is contained in a plastic or glass tube (Fig.16.2). The top end of the tube has a tap to make it airtight. Its bottom end dips into a small jar containing water. The water has been made black by having had Indian Ink added to it. The most extreme changes of pressure and temperature make the water level move over a range of about 2cm.

In the region over which the water level will move, the tube is covered with black insulating tape, except for slits on opposite sides of the tube. Light from a small lamp passes through one slit, across the tube and out through the other slit. It is picked up by a light-dependent resistor (LDR). The resistance of this varies according to the brightness of the light (p. 66). Since the water is black, it cuts off a varying amount of light, depending on its level. In this way, changes in water level are turned into changes in electrical resistance. In the prototype of this Project the LDR resistance was 940ohms when the water was at its lowest level but increased to 1700

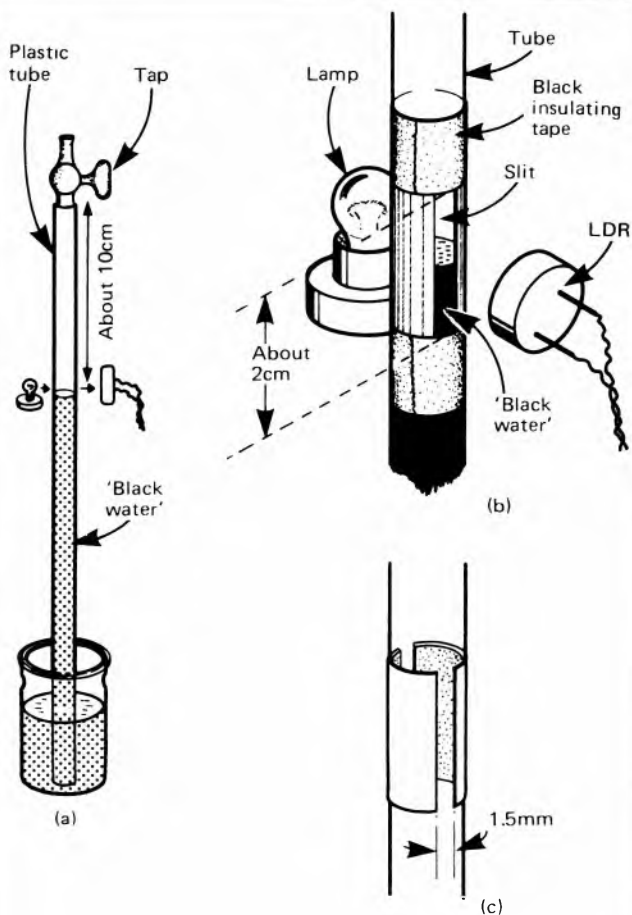


Fig. 16.2 The main features of the barometer. (a) the whole barometer (masking on tube not shown); (b) how the tube is masked. There is an identical slit behind the tube to pass light coming from the lamp. Screening around the lamp and LDR not shown. (c) The first stage in masking

ohms when the water had completely cut off the light. For the same reasons as are given on p. 67, changes in the resistance of the LDR (Fig.16.3) affect the voltage at point A. This changing voltage is fed to IC2, which is a voltage-to-time converter. The way this works is explained on p. 102. The overall effect is that as pressure increases, the volume decreases, the voltage at A increases and the waveform from IC2 spends proportionately more time in the 'high' state. If the micro is programmed to measure this time, the figure obtained is greater.

It is important to realize that the response of this device is far from linear. That is to say, equal increases in pressure do not bring about equal increases in the value found by the micro. However, the figure the micro obtains can be used to assess whether pressure has increased or decreased over the past hours or days, and whether it is changing slowly or quickly.

## Building it

First of all build the barometer itself. There are several ways of doing this. The tube may be of glass, though clear plastic tubing such as is sold for aquarium aerating systems is very suitable. A tap to fit the tubing may also be obtained from a shop specializing in aquaria. The tube may be held straight by stapling it to a strip of wood. This is mounted vertically on a wooden base. The water container can be a small plastic or glass bottle such as is used to hold medicine tablets.

Prepare two strips of black insulating tape about 2cm long and wide enough to leave two gaps about 1.5mm wide when they are fixed on the tube (Fig.16.2c). Wind two other strips around the tube about 2cm apart, as in Fig.16.2b. This leaves two slits about 2 cm long and 1.5 mm wide on opposite sides of the tube. Measure the length of the slit and the distance between the tap and the top of the slit to the nearest millimetre. Mount a small filament lamp to one side of the tube, about 1cm from it. The lamp should be housed in a small box, made from thin white card, to keep light from reaching the LDR directly, and to reflect as much light as possible toward

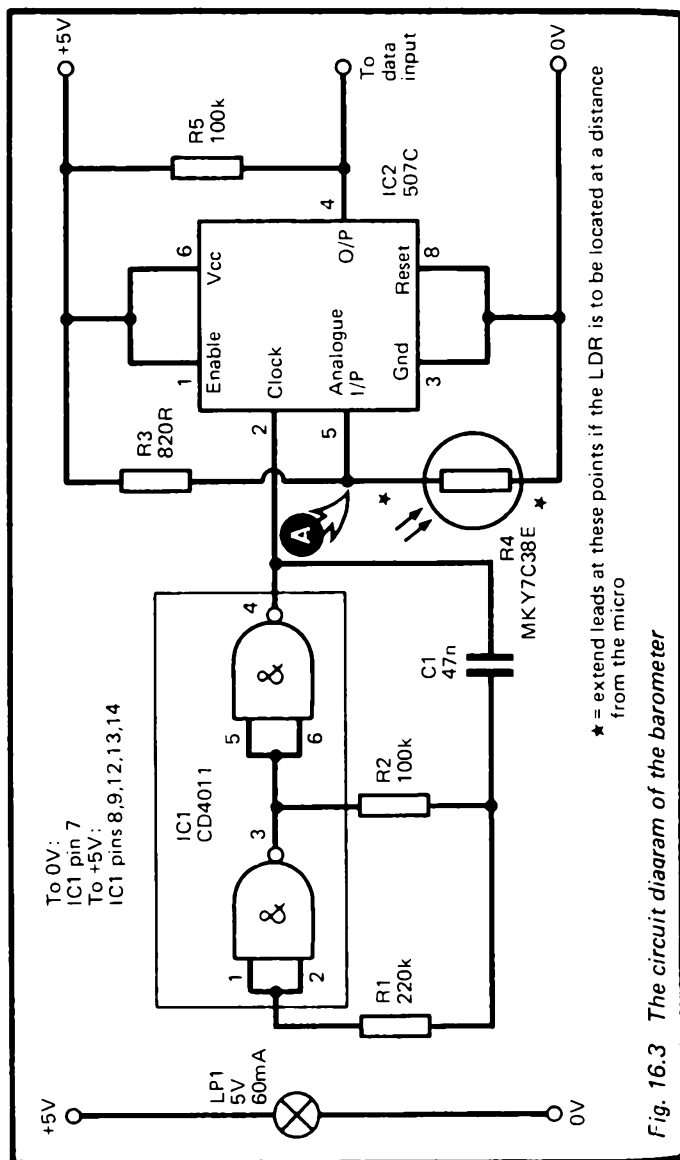


Fig. 16.3 The circuit diagram of the barometer



the slit. On the other side the LDR is enclosed in a similar box made of thin card. It is better if the whole assembly is housed in a light-proof case to prevent daylight from interfering with the readings.

The electronic side of this device requires three connections to the decoder board: the two power lines and a wire to a Data Input. If the barometer is to be placed at some distance from the micro, the leads may be extended at the points marked \* in Fig.16.3. The circuit itself must be reasonably close to the micro. In practice there is much to be said for having the entire apparatus indoors, for atmospheric pressure is more or less the same indoors and out. If it is placed indoors, in a room which is kept at a reasonably steady temperature, there is no need to correct for changes of temperature, and programming is simplified.

When construction is complete, test the leads going to the decoder to make sure that there are no short-circuits between them. Then plug the barometer into the decoder board and switch on the power. Since this device uses the same converter IC as Project 11, a similar program may be used in testing:

```
10 LET x=0
20 FOR j=1 TO 1000
30 IF IN 31=255 THEN LET x=x+1
40 NEXT j
50 PRINT x;" ";
60 GO TO 10
```

Run the program with the tube empty. The result should be reasonably low, perhaps between 200 and 500, but the value obtained depends very much on the type of LDR used, and many other features of construction. However, it should not vary by more than about 2 or 3 each time the program repeats. Take an average of 20 such readings; these are the scale minimum readings.

Now make a mixture of 3 parts of water with 1 part of black ink (Indian Ink preferred). Add one drop of washing-up detergent. Pour the 'black water' into the jar. Open the tap and slowly suck the water up the tube. Take care, or you may find yourself sucking a mouthful of black water! Suck the

water up to a level just above the taped region, so that the slit is completely blacked out. Now run the program again. This gives a series of readings higher than before. The values obtained should not vary by more than 2 or 3. They should be at least 50 or 60 greater than the values obtained with the empty tube. If the increase is smaller than this, try replacing R3 with a resistor of lower value. When you have settled on a satisfactory value, take the average of 10 such values; this is the scale maximum reading.

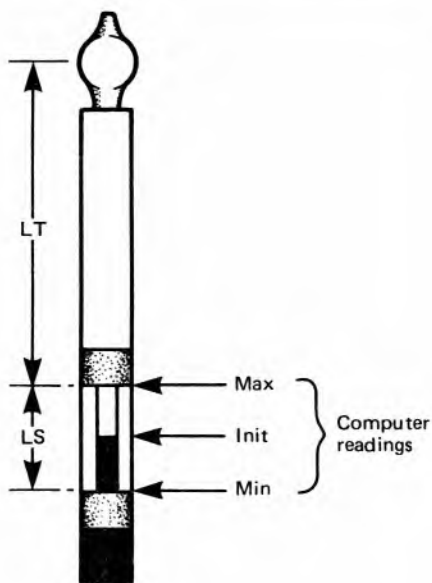
Now open the tap slightly and let the water run down until its level is about half-way along the slit. Allow the tube to stand for a minute to let water drain down from the walls of the tube above.

If the barometer is to be kept in a fairly constant temperature, it is now ready for programming. Then run the program and take an average of half-a-dozen results. This is the initial reading. At the same time, measure the temperature and pressure (use an ordinary aneroid or Fortin barometer). You now have several figures ready for the final programming (Fig. 16.4):

MAX	Maximum scale	)	
MIN	Minimum scale	)	values obtained from the test
INIT	Initial reading	)	program
P	1st pressure		— in millibars etc
T	1st temperature		— in kelvin (Celsius + 273)
LT	Length of tube above slit		(in mm)
LS	Length of slit		(in mm)

## Programming

The first thing to work out is the value we are to use in the program to represent the initial position of the water level, as measured from the top of the tube. This is what we have called the '1st volume'. It is the volume with which all later readings of volume are to be compared. Using the abbreviations listed at the end of the previous section, the formula for this is:



*Fig. 16.4 The readings needed to calibrate the barometer*

$$\text{1st volume} = LT + \frac{(\text{MAX} - \text{INIT}) \times \text{LS}}{\text{MAX} - \text{MIN}}$$

For example, if the tube is 100mm long, the slit 20mm long, MAX count is 270, MIN count is 200 and the INIT count is 250, then we work out:

$$\text{1st volume} = 100 + \frac{(270 - 250) \times 20}{270 - 200} = 106 \text{ (rounded off)}$$

This value is used in all later calculations until, perhaps you find that the water has dried out and needs replacing, in which event you start again with new measurements and a new value for 1st volume.

The same formula applies for calculating the volume at other times, except that you use the actual count obtained in place of INIT. For example, if the reading (x) is 230, the computer calculates the new volume as 111, using the line:

$$60 \text{ LET new} = 100 + (270 - x) * 20 / 70$$

We now have a new volume (the same thing as the 2nd volume referred to on p. 139), which is used to work out the new pressure. Actually, if you are not bothering to correct for temperature there is no need to go any further. You can use the new volume figures directly. If pressure is rising, these figures show a decrease, and conversely, if pressure is falling, the figures rise. You then discover, by experience, just how rapid the rise or fall has to be to predict a change in the weather.

If you want to convert the new volume into a pressure reading, the program line is:

$$70 \text{ LET pressure} = 106 * P / \text{new}$$

where P is the initial pressure. The figure 106 refers to the 1st volume as already calculated. Your 1st volume will probably require a different figure to be substituted here. The value of 'pressure' can then be displayed, and its units are the same as those used when you read the aneroid or Fortin barometer to begin with.

If you want to take temperature into account, you first need to program the micro to read this, using Project 15. Let us assume that you have done this and the temperature in degrees Celsius is held in the variable 'temp'. The *unconverted* pressure is the variable 'pressure' calculated as already described. The line for conversion is:

$$80 \text{ LET pressure} = \text{pressure} * T / (\text{temp} + 273)$$

T is the initial temperature (p. 144).

If you want extra precision, you should allow for the fact

that the water vapour in the tube also exerts its own pressure, depending on temperature. An approximate correction for this, over the range that the barometer is likely to encounter can be added to the line above:

$$80 \text{ LET pressure} = \text{pressure} * T / \text{temp} + 273 - (\text{temp} + 10) / 27$$

This gives you a reading of pressure with as much exactness as the barometer is capable. You can use it for keeping records. Check it against a proper barometer, or against a recent weather map sometimes, as factors within the apparatus may change and you may need to alter some of the constants of the program. Certainly the water will need topping up occasionally, and eventually need renewing, when many of the constants will need recalculating. Thus it is better to regard this as an experimental project, suited for detecting short-term pressure changes and forecasting the weather accordingly.

### *PARTS REQUIRED for the BAROMETER*

*Resistors* (carbon, 0.25W, 5% tolerance)

R1 220k

R2 100k

R3 830R

R4 MKY7C38E light-dependent resistor (or similar type  
such as ORP12)

R5 100k

*Capacitor*

C1 47n polyester

*Integrated Circuits*

IC1 CD4011 CMOS quadruple 2-input NAND gates

IC2 507C voltage-to-time converter

*Miscellaneous*

LP1 5V 60mA filament lamp, wire ended (socket required  
if wire-ended type not used)

Circuit board

8-pin IC socket

14-pin IC socket

3-way socket to fit 3-way plug of decoder

1mm terminal pins (5 off)

Plastic tubing about 20cm long, about 5mm external diameter (e.g. aquarium aerator tubing, but must be transparent with glass-clear walls)

Tap to fit tubing (aquarium aerator line tap), or screw clip  
black PVC insulating tape

Small glass or plastic jar, bottle or medicine tube

Other materials for building the assembly

## Project 17

### SUNSHINE RECORDER

We finish this series of weather station projects and the book itself in an optimistic mood! This project measures the amount of light energy arriving from the Sun over a given period of time. It is an integrating circuit, which means that instead of telling you how much energy is arriving at any given moment, it tells you how much has arrived since you last set it. Thus the micro has only to take a reading from time to time. The fact clouds may pass across the Sun or the shadow of a tree may pass across the garden while the micro is otherwise engaged, will be taken account of when the next reading is made.

#### How it works

This is one of the more complicated projects, but certain features of it will be familiar to you if you have already built some of the other projects. Sunlight falls on a photovoltaic cell (B1, Fig.17.1). This is a silicon cell of the type often referred to as a *solar cell*. This produces a voltage of approximately 0.45V when fully illuminated, the voltage depending on the amount of sunlight reaching it. This voltage is fed to an operational amplifier (IC1) which is wired as an *adder*. Its function is to add the voltage from the photocell to the voltage from the potential-divider RV1. The reason for this will be explained in a moment.

The output of IC1 goes to a transistor Q1. This is acting as a constant-current device which is charging capacitor C1. We rely on the fact that, for any given base current, the collector current is relatively unaffected by the PD between the collector and emitter. As C1 charges, the potential at point A rises. The result is that the PD between the collector and emitter of Q1 decreases. However, until this PD falls much lower than we ever allow it to fall, the reduction of PD does not reduce the current flowing into C1.

The amount of current flowing depends on the base current

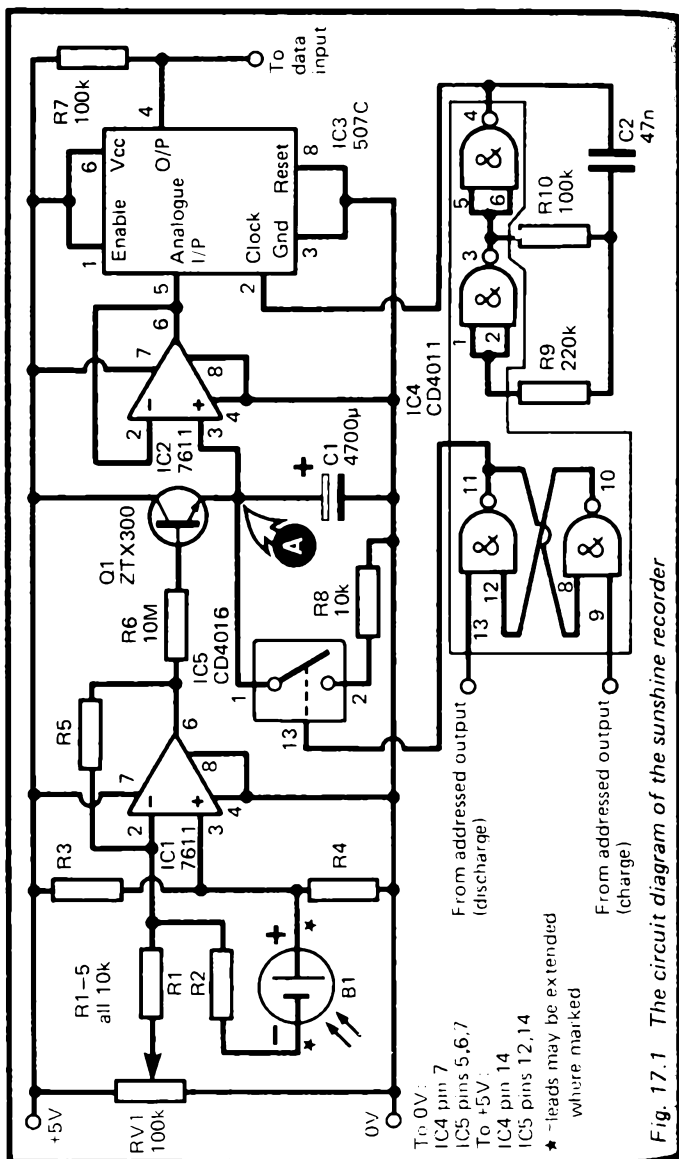


Fig. 17.1 The circuit diagram of the sunshine recorder



flowing through to Q1 from IC1 and passing through R6. The value of R6 is high, to keep this current very small. Until the potential at the output of IC1 exceeds about 0.6V there is no base current at all and Q1 is switched off, but, in dim light the PD across B1 may be *less* than 0.6V. We have to ensure that as soon as *any* PD is developed across B1 a current begins to flow to Q1. This is the reason for the adder. The output of IC1 is the sum of the two inputs. RV1 is set to deliver an input of 0.6V, so that any additional input due to B1 goes fully toward driving Q1.

Actually this is an inverting adder, so we apply 1.9V from RV1 (this is 0.6V less than the half-way voltage at pin 3 of IC1) and the cell is connected with its negative terminal to R2. This results in a positive output to Q1.

Over a period of several tens of minutes, or perhaps an hour, C1 charges at a rate depending on the amount of sunlight from moment to moment. The PD across C1 rises, faster when the Sun shines strongly, more slowly when a cloud covers it or at the end of the day, and not at all during the night. The PD is fed to another operational amplifier, IC2, which is wired as a *unity gain voltage follower*. This is used as a buffer between C1 and the voltage-to-time converter, IC3. The input of IC3 has an impedance of about 100k, which would allow current to leak away from C1 almost as quickly as it was arriving from Q1. But the input impedance of IC2, which is a CMOS IC, is extremely high (about 1 Teraohm, or  $1 \times 10^{12}$  ohms!). Leakage to this input is virtually non-existent. The output of IC2 follows the PD which is across C1 exactly. As explained in Project 11 (p. 102), IC3 converts this voltage into a waveform. The shape of this depends on the level of the voltage.

For IC3 to operate properly the input voltage must be in the range 1.25V to 3.75V. During a period of measurements the PD should start at a little above 1.25V and must not exceed 3.75V before the reading is taken.

The micro is given the task of adjusting the PD across C1 to a little more than 1.72V each time a measurement period is to begin. It discharges C1 through resistor R8, which is switched into circuit by a CMOS switch (IC5). This is con-

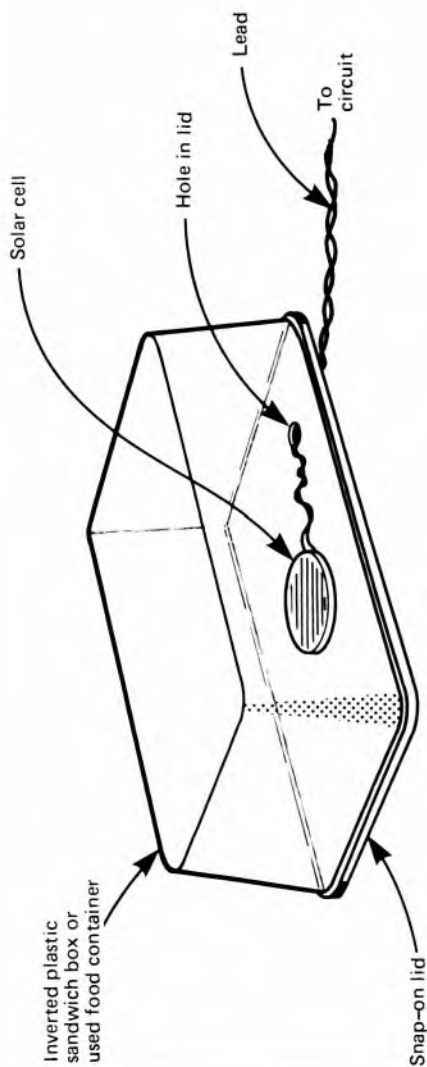
trolled by a flip-flop made from two gates of IC4. The other two gates of this IC make up the clock for IC3 (see p. 105). The flip-flop is controlled by two Addressed Outputs. Addressing one of these sets the flip-flop, which closes the CMOS switch and begins to discharge C1. The micro checks the PD by reading the Data Input. When the PD has reached a suitably low level, the micro addresses the other Addressed Output. This action resets the flip-flop, which opens the CMOS switch and C1 begins to charge again. It requires a 'high' level on pin 13 of IC5 to close the switch.

The length of each measurement period is fixed so that, even with continuously brilliant sunshine, the PD across C1 never exceeds 3.75V.

## Building it

The project requires 5 lines to the decoder: two power lines, a wire to a Data Input, and two lines from Addressed Outputs. If the solar cell (B1) is to be mounted away from the micro, the leads may be extended where marked \* in Fig.17.1. The solar cell need not be a large one, in fact a small one is better, for it delivers less current and so allows longer measurement periods. It is often possible to buy small fragments of solar cells very cheaply from hobby suppliers. One of these may well prove to be suitable. This is a reminder that solar cells are easily broken, unless you have one that is in a protective case. It must be mounted where it is protected from the weather and physical damage yet receives full sunshine for as much as possible of the day. The cell must be horizontal, facing upward. Fig.17.2 shows one way of protecting it. If, later, you find that the capacitor charges too rapidly, a layer or two of white paper may be fixed inside the cover to reduce the amount of light reaching the cell. Remember to clear away any debris which may fall on the cover, especially leaves in autumn.

Wire up IC1 and Q1 and their associated components first. Do not include the connection to IC5 at this stage. The circuit may be tested by connecting it to a +5V or +6V supply. Connect a voltmeter across C1 (at point A). Shade B1 so that



*Fig. 17.2 Mounting and protecting the solar cell. The plastic should be clear, or white and translucent. The cell may be held in place by adhesive or by a double-sided 'sticky fixer'*

no light reaches it. Alter the setting of RV1 several times. In some positions you will see the PD across C1 slowly increasing. It should take several minutes to increase by as much as 0.1V. Adjust RV1 so that the PD *just* does *not* rise. This is the point at which it is delivering approximately 1.9V. You would need to wait for, say, 10 minutes to make certain that no rise in PD is occurring, but for the present be satisfied if it does not apparently change in one or two minutes. Now uncover B1. The needles should start to move, showing a rising PD. The rate of rise depends on the amount of light. Let bright sunlight fall on it, or hold a table-lamp a few centimetres away from it and the rate of rise should increase perceptibly.

Next add IC2, IC3 and IC4 to the circuit. When tested as above, with a voltmeter connected to the output of IC2 (pin 6), the same result should be obtained. With an oscilloscope connected to the output of IC3, you see a waveform similar to that of Fig.11.2. The proportion of time spent in the 'high' state is gradually reduced as C1 charges.

Finally, wire up the flip-flop of IC4 and the single CMOS switch of IC5. Note that the control inputs of the other switches need to be connected to either 0V or +5V (see list of connections on Fig.17.1). The flip-flop may be controlled by wiring its inputs to +5V temporarily, then briefly connecting either one or the other to 0V instead. As the flip-flop changes state the switch is turned off or on. When the switch is on, C1 discharges.

The capacitor discharges much more rapidly than it charges, but still takes several seconds. It must not discharge too rapidly or the micro will be unable to stop the discharge at the correct level.

Before connecting the completed device to the decoder, test all the leads to ensure that there are no short-circuits between them. Plug the device into the Decoder and switch on the power. Allow C1 to charge until the PD across it is between 1.25V and 3.75V, as measured by a voltmeter.

Use a program like that on p. 106 to read the input from the voltage-to-time converter. Cover the photo-cell to exclude all light and set RV1 so that the reading obtained by the micro does not decrease over a long period, say 10 minutes. Of course,

individual results may vary by 2 or 3 counts, but the average should remain unchanged. The setting should be such that any slight movement of the wiper of RV1 toward the 0V end of its track, causes the capacitor to begin charging, and the figure displayed by the micro to fall. Now uncover the photo-cell. The reading should begin to fall immediately, and its rate of fall should vary with the amount of light reaching the photo-cell.

Discharge C1 until the PD across it is a little more than 1.25V. You can do this either by operating the flip-flop by 'OUT' commands or, more simply, by temporarily connecting a 10k resistor across it. When the capacitor is discharged to the correct level, run the program again several times. The reading should be at the high end of the range. Select a value as the 'discharge level'. The program for discharging the capacitor requires two output addresses. Supposing that these are 31 for discharging the capacitor and 51 for charging (i.e. stopping discharge), and supposing that the value for the 'discharge level' is 420, the following program will control the process:

```
10 OUT 31,0
20 PAUSE 10
30 OUT 51,0
40 LET x=0
50 FOR j=1 TO 500
60 IF IN=255 THEN LET x=x+1
70 NEXT j
80 IF x<420 THEN GO TO 10
90
```

Lines 10 to 30 turn on the CMOS switch for a tenth of a second. Lines 40 to 80 read the voltage to find if it has fallen to the 'discharge level'. If it has not, the process is repeated. When it has reached 'zero level', the program drops through to line 90. Here, the micro goes into some other routines while the capacitor charges again at a rate depending on the amount of sunlight. These routines must include some kind of timing sub-routine so that, say, 30 minutes later, the micro returns to read the value shown by the sunshine recorder. This is proportional to the amount of sunlight which has reached the photo-

cell during the measurement period. This value may be displayed, or stored in memory to be added to values obtained throughout the day.

### *PARTS REQUIRED for the SUNSHINE RECORDER*

*Resistors* (carbon, 0.25W, 5% tolerance, unless otherwise specified)

R1–R5	10k, 1% tolerance (5 off)
R6	10M
R7, R10	100k (2 off)
R8	10k
R9	220k
RV1	100k, miniature preset resistor

#### *Capacitors*

C1	4700 $\mu$ electrolytic
C2	47n polyester

#### *Semiconductor*

Q1	ZTX300 or similar npn transistor
----	----------------------------------

#### *Integrated Circuits*

IC1, IC2	7611 CMOS operational amplifiers (2 off)
IC3	507C voltage-to-time converter
IC4	CD4011 CMOS quadruple 2-input NAND gate
IC5	CD4016 CMOS quadruple analogue switch

#### *Miscellaneous*

B1	Silicon photo-voltaic cell (solar cell), small
	8-pin IC sockets (3 off)
	14-pin IC sockets (2 off)
	3-way sockets to fit 3-way plugs on decoder board (3 off)
	Circuit board
	1mm terminal pins (7 off)
	Materials for making protective case and for mounting the cell

## Appendix A

### THE ADDRESS DECODER

It is essential for you to make this, for you need it when you come to connect the projects to the micro. It does several things beside decode addresses. We will explain how it works in some detail, but if you would rather go ahead and build it, skip the description that follows and go straight to p. 162.

Fig.D.1 is the circuit diagram. All the lines coming in on the left-hand side of the diagram come from the micro. Lines A0 to A4 all go to IC1. This is a NAND logic gate. When the inputs to this gate are all 'high' (+5V), the output of the gate (pin 8, where the small circle is), goes 'low' (0V). The gate has 8 inputs but three of these are made 'high' permanently by wiring them to the +5V supply. Therefore the output of IC1 is normally 'high', but goes 'low' when the address on the bus is XXX1 1111. In this expression 'X' means either 0 or 1, for it makes no difference, as lines A5 to A7 are not connected to IC1. If you look at Table 0.1, you will see that all the binary numbers marked \* end with a run of five 1s, and in fact the 8 numbers so marked are all the possible ones of the form XXX1 1111. When any one of these is on the address bus, the output of IC1 goes 'low'.

The output of IC1 together with the  $\overline{\text{IORQ}}$  and  $\overline{\text{RD}}$  lines go to another logic gate, which is one of the gates in IC2. These are NOR gates, and have three inputs. The output of a NOR gate goes 'high' when all its inputs are 'low'. The only conditions for which this can happen are:

- |                                       |   |
|---------------------------------------|---|
| Output of IC1 is 'low'                | — address XXX1 1111 on the bus                                |
| and $\overline{\text{IORQ}}$ is 'low' | — the Z80 wants to communicate with an external device (p. 5) |
| and $\overline{\text{RD}}$ is low     | — the Z80 wants to read data from that device.                |

This happens only when the Z80 is trying to obtain data from

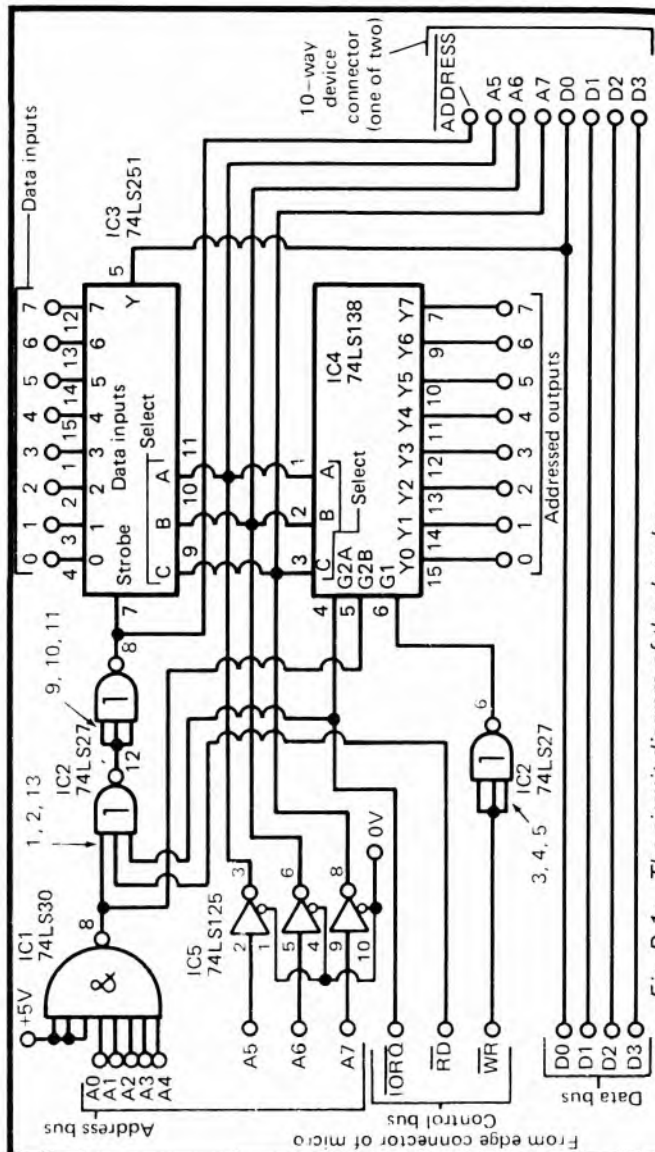
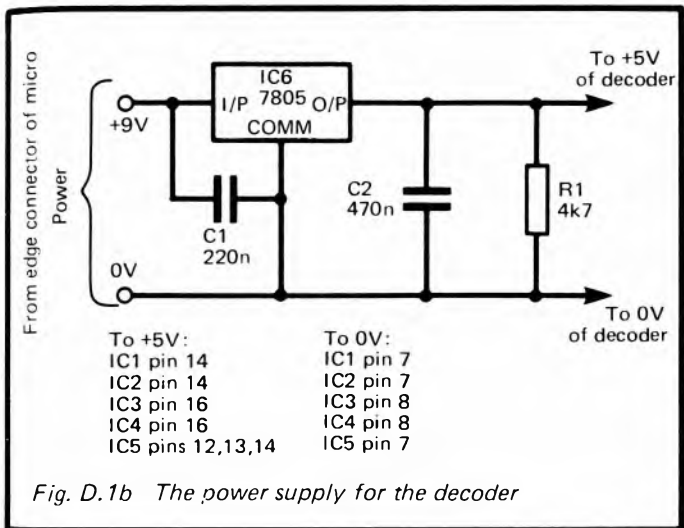


Fig. D.1a The circuit diagram of the decoder





one of the projects. When the output of the NOR gate (IC2, pin 12) is 'high', the output of the next NOR gate (pin 8) becomes 'low'. This is because this NOR gate has all its input pins wired together. This makes it behave as an INVERT gate; if its inputs are 'high', its output is 'low', and the other way about.

When the output from IC2, pin 8, goes 'low' this makes the Strobe input of IC3 'low'. IC3 is a data selector IC. It has 8 Data Inputs, to which you can connect up to 8 different projects. When the Strobe input goes 'low', *one* of these inputs becomes connected to the output Y at pin 5. The state of that input (i.e. the data from one of your projects) appears on line D0 of the data bus. The Z80 then reads the data bus to find out whether the data is 0 or 1 ('low' or 'high').

But *which* one of the Data Inputs is connected to the bus? This is decided by the state of the Select inputs A, B and C. As you can see from Fig.D.1, these are wired to lines A5, A6 and A7 of the address bus through *three* buffer gates (IC5). The buffers provide extra power for driving the inputs of IC3,

IC4 and any other decoder circuits we may need. If these three lines are all 'high', this in effect puts 111 at the Select inputs. Binary 111 is 7 in decimal (see Table 0.1). If you have a device connected to Data Input 7 of IC3, the data from this device appears at the output (Y) and goes to the data bus. When lines A5 to A7 are all 1 and lines A0 to A4 are also all 1, the address on the bus is 1111 1111, or 255 in decimal. The device which is connected to input 7 of IC3 has the address 255. If you want the micro to read the data coming from this device, you must program it to read from address 255. The way to do this is explained later.

As another example, suppose you want to read data from a device attached to Data Input 4. Decimal 4 is 100 in binary, so A7 must be 'high' while A6 and A5 are 'low'. The complete address required is 1001 1111, or 159 in decimal.

Output Y of IC3 is known as a *three-state output*. When the Strobe input is 'low', Y has the same level (0 or 1) as is present on the selected input line. If the input is 0, Y is 0; if the input is 1, Y is 1. Y goes to its third state when the Strobe input goes 'high'. The third state is neither 0 nor 1, but what is called 'high impedance'. It is as if the output terminal of the IC becomes disconnected from line D0. The ability to disconnect Y is essential, for there are many other ICs connected to the data bus. The RAM and ROM ICs, the keyboard (in some computers, but not in ZX computers), the Z80 itself and many other devices may all have connections to the bus. If they all were putting data on to the bus at once, it would be like a room full of people all shouting together. No one would be heard. So all outputs to the bus are of the three-state kind. When the Z80 addresses a particular IC, as it does when it makes  $\overline{\text{IORQ}}$  and  $\overline{\text{RD}}$  low and puts one of their addresses of IC3 on the address bus, only the addressed IC is able to put data on to the bus. At that time, all the other ICs are in a high-impedance state and are, in effect, disconnected.

So far, we have seen how the Z80 reads data from our projects. The 8 Data Inputs (Fig.D.1) are for use by those projects which deliver output on only 1 line. We say they have a *1-bit output*. A few projects need to transmit more than 1 bit at a time. These each have their own IC with three-state

outputs to connect them with the data bus.

Some projects do not send data to the micro, but simply need to be told when they are to operate. All they need is a brief message to tell them when to begin. IC4 has 8 outputs, the Addressed Outputs, which are used to trigger such projects into action. The IC has 3 Enable inputs. G2A and G2B are connected to the output of IC1 and to the  $\overline{\text{IORQ}}$  line. Both of these inputs must be 'low' to enable IC4. The third Enable input, G1, is connected to the  $\overline{\text{WR}}$  line by way of an INVERT gate (part of IC2). To enable IC4, G1 must be made 'high'; this happens when  $\overline{\text{WR}}$  goes low. So, if the Z80 wants to tell a device to do something, it puts its address (XXX1 1111) on the bus, and makes  $\overline{\text{IORQ}}$  and  $\overline{\text{WR}}$  low. This action enables IC4.

The outputs of IC4 are normally all high, but when the IC is enabled, *one* of these goes low. Which one goes low depends on the state of the 3 Select inputs, as it did for IC3. For example, if the address is 0111 1111 (127 in decimal), the number is presented to the Select inputs is 011. In decimal, this is 5, so Addressed Output Y5 goes low, while the others remain high. The device attached to output Y5 is designed so as to respond when the output goes low. In this way we can trigger up to 8 devices to operate, simply by programming the micro to write to the appropriate Addressed Output.

A few of the projects need more than 1 data line. The decoder provides them with up to 4 such lines (D0 to D3), as shown at the bottom right-hand corner of Fig.D.1. These come directly from the data bus of the micro. These projects are also supplied with lines A5 to A7 (indirectly from the buffer gates of IC5) as well as the decoded output of lines A0 to A4,  $\overline{\text{IORQ}}$  and  $\overline{\text{RD}}$ . The final stage of address decoding for these projects is carried out by the circuit of the project concerned.

The remaining section is a +5V regulator circuit. This provides power for operating the address decoder and for all the projects which are attached to it. The regulator draws its supply from the +9V line of the micro. This is able to supply only a limited amount of power (up to about 200mA, probably less if you already have a printer, disk drive or extra RAM connected). Most of the projects in this book require only a few milliamps so there is no problem in powering two

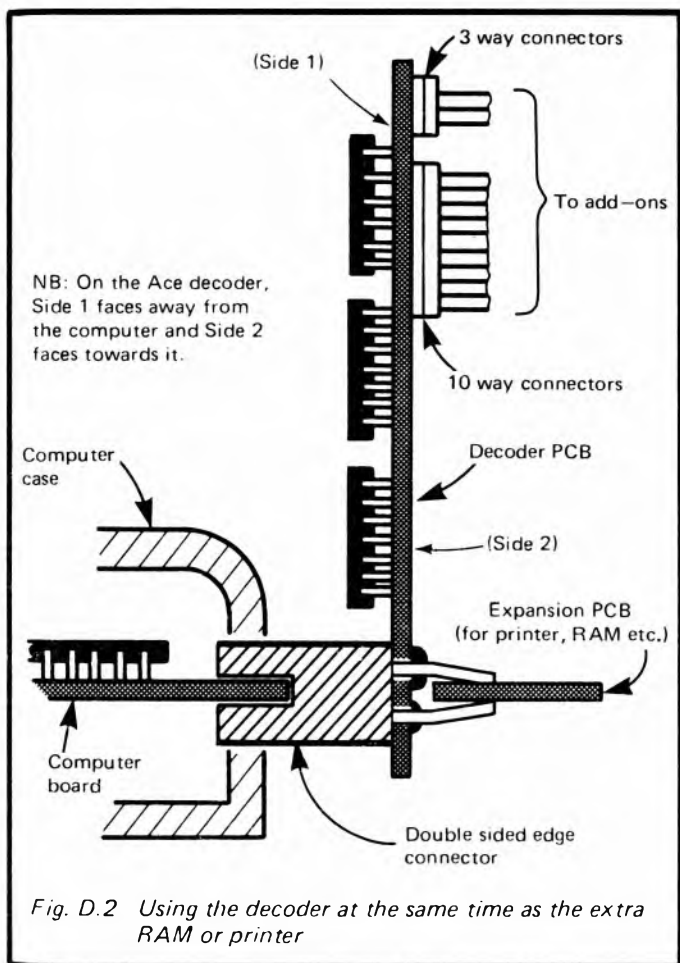
or three of these at once. If you want to have many projects connected at one time, it is advisable to use a separate unregulated DC power pack. You could employ a second ZX Power pack, or any other pack which delivers rectified direct current of suitable voltage. The regulator IC can be powered by any supply voltage in the range 7V to 25V, and delivers a current of up to 1A. If you use a separate power pack rather than the supply from the micro, the circuit diagram and PCB layout must be modified.

## Building the project

If you want to be able to run a printer, extra RAM or disk drives at the same time as the projects, adopt the method of connection shown in Fig. D.2. Fig. D.3 shows the expansion board with keyway positioning. If you are connecting the project to a Spectrum, which has a 28-way connection, with the key at position 5, you can use an edge connector socket of this description, though there is no need to do so. As Fig. 0.1 shows, the projects do not use any of the lines at the ends of the Spectrum connector and a socket which fits the ZX81 can be plugged on to the Spectrum too. On the Ace, the key is at position 23, so we simply turn the connector the other way up. The lugs of the socket pass through a row of holes in the circuit and are soldered to the board. The lugs are then bent slightly to bring the two rows closer together and are soldered to a strip of board which acts as an edge connector for any other device (such as the printer) which you may want to plug on to the system. This is a double-sided board with 22 parallel tracks (27 for the Spectrum) on each surface (Fig. D.3). If you have a ZX81 and think it likely that you will be upgrading to a Spectrum, it is worthwhile to provide a board with 27 tracks per surface at this stage.

Fig. D.4 shows the design of a PCB for the project. This is reproduced full size, so you can trace it or photocopy it if you want to make the board yourself. The board is also on sale ready-made from the supplier named on p. 179.

If you decide to design your own PCB or to assemble the project on stripboard, note the way in which connections are



provided for attaching the other projects to this board (Table D.1). The Data Input plugs each have 3 pins. One pin on each socket is connected to the 0V line, and one to the +5V line. The third pin is connected to the corresponding input



Table D.1 Connector pin outlets

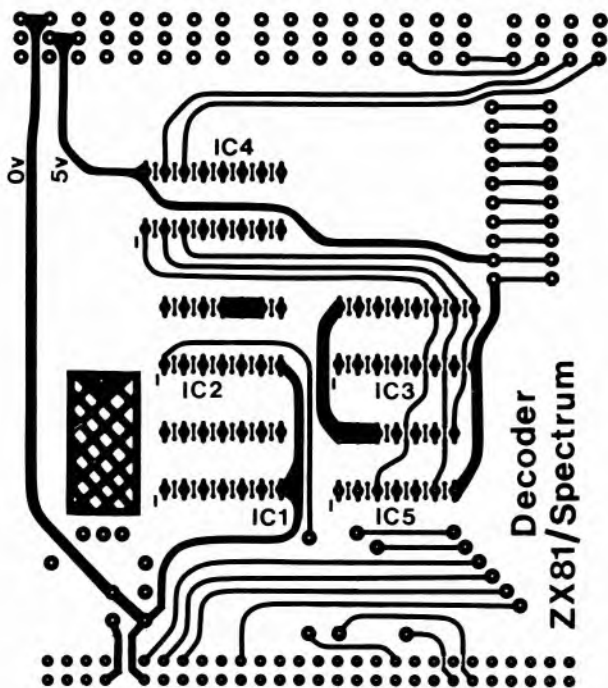
Connectors W & X (from add-ons)		Addressed Outputs		
pin	function	connector	pin 1	pin 2
1	A5	a	0v	5v
2	A6	b	0v	5v
3	A7	c	0v	5v
4	<u>ADDRESS</u>	d	0v	5v
5	D3	e	0v	5v
6	D2	f	0v	5v
7	D1	g	0v	5v
8	D0	h	0v	5v
9	5v	s	0v	5v
10	0v	t	0v	5v

pin 3	Data Inputs			
	connector	pin 1	Pin 2	pin 3
A0	i	0v	5v	D3
A1	j	0v	5v	D4
A2	k	0v	5v	D2
A3	l	0v	5v	D5
A4	m	0v	5v	D1
A5	n	0v	5v	D6
A6	o	0v	5v	D0
A7	p	0v	5v	D7
A0	q	0v	5v	D7
A1	r	0v	5v	D6



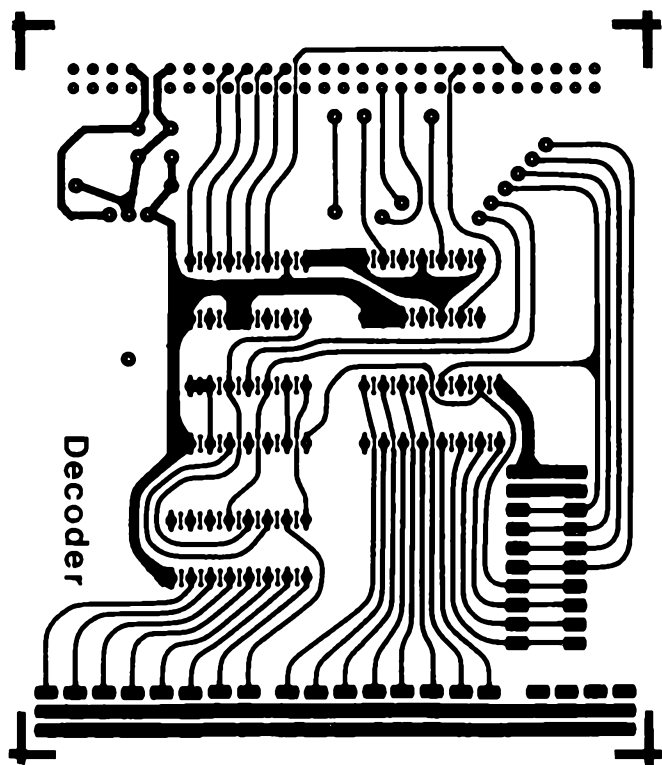
If you prefer, you can solder 3 separate 1mm diameter terminal pins in place of each socket. Make soldered connections to each pin (or use crocodile clips) to join a circuit to the board.

There are two plugs (10 pins each) for connecting those projects which require more than 1 line of the data bus. Table D.1 shows the connections to each pin.



*Fig. D.4a Layout of ZX81/Spectrum decoder board – side 1.  
(The edge connector terminals run from 1 on the left to 23 or 28 on the right.)*

The first step in building the decoder is to solder the edge-connector socket and extension strip (Fig.D.3), if used, to the board. Then assemble the 5V regulator circuit (IC6, C1, C2 and R1). You can test this by plugging the card into the micro and switching on the ZX power pack. Use a voltmeter to measure the voltage across R1, which should be almost exactly 5V. If you are using the PCB design and intend to use a separate power pack, cut the tracks at the points shown in Figs. D.5 or D.6. Solder the wires from the power-pack to

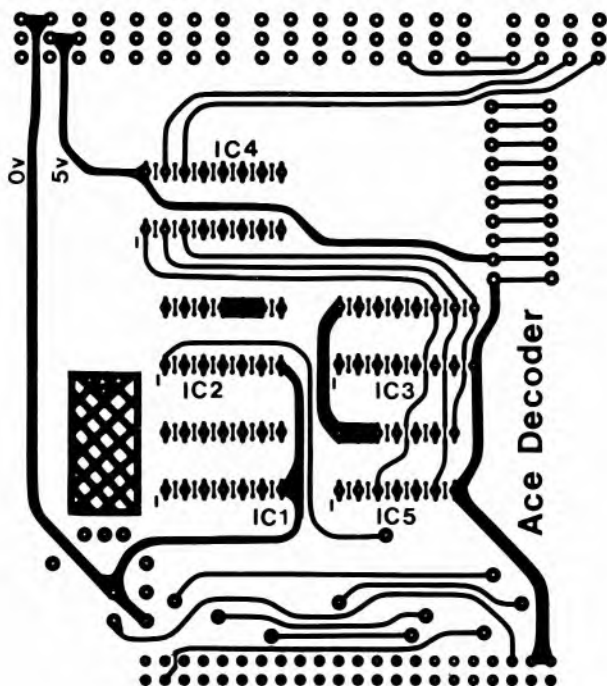


*Fig. D.4b Layout of ZX81/Spectrum decoder board – side 2*

the wires of C1, as shown.

Next wire up IC1 and IC2. Test this without plugging the board into the micro. The inputs to this circuit are all un-connected and act as if they were 'high' inputs. This being so, the output of IC1 pin 8 is 'low' IC2 pin 12 is 'high' and IC2 pin 6 is 'low'

The data input side of the circuit is completed by assembling IC3 and IC5. Test the output of IC5 by connecting each of A5 to A7 in turn to the 0V line. Their outputs should fall from



*Fig. D.4c Layout of Ace decoder board – side 1.  
(The edge connector terminals run from 2 on the left to 23 on the right.)*

+5V to 0V. Now connect  $\overline{IORQ}$  and  $\overline{RD}$  to 0V, as if the micro is attached and is trying to read data. This should make the output of IC2 (pin 8) go 'low' and so enable IC3. Its Y output (pin 5) should be 'high'. If A5 to A7 are unconnected, they act as 3 'high' inputs, equivalent to address '7'. If you now connect input 7 to the 0V line, the output of IC3 should likewise fall to 0V. There is no need to check the circuit with other addresses unless you have reason to suspect that the IC itself is faulty.

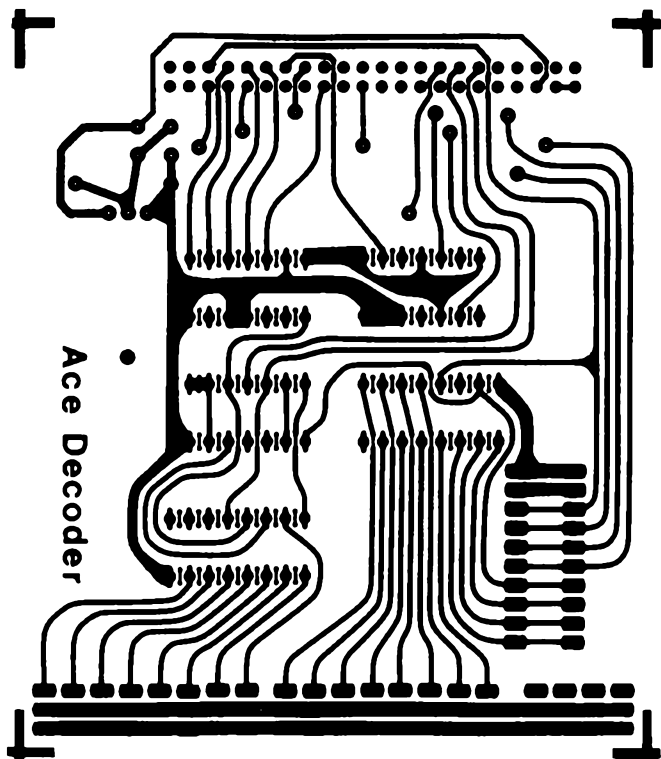
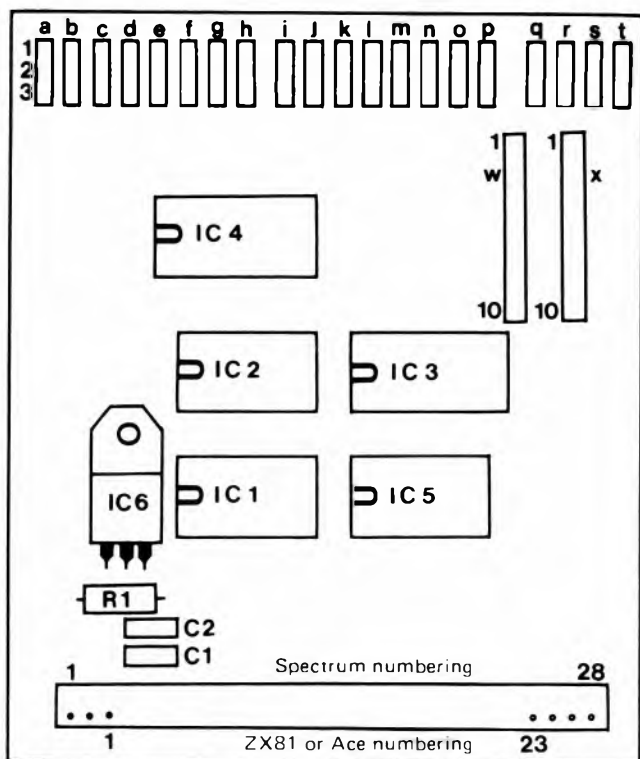
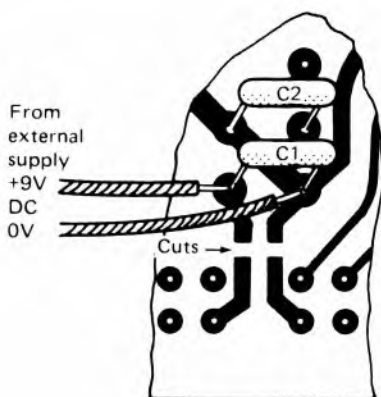


Fig. D.4d Layout of Ace decoder board – side 2

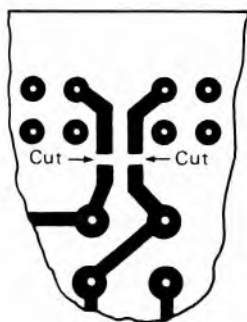
Finally, add the remaining IC, IC4. This can be tested in a similar manner. First test all its outputs, which should be at +5V. Now connect  $\overline{WR}$  and  $\overline{IORQ}$  to the 0V line, to enable this IC. The address is '7', as before, so test output 7 (pin 7) to check that it has fallen to 0V. The remainder of the checking is best done by using the micro, as described in the next section.



*Fig. D.4e Component layout for both decoder boards – side 1.  
(Note terminal 1 is not present on the Ace board as it is a key way.)*

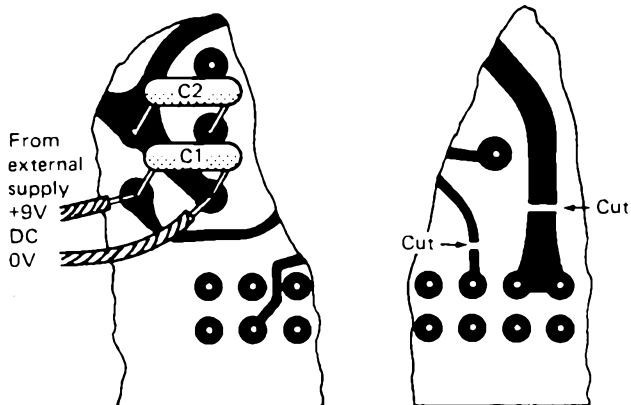


SIDE 1



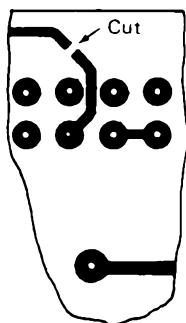
SIDE 2

*Fig. D.5 Modifying the ZX81/Spectrum decoder board for use with an external power supply*



SIDE 1

SIDE 2



*Fig. D.6 Modifying the Ace decoder board for use with an external power supply*

## Controlling the decoder

The Spectrum is easier than the ZX81 to program for testing the decoder, so we will deal with this first. To test the Data Input side we simply use the command `LET X = IN 31`, where the number following `IN` is the decimal address of the input we want to read. These addresses are listed in Table 0.1, p. 7 and again in Table D.2. Address 31 corresponds to data input 0. With all data inputs *unconnected*, run this program:

```
10 LET X = IN 31
20 PRINT X
```

Data bus lines which are unconnected (i.e. lines D4 to D7) count as 1s, and so do any unconnected data inputs to the decoder. Therefore all lines read 1, and the value returned for `X` is 255 (i.e. 1111 1111 in binary). Now connect data input 0 of IC3 (NOT line D0 of the data bus) to 0V. Run the program again. This time the data bus is 1111 1110 in binary, so the program returns 254. If you disconnect data input 0, and connect data input 1 to 0V instead, the program returns 255 for you are still addressing input 0. Change line 10 to `LET X = IN 63` and the result 254 is returned.

This program finds out whether a given data input is 0 or 1. When you have a project attached to the decoder and are programming the micro to work with it, use the command

*Table D.2 Addressing codes*

Data input number	Address code	Character in REM	POKE N to 16517
0	1F	3	31
1	3F	Z	63
2	5F		95
3	7F		127
4	9F	inverse *	159
5	BF	inverse Z	191
6	DF	TO	223
7	FF		255



“IF . . . .THEN. . .” to make the micro perform one action if  $X = 254$ , and a different action if  $X = 255$ .

The Addressed Output section is tested in a similar way, but there is one small problem. The  $\overline{IORQ}$  and  $\overline{WR}$  lines are in their active-low state for such a short time that you can not use a voltmeter to detect the brief fall in voltage at the output of IC4. Even an oscilloscope will not show what happens. The best method is to build a simple pulse detector. Project 1 describes how to do this. Connect the detector to output Y0 (pin 15), reset the detector (LED on) and run this program:

```
10 OUT 15,0
```

The 15 is the address of Y0. The 0 is the data which is supposed to go to this address, but in this case we are not sending any data. It is sufficient to address the output. When you run the program, the LED goes out. The other outputs can be tested in the same way, but using their addresses (Table D.2) instead.

## Programming the ZX81

The Z80 microprocessor inside the ZX81 can do just the same things as the Z80 inside the Spectrum, but the ZX81 does not have the IN and OUT commands in its BASIC. We have to use a machine-code program to tell the ZX81 to communicate with attached devices. Fortunately, these programs are very short. The program for reading from a data input (equivalent to using IN, as described above for the Spectrum) is simply this:

```
06  FF  LDB, FF
0E  1F  LDC, 1F (the address)
ED  48  IN C, C (read addressed port)
C9      RET
```

Do not worry if you do not understand what machine code is all about. It is listed here just as a matter of interest for those who do. You can use the program from an ordinary BASIC program without the need to know how it works. The best way of using this program is to place it at the very

beginning of your BASIC program, in the form of a REM statement. Chapter 26 of the ZX81 Handbook explains something about this.

Here is a BASIC program which puts the machine code program in memory from address 16514 (where all BASIC programs begin), and lets you use it by means of theUSR command:

```
10 REM █ :3 TAN
20 POKE 16515, 255
30 POKE 16518, 237
40 POKE 16519, 72
50 LET X = USR 16514
60 PRINT X
70 STOP
```

The REM puts some of the machine code groups directly into memory by making the ZX81 store the codes which correspond to various symbols and characters (see Appendix A of the ZX81 Handbook). The 3, for example, gives the code for 1F, the address of Data Input 0 of the address decoder. Type in line 10 with great care, leaving one *or two* spaces where shown.

Some codes can not be put in as a REM statement so we POKE them in with lines 20 to 40. Line 50 makes the micro run the machine code routine from address 16514 onward. When it comes back again, having read the input from data input 0, the value of X tells us what is in the B and C registers of the Z80. We already know that register B contains 1111 1111 for we put it there with the program. Register C shows the data which has just been read. This is 1111 1111 if all lines are high (all data inputs left unconnected), and 1111 1110 if data input 0 has been connected to the 0V line. So X has one of two values:

1111 1111 1111 1111, which is 65535 (=high input)  
or 1111 1111 1111 1110, which is 65534 (=low output)

When you run this program the number 65535 or 65534 appears at the top left corner of the screen.

When the program has been run once, the 'missing' codes will have been inserted in the REM line, so do not be surprised

that line 10 now reads:

```
10 REM █ COPY :3 GOSUB ? TAN
```

To test the other addresses, alter line 10 as in Table D.2. If the table lists a character, put this in place of the 3 in line 10. If it does not, poke the number N. Type a space in place of the 3, and add this line to the program:

```
25 POKE 16517, N
```

If you have several devices controlled by the same BASIC program, you will want to be able to alter the address in the machine-code program to address each device. This is simply done by:

```
200 POKE 16517,N
```

where N is the decimal address required (see Table D.2). Follow this with the USR function (as in line 50 above) to read data. The address can be changed as often as you want during the program.

The machine-code program for output is slightly longer:

```
06 FF  LDB, FF
0E 1F  LDC, 1F (the address)
16 00  LDD, 00 (the data)
ED 51  OUT(D),D
C9     RET
```

As a BASIC program, this becomes:

```
10 REM █ :3— TAN
20 POKE 16515,255
30 POKE 16520,237
40 POKE 16521,81
50 LET X = USR 16514
60 STOP
```

There is no need for 'PRINT X' this time, since we are not trying to obtain any data but merely to trigger the attached circuit. Using the USR function is all we need to do. Connect the Pulse Detector (Project 1) to output Y0 of IC4. Reset it so that the LED comes on. Now run the program given above.

The LED goes out immediately. You can test other outputs by using other addresses. Table D.2 tells you what to use instead of the 3.

As with the other program, new symbols appear in the REM after the program has been run once:

```
10 REM ■ COPY :3— GOSUB ?TAN
```

There are two spaces after the —. The first of these corresponds to the data (a space is code 00). With Project 4 you need to send data to the data latches. In this case you will need to modify the program by substituting one of the symbols shown in Table 4.1 for the first space after the —. As with the first program, you can POKE different addresses into the machine-code program at any stage during your BASIC program so as to control several devices at once.

With this program and the other you can delete the 'POKE' lines when the program has been run once, for the first run places all the required codes in memory. These programs may be saved on tape or disk. After the first time the program is run, the REM line holds the address of the *first* device to be addressed by that program. If your BASIC program is controlling several devices with different addresses, note that the saved version holds the address of the *last* device to be used. If this is not the same as the first device, your program must begin by POKEing in the address of the first device.

If you want to read and write in the same program, both machine code programs may be placed one after the other. The combined program is:

```
10 REM ■ :3 TAN■ :3— TAN
20 POKE 16515,255
30 POKE 16518,237
40 POKE 16519,72
50 POKE 16522,255
60 POKE 16527,237
70 POKE 16528,81
```

Run the program once, then delete lines 20 to 70. These can be replaced by lines of your own program. To read data, use the statement LET X = USR 16514 as before. To write data

or trigger a device, use the statement `LET X = USR 16521`. To change the address of the device to be read, `POKE 16517`, as before. To change the address of a device to be written to, `POKE 16524`.

## Programming the Jupiter Ace

The Ace has two words which allow it to interact with the decoder. For reading data from the decoder, or from interfaces attached to the decoder, it has the word `IN`. Before `IN` is used, we place the address to be read from on the top of the stack. For example, to read from address 31 (Data Output 0), we use

```
31 IN
```

This leaves the required data on the top of the stack. As explained on p. 10, it is safer to `AND` this to eliminate floating values on other data lines. To read line D0 only, we use:

```
31 IN 1 AND
```

To read the bottom three lines D0 to D2 we use:

```
31 IN 7 AND
```

To read all four data lines D0 to D3 we use:

```
31 IN 15 AND
```

To write to an interface we use the word `OUT`. This needs the data to be present as second on stack and the address on the top of stack. Thus to send data 12 to address 63 we type:

```
12 63 OUT
```

Many of the interfaces work by simply addressing an Addressed Output, without actually sending data. Data of some sort is still required on the stack for use by `OUT`, so a convenient form of command is:

```
0 63 OUT
```

## *PARTS REQUIRED for the ADDRESS DECODER \**

Items marked \* are optional

### *Resistor*

R1 4k7, 0.25W, 5% tolerance

### *Capacitors*

C1 220n polyester

C2 470n polyester

### *Integrated Circuits*

IC1 74LS30 8-input NAND gate

IC2 74LS27 triple 3-input NOR gate

IC3 74LS251 8-line to 1-line data selector/multiplexer

IC4 74LS138 3-to-8 line decoder/multiplexer

IC5 74LS125 quadruple bus buffer gate with three-state output

IC6 7805 voltage regulator, 5V, 1A

### *Miscellaneous*

Stripboard or ready-made PCB

\* Extender board

Edge-connector to suit machine (pin 1–28 marked for Spectrum; pin 1-23 marked for ZX81 and Ace)

\* 14-pin IC sockets (3 off)

\* 16 pin IC sockets (2 off)

\* 3-way connector and socket (a–t) (20 off)

\* 10-way connector and socket (w & x) (2 off)

\* 1mm Terminal pins (if used instead of PCB plugs)

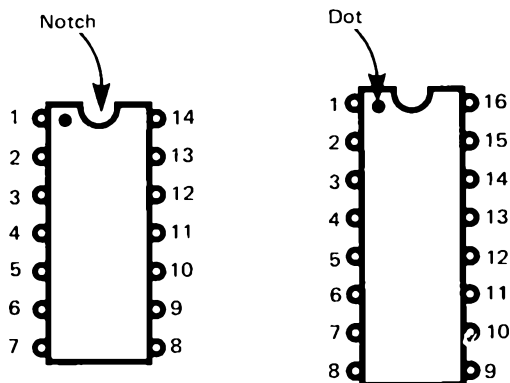
\* The double-sided plated through hole printed circuit board for the Decoder, together with all necessary components either in kit form or as individual items are available from:

KELAN (HOBBYBOARD),  
A DIVISION OF KELAN ENGINEERING LTD,  
NORTH WORKS,  
HOOKSTONE PARK, HARROGATE,  
NORTH YORKSHIRE, HG2 7BU, ENGLAND.  
Telephone: Harrogate (0423) 883672

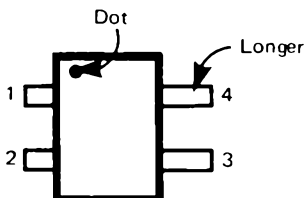
PLEASE NOTE: The Publishers are in no way responsible for the manufacture or supply of the above and all enquiries must be sent directly to Kelan (Hobbyboard).

## Appendix B

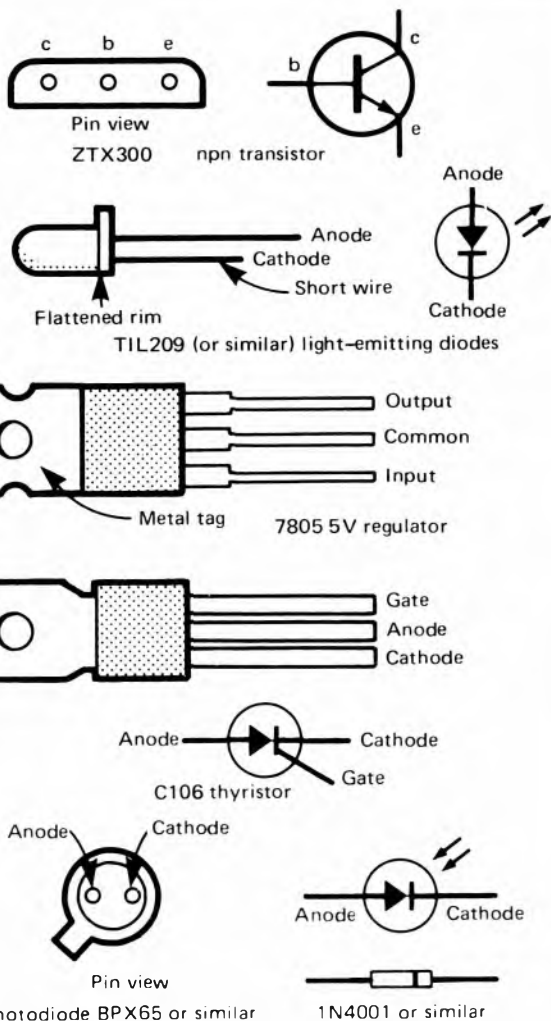
### PIN LEAD-OUT DETAILS



*Fig. A.1 Pin numbering of integrated circuits, as seen from above*



*Fig. A.2 Pin numbering of the 634SS2 Hall effect IC, as seen from above*



*Fig. A.3 Pin connections and symbols of semiconductor devices used in this book*



## SPECIAL NOTE FOR READERS IN USA

The ZX81 is available in the USA under that name or as the Timex-Sinclair TS 1000. When this book was in preparation both the ZX Spectrum and Jupiter Ace were imminently due for launching in America.

The circuits in this book have *not* been tested with the versions of these machines marketed in the USA, but providing the USA versions have not been radically changed in those details which relate to the operation of the add-ons, it is thought that the reader should not experience any difficulties in this direction.

Almost all the integrated circuits specified in the projects originate from the USA, so there should be no difficulties in obtaining these. Below are brief details of components which may not be so readily available:

BPX65 photodiode: planar 1 mm sq. silicon PIN photodiode, in TO18 case with glass window. Almost any photodiode can be substituted for this.

C106 p-gate thyristor: since this is operated at low voltage, almost any other type will do.

MKY7C38E: any light-dependent resistor, with dark resistance about 300 k and sunlight resistance 100 ohms. One with small diameter is preferred.

ZTX300 transistor: any npn transistor with gain of 100 or more and able to take a 500 mA collector current.

311 is the LF311 comparator IC.

7611 is the ICL7611 CMOS operational amplifier.

Please note overleaf is a list of other titles that are available in our range of Radio, Electronics and Computer Books.

These should be available from all good Booksellers, Radio Component Dealers and Mail Order Companies.

However, should you experience difficulty in obtaining any title in your area, then please write directly to the publisher enclosing payment to cover the cost of the book plus adequate postage.

If you would like a complete catalogue of our entire range of Radio, Electronics and Computer Books then please send a Stamped Addressed Envelope to:

BERNARD BABANI (publishing) LTD  
THE GRAMPIANS  
SHEPHERDS BUSH ROAD  
LONDON W6 7NF  
ENGLAND

205	First Book of Hi-Fi Loudspeaker Enclosures	95p
221	28 Tested Transistor Projects	1.25p
222	Solid State Short Wave Receivers for Beginners	1.25p
223	50 Projects Using IC CA3130	1.25p
224	50 CMOS IC Projects	1.35p
225	A Practical Introduction to Digital IC's	1.25p
226	How to Build Advanced Short Wave Receivers	1.95p
227	Beginners Guide to Building Electronic Projects	1.95p
228	Essential Theory for the Electronics Hobbyist	1.95p
RCC	Resistor Colour Code Disc	20p
BP1	First Book of Transistor Equivalents and Substitutes	1.50p
BP6	Engineers and Machinists Reference Tables	1.95p
BP7	Radio and Electronic Colour Codes and Data Chart	40p
BP14	Second Book of Transistor Equivalents and Substitutes	1.75p
BP24	52 Projects Using IC741	1.25p
BP27	Chart of Radio Electronic Semiconductor and Logic Symbols	50p
BP32	How to Build Your Own Metal and Treasure Locators	1.95p
BP33	Electronic Calculator Users Handbook	1.50p
BP34	Practical Repair and Renovation of Colour TVs	1.25p
BP36	50 Circuits Using Germanium, Silicon and Zener Diodes	1.95p
BP37	50 Projects Using Relays, SCRs and TRIACs	1.95p
BP39	50 JFET Field Effect Transistor Projects	1.75p
BP40	Digital IC Equivalents and Pin Connections	3.50p
BP41	Linear IC Equivalents and Pin Connections	3.50p
BP42	50 Simple L.E.D. Circuits	1.50p
BP43	How to Make Walkie Talkies	1.95p
BP44	IC555 Projects	1.95p
BP45	Projects in Opto Electronics	1.95p
BP47	Mobile Discotheque Handbook	1.35p
BP48	Electronic Projects for Beginners	1.95p
BP49	Popular Electronic Projects	1.95p
BP50	IC LM3900 Projects	1.35p
BP51	Electronic Music and Creative Tape Recording	1.95p
BP52	Long Distance Television Reception (TV-DX) for the Enthusiast	1.95p
BP53	Practical Electronic Calculations and Formulae	2.95p
BP55	Radio Station Guide	1.75p
BP56	Electronic Security Devices	1.95p
BP57	How to Build Your Own Solid State Oscilloscope	1.95p
BP58	50 Circuits Using 7400 Series IC's	1.75p
BP59	Second Book of CMOS IC Projects	1.50p
BP60	Practical Construction of Pre-amps, Tone Controls, Filters & Attn	1.45p
BP61	Beginners Guide to Digital Techniques	95p
BP62	Elements of Electronics - Book 1	2.25p
BP63	Elements of Electronics - Book 2	2.25p
BP64	Elements of Electronics - Book 3	2.25p
BP65	Single IC Projects	1.50p
BP66	Beginners Guide to Microprocessors and Computing	1.75p
BP67	Counter Driver and Numerical Display Projects	1.75p
BP68	Choosing and Using Your Hi-Fi	1.65p
BP69	Electronic Games	1.75p
BP70	Transistor Radio Fault Finding Chart	50p
BP71	Electronic Household Projects	1.75p
BP72	A Microprocessor Primer	1.75p
BP73	Remote Control Projects	1.95p
BP74	Electronic Music Projects	1.75p
BP75	Electronic Test Equipment Construction	1.75p
BP76	Power Supply Projects	1.75p
BP77	Elements of Electronics - Book 4	2.95p
BP78	Practical Computer Experiments	1.75p
BP79	Radio Control for Beginners	1.75p
BP80	Popular Electronic Circuits - Book 1	1.95p
BP81	Electronic Synthesiser Projects	1.75p
BP82	Electronic Projects Using Solar Cells	1.95p
BP83	VMOS Projects	1.95p
BP84	Digital IC Projects	1.95p
BP85	International Transistor Equivalents Guide	2.95p
BP86	An Introduction to Basic Programming Techniques	1.95p
BP87	Simple L.E.D. Circuits - Book 2	1.35p
BP88	How to Use Op Amps	2.25p
BP89	Elements of Electronics - Book 5	2.95p
BP90	Audio Projects	1.95p
BP91	An Introduction to Radio DX'ing	1.95p
BP92	Electronics Simplified - Crystal Set Construction	1.75p
BP93	Electronic Timer Projects	1.95p
BP94	Electronic Projects for Cars and Boats	1.95p
BP95	Model Railway Projects	1.95p
BP96	C.B. Projects	1.95p
BP97	IC Projects for Beginners	1.95p
BP98	Popular Electronic Circuits - Book 2	2.25p
BP99	Mini Matrix Board Projects	1.95p
BP100	An Introduction to Video	1.95p
BP101	How to Identify Unmarked IC's	65p
BP102	The 6800 Companion	1.95p
BP103	Multi-Circuit Board Projects	1.95p
BP104	Electronic Science Projects	2.25p
BP105	Aerial Projects	1.95p
BP106	Modern Op Amp Projects	1.95p
BP107	30 Solderless Breadboard Projects - Book 1	1.95p
BP108	International Diode Equivalents Guide	2.25p
BP109	The Art of Programming the 1K ZX81	1.95p
BP110	How to Get Your Electronic Projects Working	1.95p
BP111	Elements of Electronics - Book 5	3.50p
BP112	A ZX80 Workshop Manual	2.75p
BP113	30 Solderless Breadboard Projects - Book 2	2.25p
BP114	The Art of Programming the 16K ZX81	2.50p
BP115	The Pre-Computer Book	1.95p
BP116	Electronic Toys Games and Puzzles	2.25p
BP117	Practical Electronic Building Blocks - Book 1	2.25p
BP118	Practical Electronic Building Blocks - Book 2	2.25p
BP119	The Art of Programming the ZX Spectrum	2.95p
BP120	Audio Amplifier Fault Finding Chart	65p
BP121	How to Design and Make Your Own PCBs	2.25p
BP122	Audio Amplifier Construction	2.25p
BP123	A Practical Introduction to Microprocessors	2.25p
BP124	How to Design Electronic Projects	2.25p





# BERNARD BABANI BP124

---

## Easy Add-on Projects for Spectrum, ZX81 & Ace

- This book describes how to build a number of electronic projects which you can use with your Spectrum, ZX81 or Jupiter Ace microcomputer.
  - The projects include a Pulse Detector, Picture Digitiser, Five-key Pad, Model Controller, Bleeper, Lamp Flasher, Light Pen, Magnetic Catch, Lap Sensor, Photo-flash, Games Control and six more projects that make up a Weather Station.
  - All the projects are fairly simple and inexpensive to construct. The most complicated part, the Address Decoder, is constructed as a separate item that can then be used with any of the projects.
  - Once built, the projects are easy to operate and a simple program or two is included to get you started. Of course, those readers who are more experienced at programming can have a lot of fun in writing elaborate programs for these projects, but the beginner can start with a short program and perhaps add extra features later.
- 

ISBN 0-85934-099-6

£2.75



9 780859 340991