TALKING ELECTRONICS

$2.20
$3.00 NZ

Issue No 14

CONTINUITY TESTER

* INPUT/OUTPUT BOARD FOR THE TEC
* MICROCOMP-1 PART II

TO DECK SOCKET ON TEC

TE GUITAR PRACTICE AMP

CRYSTAL OSCILLATOR   CO-ORDINATOR   GUITAR PRACTICE AMP

TALKING ELECTRONICS

Editorial...

Vol. 1   No: 14.

A week may be a long time in politics, but so it is too in electronics.

Today's super seller may be tomorrows forgotten wonder.

Take the rise and fall of the Personal Computer.

From a tentative start, the PC rose rapidly to become one of the most popular sales lines ever. The market seemed insatiable. As fast as a new model was released, it was snapped up.

But then the crunch came. The bubble burst and saturation occured. The boom of last week turned into the fizz of this week.

How or why the abrupt end occured, nobody seems to know. Everyone got caught. The biggest losers were the importers. With thousands of boxed computers lying in their warehouses, many have just let them sit; waiting and hoping for a change in the market.

Even though we can buy a model with twice the features, cheaper than last year, nobody wants them.

Maybe it was the realisation that the personal computer is really little more than a glorified games machine, that started a nation-wide re-think. Or was it the improvement in graphics on the arcade machines, that made their graphics look so purile?

In any case, the secondary effect of the slump has been to create an absolute glut of chips on the open market.

All those used in one or more of the popular computers are now available by the million!

To clear this back-log, chip production has slowed to a mere trickle and prices have fallen to a level below the actual cost of manufacture.

A 64k DRAM can now be bought in small quantities for 4c less than production cost!

Sadly, the slump is exactly as we predicted.

Computers failed to interface to the real world and although they appeared exciting within themselves, they could not be readily connected to external appliances and gadgets.

Had someone produced a universal interface consisting of say a robot arm, a telephone interface and an alarm system, the capability of the PC would have been extended enormously and its popularity would still be with us.

Until someone comes up with a useful adaptor like this, I cannot see things improving.

We at TE are just as keen as you to see a turn around, as the TEC computer and Microcomp are ideally suited to interfacing to a robot arm.

The only thing holding things up is the non-availability of gearboxes and motors etc.

Let's hope someone has the foresight to produce a range of mechanical units at an economical price so that our ideas for automatics and robotics can come to fruition.

As soon as something comes along, we will be the first to let you know.

Colin Mitchell

PUBLISHER

INDEX

For all those who have asked to see a shot of me, I have reluctantly included this recent pose. See, I'm just a normal balding, existentialist.
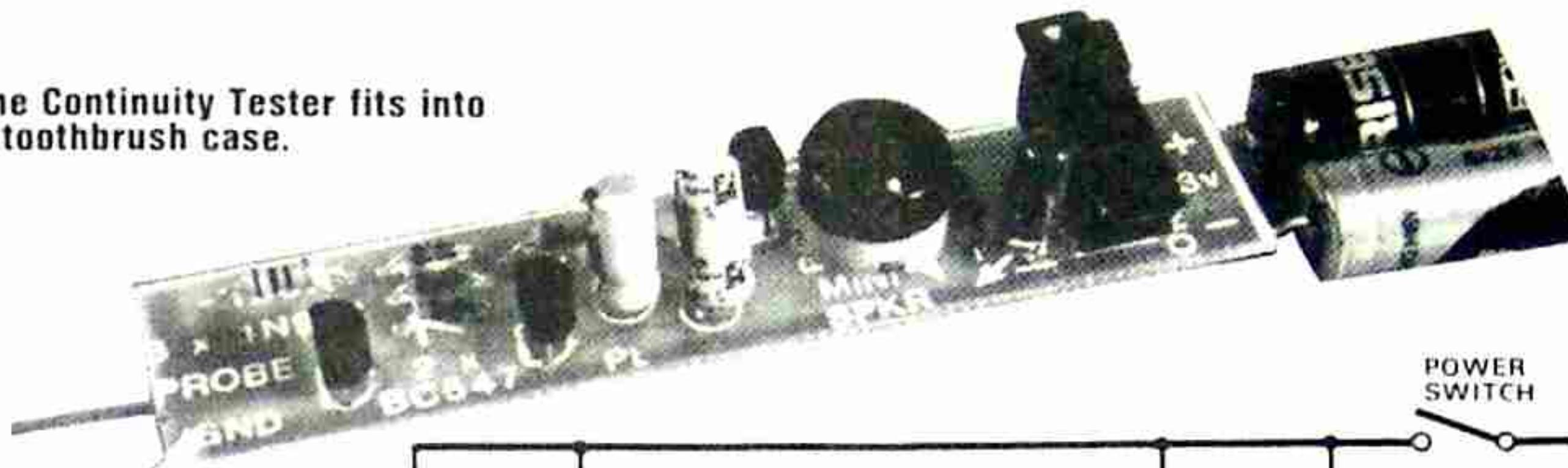
**COVER PHOTO:** The cover photo shows a Z-80 sitting on a micro-photograph of the internal workings.

CONTINUITY TESTER
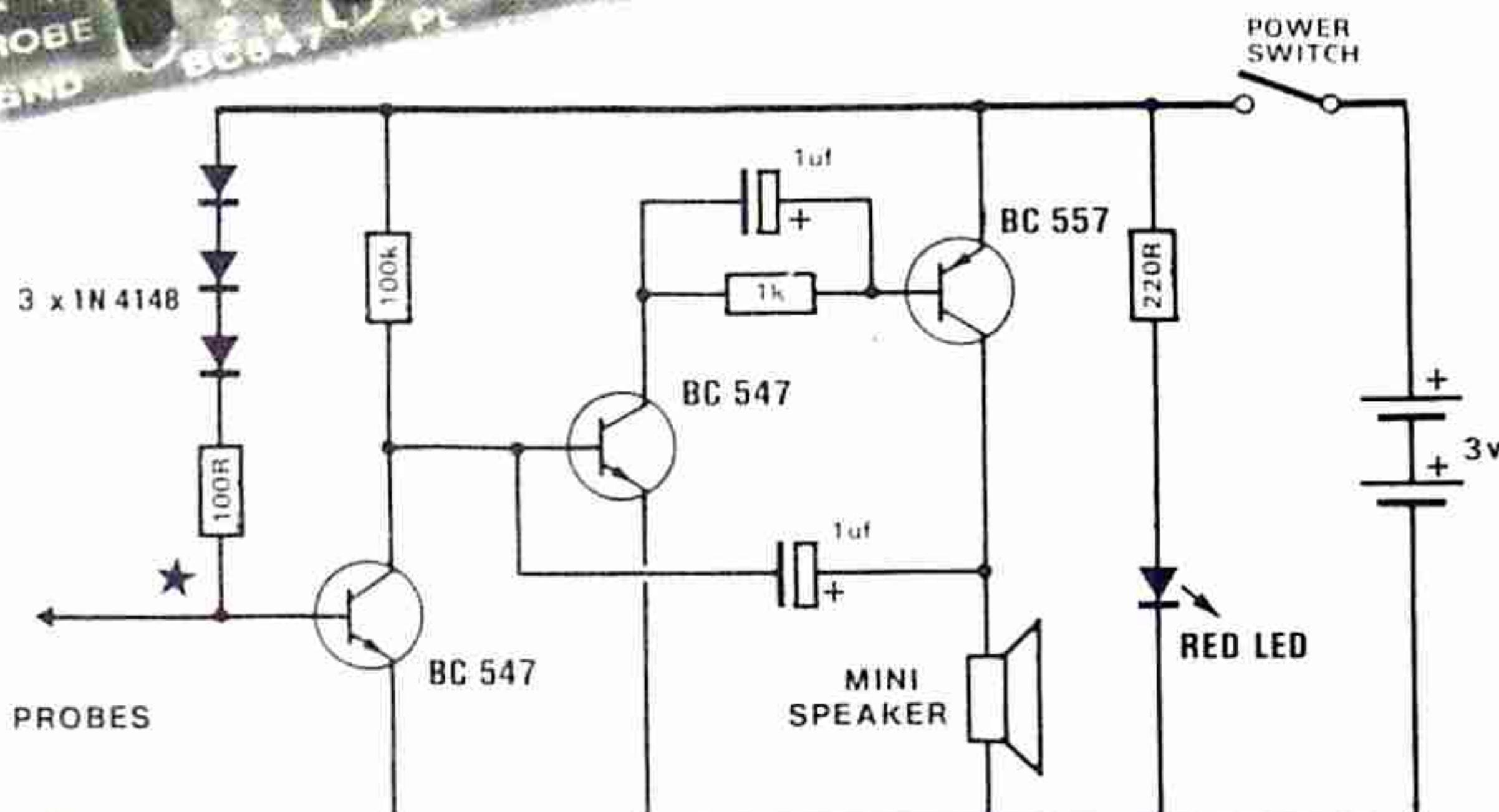
THE THIRD IN OUR 'TEST PROBE' TRIO. . .

The Continuity Tester fits into a toothbrush case.

★ See P. 8 for modification.

3 x 1N 4148

PROBES

BC 547

BC 547

BC 557

1uf

1k

100k

100R

220R

1uf

MINI SPEAKER

RED LED

POWER SWITCH

3v

CONTINUITY TESTER CIRCUIT

The continuity tester is the third and final piece of test equipment in the 'test probe' trio.

On the face of it, a continuity tester seems pretty unimportant and you may be tempted to use a multimeter for the job.

But 'horses for courses' is what I always say. The right tool for the application. A multimeter might be alright for some applications but if you are troubled with a nasty fault in a digital project, the continuity tester is faster, better and easier to use than anything else.

It is specially designed for the job and has three very interesting features.

Firstly it gives an audible indication so that you can keep your eyes on the job. This is important when making a continuity check between adjacent pins of a 40 pin IC or on a closely packed bus network such as the data or address bus.

You cannot afford to take your eyes off the board as either the probe will slip off the track or you will miss one of the lines!

Secondly, its response-time is very brief so that you can make contact in a sweeping or stroking motion so that a number of lines can be swept in the one operation.

And thirdly, the continuity tester responds only to a definite short circuit or one in which the resistance is 150 ohms or less.

It will not respond at all to values above 180 ohms and most important it will not respond to the voltage drop across a diode.

This is where the multimeter falls down.

When you are measuring between some of the lines in a digital circuit, the impedance will be quite low or a protection diode will be in the circuit.

PARTS LIST

1 - 100R ¼watt
1 - 120R (for mod.)
1 - 220R
1 - 1k
1 - 100k

2 - 1uF electro

3 - 1N 4148 diode
2 - BC 547 transistors
1 - BC 557 transistor
1 - 5mm red LED
1 - Mini speaker

1 - DPDT slide switch (or SPDT)
2 - AAA cells

1 - paper clip
10cm tinned copper wire
50cm Hook-up flex

1 - CONTINUITY TESTER PC BOARD

The resultant reading on a multimeter will be low (nearly full-scale deflection) but it will be difficult for you to determine if the meter is picking up the voltage drop across a diode or detecting a very low resistance. Apart from this, the time taken for the needle to swing across to its final reading, makes the multimeter approach very slow.

The continuity tester eliminates these problems.

We have found it invaluable for diagnosing the TEC's that have come in for repair. Most of the problems have been shorts between lands or open connections in one of the buses.

This is how we use the tester:

Once we have established that the fault lies in the trackwork (all the chips have been replaced and the system remains dead) we test each pin of the Z-80 against every other pin of the chip. This is actually 40x40 tests and by using the continuity tester it is simplified to only a few operations.

Firstly place the wander lead on pin 1. With the tester turned ON, start at the top of the other side of the chip and quickly wipe the probe down the 20 pins. Repeat for pins 20 to 1. The only time you will hear a beep is when the two probes touch. If a short beep is heard at any stage during the test you should go back and determine if the two lines are joined or if a fault exists.

Continue this procedure with pins 2, 3, 4 etc and very soon you will have covered all 40 pins.

By doing this you will have also covered the bus lines on the EPROM and RAM, however they can be individually checked if you like.

The next part of the diagnosis is to check the continuity of each line in the data and address bus. For this you will need a circuit diagram and pin-out data. Start at data line D0 and check D0 on the EPROM and also the RAM. If a tone is heard, the line is continuous.

It is essential to carry out all these checks as you don't know the exact location of the fault and most faults will be found with a systematic approach.

HOW THE CIRCIT WORKS

As we have mentioned above, the circuit detects resistance values of 150 ohms or less between the probes and allows an oscillator to turn ON and produce a tone in the mini speaker.

A LED is also included on the board to indicate when the unit is switched ON as the electronics consume about 2-4mA and thus the battery would eventually go flat if the tester were left on for long periods.

Actually the circuit doesn't detect resistance at all. It detects threshold voltage across the base-emitter junction of a gating transistor.

When the tester is in the "rest" state, the first transistor is turned ON and this inhibits the oscillator.

It gets its turn-on voltage via the 100R resistor. The 3v supply is passed through 3 diodes which drop a total of 1.9v, leaving 1.1v for the base bias.

When the transistor is turned ON, the base-emitter voltage (the junction voltage) is .7v and thus .4v is dropped across the 100R resistor. This means we have only .4v leeway for the batteries and when they drop to below 2.6v, the tester will fail to work. That's why we have to conserve battery voltage as much as possible by putting an indicator LED on the project to prevent it being left on.

0.4v across the 100R resistor delivers 4mA into the base of the gating transistor and this keeps the oscillator circuit in the OFF state.

When a resistance of 150 ohms or less is placed between base and emitter, the voltage on the base falls sufficiently to turn the transistor OFF.

This allows the 2-transistor feedback oscillator to come into operation and produce a tone in the speaker.

A diode placed between the base and emitter of the first gating transistor will have no effect on the circuit as it will allow .6v to .7v to be present across the probes and thus the first transistor will not change state. The voltage must drop to .5v or less for the circuit to change and this requires a resistance of about 200 ohms.

The two transistor feedback oscillator is set into motion by the 100k base bias resistor.

This turns on the first transistor and thus its collector voltage falls. The collector is connected to the base of the second transistor in the oscillator and this is also turned on.

The result of this action is to raise the voltage of the collector and as you can see, the mini speaker is connected to this lead. Thus a voltage appears across the speaker.

Also connected to the collector is a 1uF electrolytic and it is presently in the discharged state.

As the voltage on the collector rises, it pulls the electrolytic up with it and since it is uncharged, the other lead is pulled up too.

This causes the base of the first transistor to be turned on hard and very soon we have a situation where both transistors are SATURATED.

The next point to understand is the voltage across the electrolytic under discussion. Its negative will be at .65v and its positive will be at about 2.4v. The electro has effectively been stretched between base and rail and its important to understand that the base voltage cannot rise above .65v.

The circuit sits in this condition while the electrolytic gradually charges a little more and this causes the base of the first transistor to turn off slightly.

This is passed to the second transistor which also begins to turn off.

In a short period of time the voltage on the collector falls slightly and this drop is transferred directly the first transistor via the electrolytic. Very soon we have a situation where the first transistor is turning the second off and the second is turning the first off. Both are now completely OFF and the 100k resistor takes over to start the process again.

Each time the circuit "cycles" the speaker produces a 'click' and since these clicks are produced in rapid succession, the result is a pleasant tone.

CONSTRUCTING THE TESTER

All the components are mounted on a small PC board that is designed to fit into a toothbrush case. There are a number of suitable cases and even the small size will fit the board. At first we thought the soft type of case would not be suitable but after Paul tried it, we found it was the best. The soft plastic is more durable and will not fracture if dropped or bumped. The rigid styrene cases tended to crack very easily and one of ours was crushed under foot when it fell on the floor!

The case is the first item to purchase and it will give you a guide as to the maximum height allowable for the components. If some of the parts are too high, they can be bent over and it is important to know about this before you start.

Next you need to determine the type of switch you will be using. The board will take two sizes: a mini single pole double throw or a mini double pole double throw.

Depending on which one you intend to use, the appropriate holes must be drilled in the PC board.

Once this is done, the components can be mounted.

Start assembly at one end of the board and fit each part as you come to it. The mini speaker can be inserted either way around as it is not polarity sensitive but the LED, transistors, diodes and electrolytics must be fitted as shown. If you are not sure about the placement, don't guess, refer to data or get someone to assist you.

The probe is made from a paper clip that has been straightened at one end and bent into a hook at the other so that a strong solder connection can be made.

The two batteries are soldered to the board via short lengths of tinned copper wire. This will keep them firmly together and keep the whole assembly rigid.

The wander lead has either an alligator clip or E-Z clip attached and this allows it to be connected to one rail of the project under test so that the probe can be used to go over the rest of the board in the hunt for the fault.

When everything has been soldered in place, slide the switch ON and the LED will illuminate. Touch the two probes together and the oscillator will emit a tone.
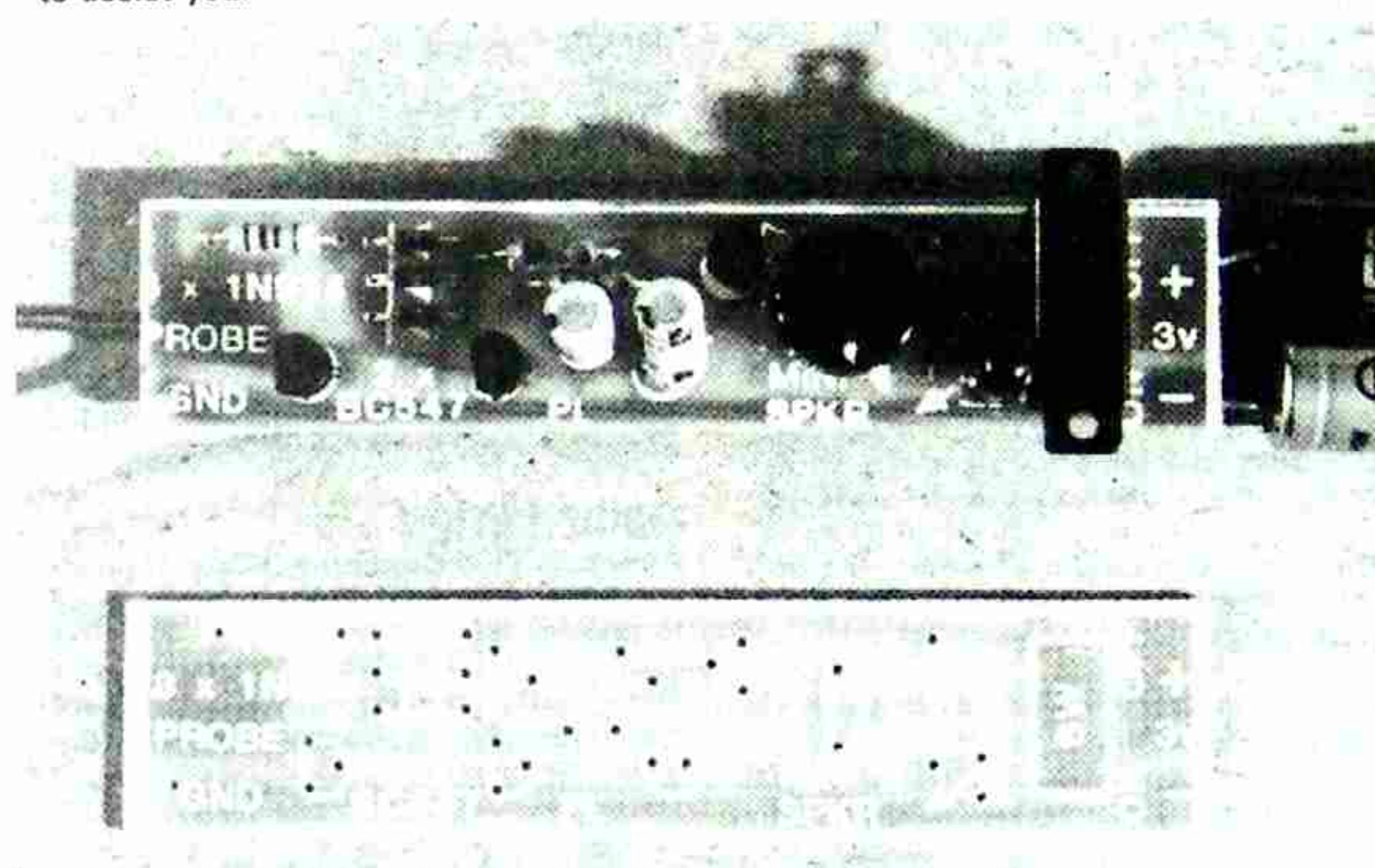
TESTING THE UNIT

You will require a diode, 180R resistor and a 220R resistor.

Place the probes across the diode, firstly one way then the other. The tone should not be heard.

Place the probes across the 180R resistor. The tone should be emitted. Place the probes across the 220R resistor. The tone should not be emitted.

You may find the tester will operate on a resistor which is one value higher or lower than this. The actual value will depend on the battery voltage and the base-emitter junction voltage of the

A close-up of the Continuity Tester and PC board before the modification to the front end. See details of this modification on page 8.
The tester can be housed in a tooth-brush case and the soft-type cases are the best as they don't crack. It can then be added to our two other pieces of test equipment to make a very valuable trio for testing digital projects, especially processor designed projects, as these will be the products of the future.

gating transistor. But don't worry too much about the actual turn-on and turn-off values as most bus lines in computers have resistors of about 470R in their lines and the tester will not detect resistance values above 330R.

IF IT DOESN'T WORK

If the Tester does not emit a tone when the two leads are touched together, follow these steps:

Remove the gating transistor (near the input probe) and turn the unit ON. This should allow the tone circuit to operate. If not, check the value of the 1k and 100k resistors and also their positions. Next check the 1uF electrolytics. Check also the BC 547 and BC 557 transistors. They cannot be swapped over. Finally check the speaker with a multimeter set to ohms to make sure it has continuity.

You can determine if the oscillator is jammed in the ON mode or OFF mode by taking a current reading across the switch. If the current is more than 20mA, it is jammed in the ON mode and this means the BC 547, in the oscillator, is conducting.

The transistor could be shorted between collector and emitter or the 1uF feedback electrolytic is open circuit.

Create a short between the base and emitter of the BC 547 to start the circuit into oscillation or replace the 1uF with 100n.

Remember, the mini speaker is a special 80R type and this value must be maintained for the circuit to work properly.

If the circuit is jammed in the OFF mode, either of the transistors in the oscillator may be open circuit or incorrectly fitted. Try new transistors.

If the circuit remains in the OFF state, remove the 1uF feedback electrolytic between the speaker and 100k and the circuit should turn ON.

If it does, the fault will lie in the electro being either a short circuit or very leaky.
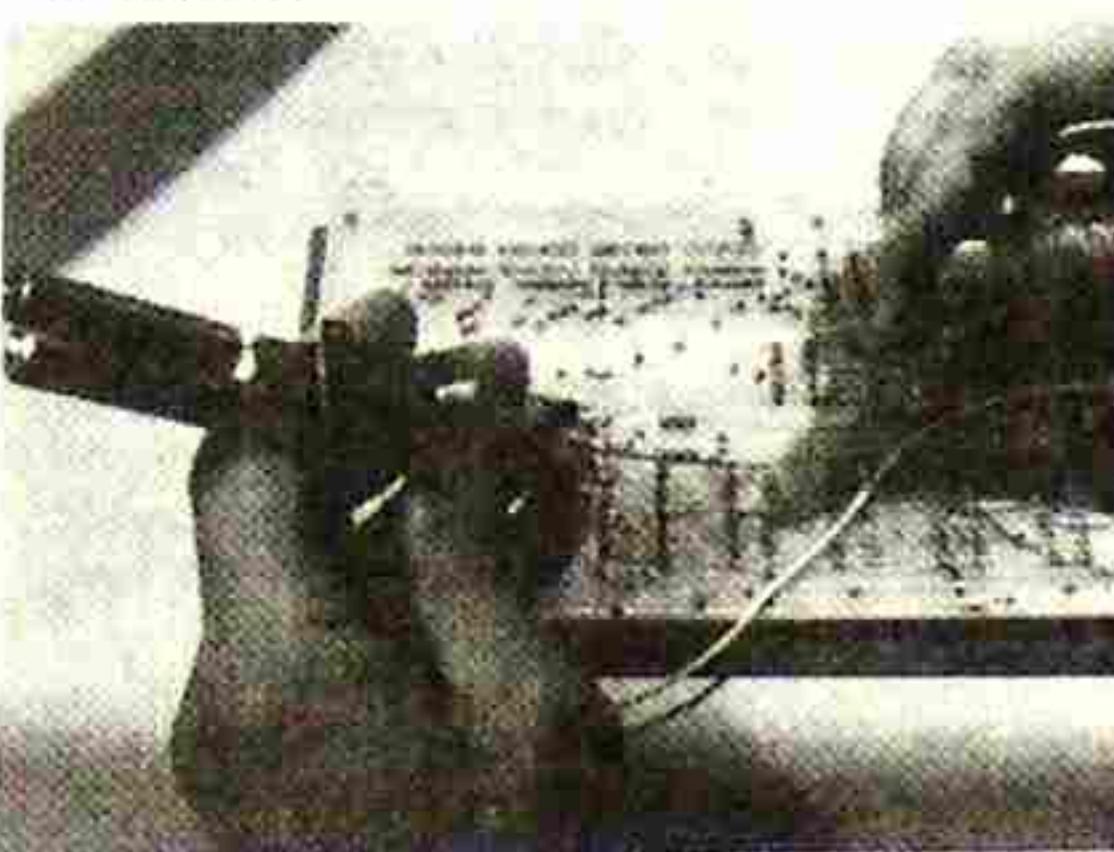
Once the tone circuit operates, the only other stage to be checked is the gating transistor.

If it fails to turn the tone off, the fault will lie in the base bias.

Check the voltage of the batteries. They should give at least .1v drop across the 100R resistor. If there is no voltage drop, check the voltage-dropping diodes. They should drop .65v each. Also check the supply voltage. It should be higher than 2.6v.

If the gating transistor still fails to turn the tone off, replace it. You may have damaged the base-emitter junction.

This should be sufficient to get the Tester working. If not ask another hobbyist to help you.

Using the continuity tester on the BUS lines of the TEC.

HOW TO USE THE TESTER

The project you are going to test must be switched OFF and no part of it should have any voltages present.

This is because the input of the Tester connects directly to the base of a transistor and any voltage over .7v will destroy the junction.

Switch the Tester ON and the indicator LED will illuminate. Touch the two probes together and the tone will be emitted. This is the tone you will be listening for during the tests.

It is important to have a set plan of attack as lots of tests will be required for even the simplest of circuits and a logical approach will prevent you going over the same area twice or missing a test.
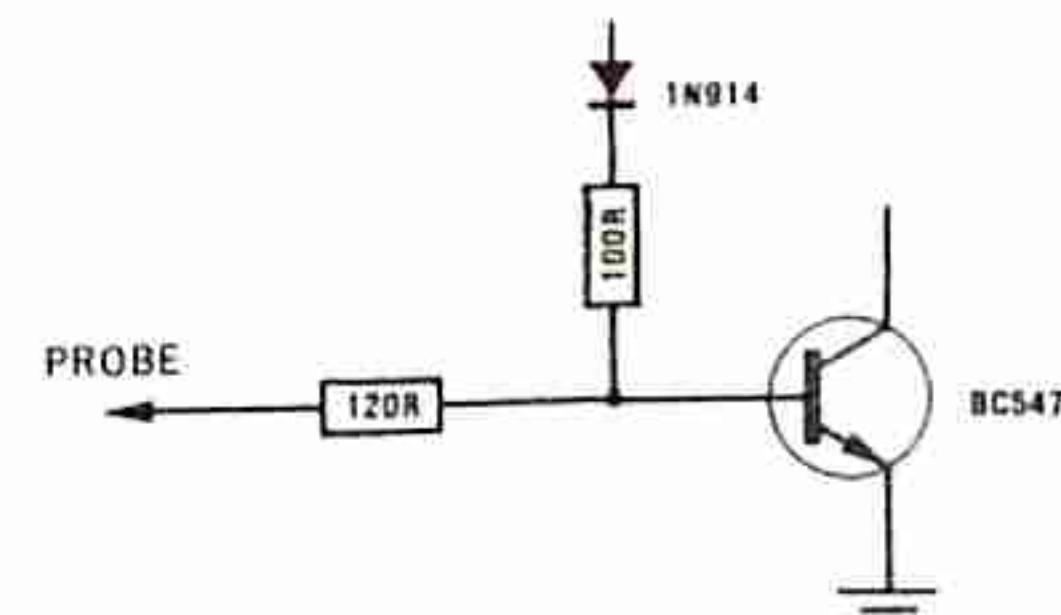
MODIFICATION

After using the Continuity Tester for over 3 months, with total success, we decided to present it in TE.

Then Paul came up with a simple but clever modification that would allow the

probe to detect down to 47 ohms and also provide a safety feature for the first transistor.

It is simply the addition of a 120 ohm resistor between the base of the first transistor and the probe. Firstly this will act as a limiting resistor to prevent damaging the base of the transistor, should the probe be placed across power rails of a project when the voltage is present.
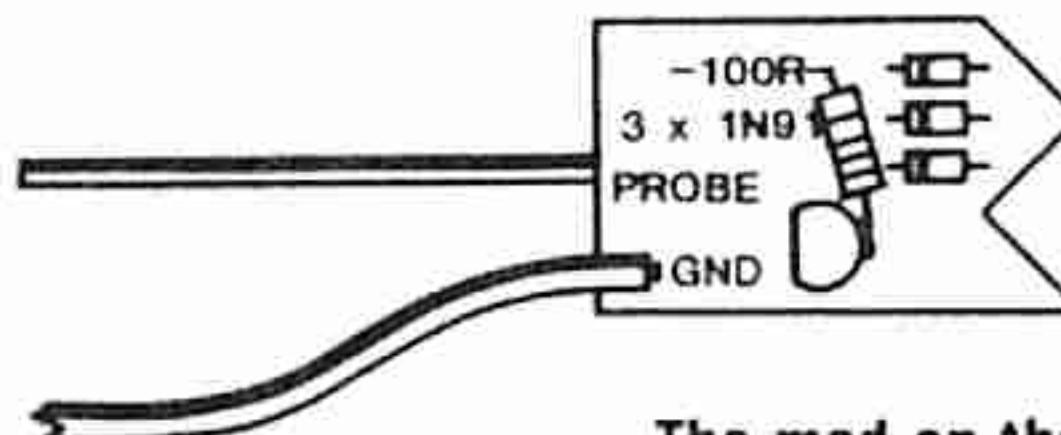
The 120R is added in series with the probe to protect the first transistor.

Secondly it reduces the input range so that the probe detects values from zero ohms to about 47 ohms.

Whereas we have been careful to turn off the supply to any project under investigation, the idea of directly connecting the base to the probe is not wise. It means any voltage above about .7v will have the capability of damaging the input transistor.

This mod overcomes the problem and makes it more robust in use.

The modification is carried out as follows:
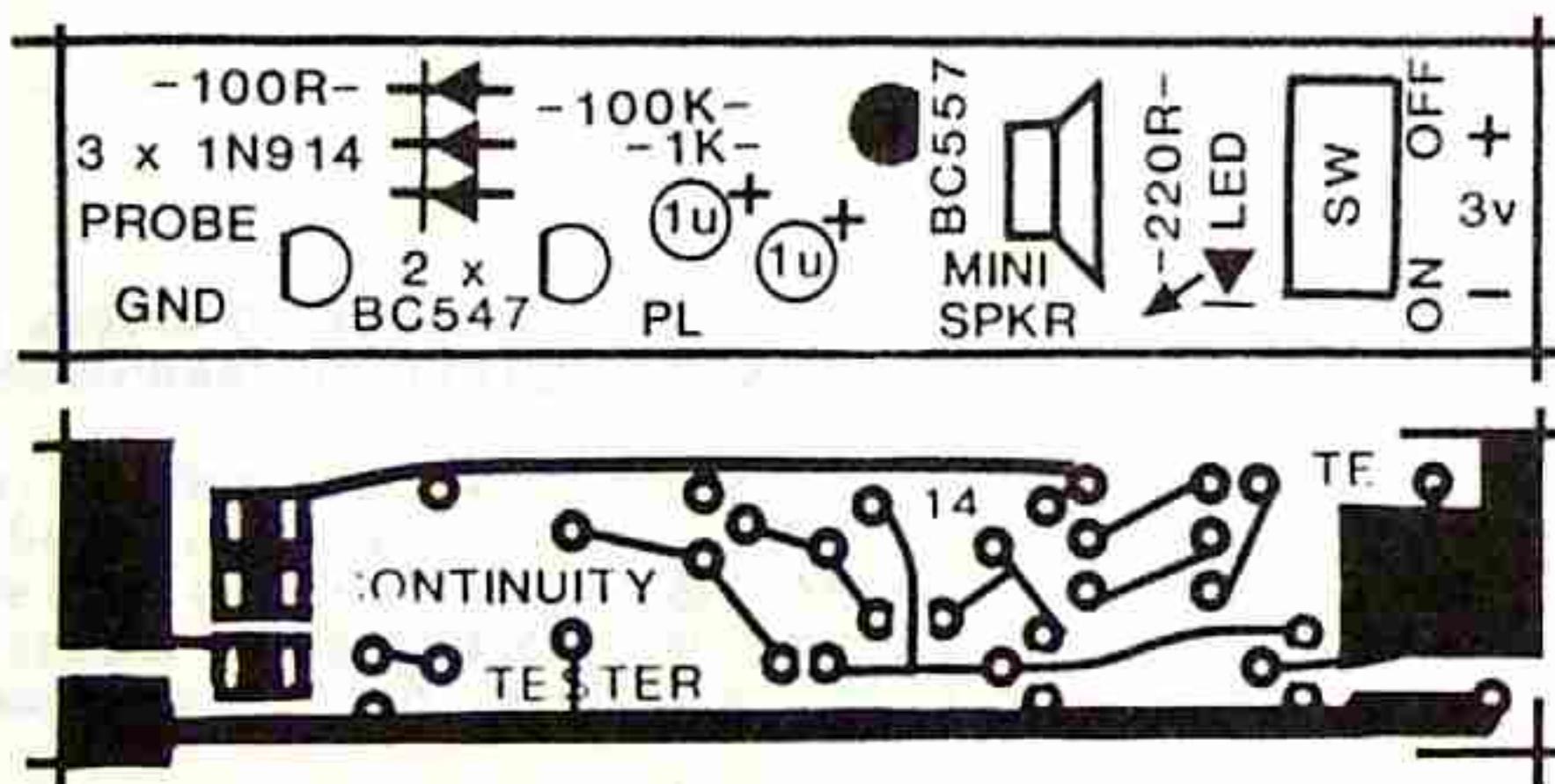
The mod on the front-end of the tester. The 120R is on the underside of the board.

Desolder the probe end of the 100R resistor and take it directly to the base of the transistor. Refer to the diagram for this. Cut the track between the base of the same transistor and the probe. Solder a 120R resistor between the base and the probe, either on the underside of the board or on top, so that it looks as neat as possible.

You may need to use a higher or lower value, according to the characteristics of your probe and experimenting will determine this.

This will make the probe relatively indestructible and I hope you find as much use for it as we have.
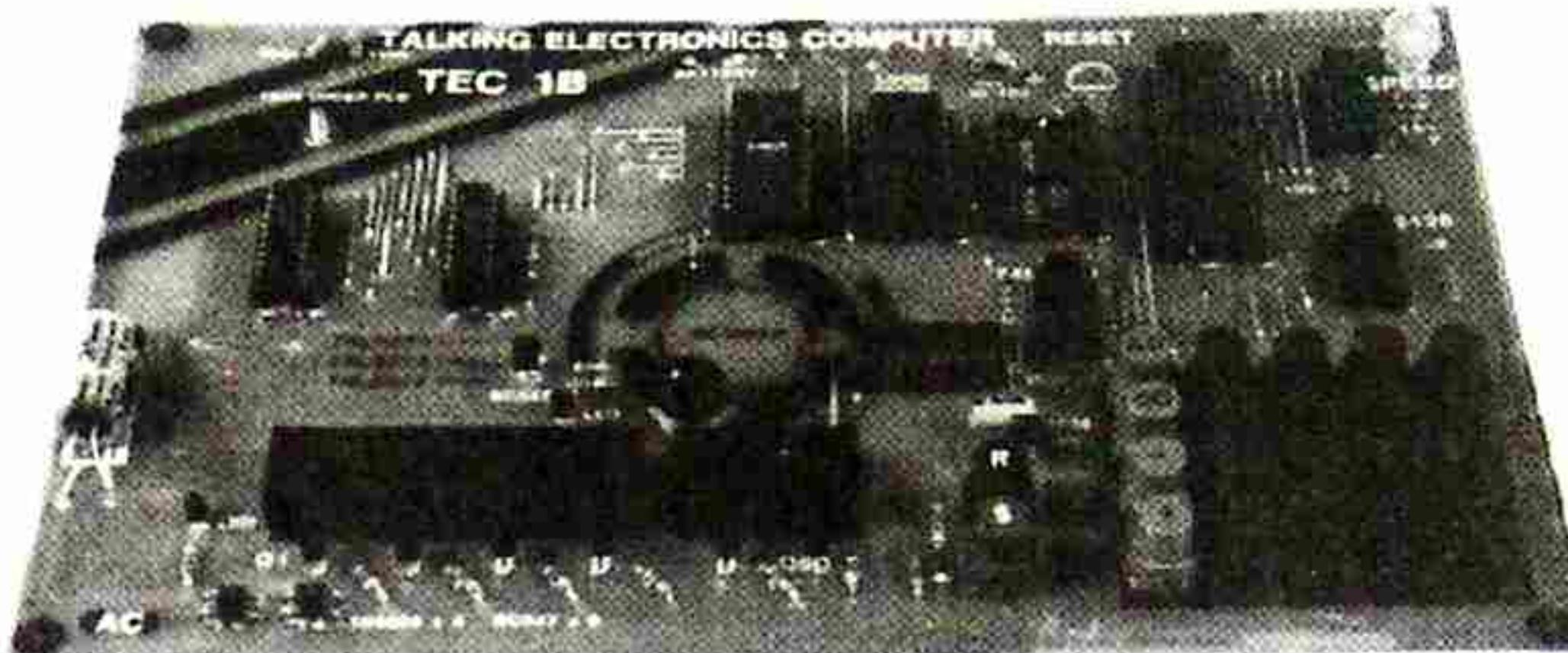
Quite honestly, it has found cracks and dry joints that the eye missed and has fixed at least 10 TEC's in the past 2 months. I'm sure you will like it.

TEC-1A & 1B

Kit of parts: $90.60
PC Board: $24.30
Complete: $114.90

TALKING ELECTRONICS COMPUTER
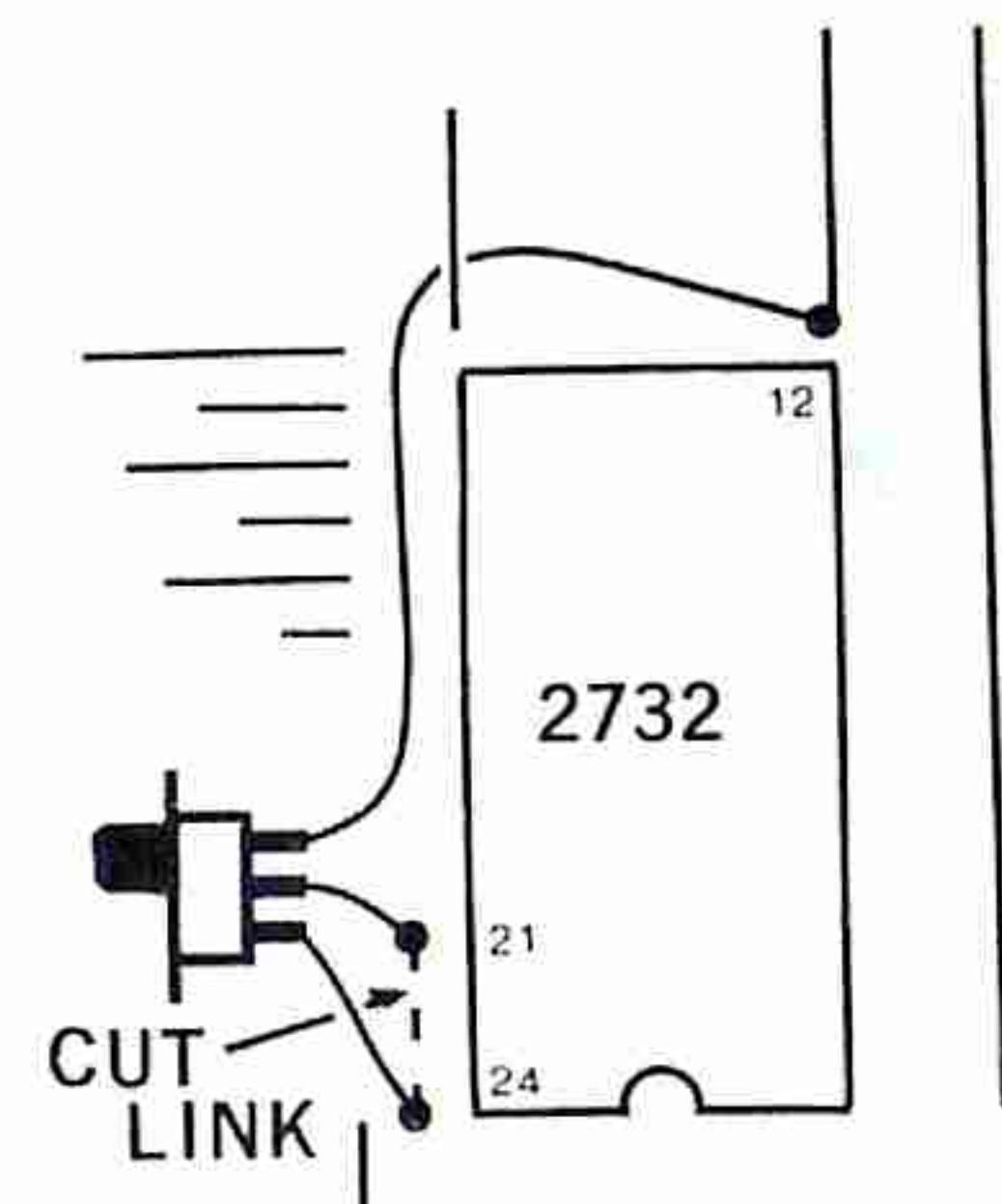
TEC 1A's can be converted to TEC 1B's by ading a push button, a 47k resistor and a diode. When you update to MON 2, the SHIFT function allows INSERT and DELETE and a number of other commands.

PART V

Features in this article:
★ Crystal Oscillator
★ Input/Output Module

TEC 1B with SHIFT KEY FITTED.

This is the fifth article on the TEC and quite frankly we have only just scratched the surface up to now.

The more ideas you try, the more you realise the potential of programming.

We have received a number of programmes for the 7-segment displays as well as the 8x8. These have been included in this article and also a few more hints on programming in general.

But before we get onto the programmes, there are a number of loose ends we have to tidy up, to bring the documentation up to date.

So far there have been 4 different models of the TEC and although the changes have been slight, they have not been put down on paper.

As far as the software is concerned, all models are compatible as the only modifications have been in the hardware.

The output latches have been changed from 8212's to 74LS273's, the 2200uF filter electrolytic changed to 1000uF and the 7805 mounted under the board so that its leads cannot be bent or broken.

The rest of the design remains substantially the same with the only addition being a shift button near the keyboard.

This button allows the keys to have a second function and we have already described these in issue 13.

Kits are now supplied with both the 1B ROM and also MON 2 ROM. It is possible to fit both programs into a single 2732 and to select either one program or the other requires a slide switch to take pin 21 HIGH or LOW. With this you can get the best of both monitors.

The computer can be switched between one MONitor and the other by pressing the reset button and while it is pressed, the slide switch is changed. When the reset button is released, the other MON will come into operation.

The following is a reprint of an information sheet supplied with the latest kits:

THE 2732 MONITOR

Both MON 1B and MON 2 are in the same chip and is called MON 1B/2. The MON 1B program has been placed in the upper half of memory so that when it is placed in the TEC, the MON 1B section will run and the computer will display 0800. You can now access all the games, tunes and running letter routines as covered in issues 10, 11, 12 and 13.

The MON 2 routine is more advanced and does not contain any of the games. Instead it has a SHIFT routine that enables you to insert bytes into a program by shifting all the higher bytes, and the byte at the present address, up one location. And a delete function, as well as a number of other routines that have been covered in issue 13.

When you want to accesss the MON 2 program, a switch must be fitted to the board so that pin 21 can be taken to ground. This will enable the lower half of the 2732 to be brought into the system and thus run the MON 2 listing.

The diagram above shows how to fit the mini slide switch to the two halves of the link that has been cut as shown.

You can switch from one monitor to the other at any time by pressing reset and altering the switch.

If you are writing a program using the MON 1B, it is best to start at 0900, so that when (if) you want to use the INSERT or DELETE functions, you can change to MON 2, use the function and then change back to MON 1B.

Gradually you will realise it is best to use MON 2 for most of your programs.
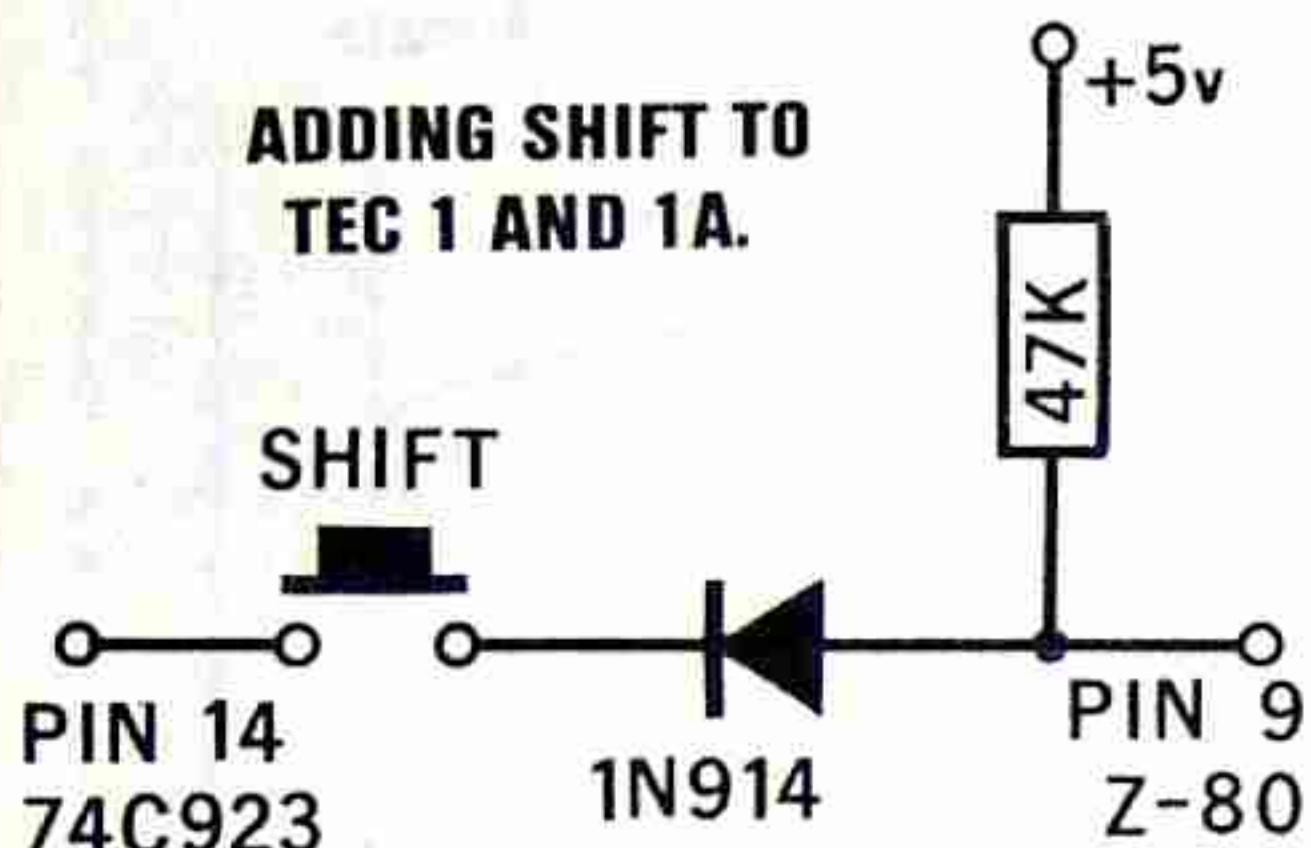
There are two major differences between MON 1B and MON 2. MON 1B uses a simple routine that places the value of a key directly into the accumulator, without firstly saving the value of the accumulator. Thus its original value is destroyed. MON 2 loads the key value into location **08E0** and thus your program must include an instruction that looks at this location for the value of the key.

Unless you load directly into the A register.

Simple programs designed for MON 1B will not run on MON 2 if they include a key press; unless they are altered accordingly.

The second difference is the start address for programming. MON 1B starts at 0800, while MON 2 starts at 0900. Programs written at 0800 cannot be successfully modified via the insert and delete functions as they will run into part of the scratchpad area for the MON 2 system.

The following diagram shows how to add the diode and resistor for the shift function. The diagram in issue 13 was not clear and this is an improvement:

ADDING SHIFT TO TEC 1 AND 1A.

SHIFT

PIN 14
74C923

1N914

PIN 9
Z-80

+5v

47K

TEC 1A/1B CONSTRUCTION HINTS:

The output latches for the latest TEC's are 74LS273's and the dotted link below each chip is fitted.

The 7805 regulator bolts directly under the PC board and a little thermal compound can be applied to assist heat transfer.

The small link from pin 4 of the 74LS138 IN/OUT decoder must be added. It can be cut later if expansion is required.

About 58 empty holes will be on the board after construction. Some provide for expansion while others are unused.

After the keys have been added and everything is operating satisfactorily, the letters and numbers can be applied to the tops.

Firstly clean the buttons with methylated spirits and apply the rub-down letters. Cover them with clear nail varnish to protect them. If you want to add another layer, wait for the first to dry, otherwise the letters will smudge!

NOTES ON THE 8x8 DISPLAY

The 8x8 has been modified to include sinking and sourcing transistors as described on P 27 of issue 12 and all kits now include 16 transistors and the necessary current limiting resistors.

This results in the LEDs being driven harder and increases the brightness of the display noticeably.

This is important when multiplexing as each LED will be turned on for only about one-eighth of the time and if sufficient current is supplied during this instant, the LED will appear to be on for the total period of time with an acceptable brightness.

We had an interesting fault in an 8x8 last week. It is interesting because the knowledge we gained applies to other projects where LEDs are driven in parallel.

A constructor built the 8x8 and was not happy with the output of about 3 of the LEDs.

He went to his local electronics shop and bought a few replacements.

After fitting them, he was quite surprised that they did not work at all! So he rang us. At this particular point in time we were not familiar with the fault and did not know how to advise him. So we suggested he call around with the project.

Some time later that day he arrived and we noticed the first difference was the colour of the LEDs he had used. They were less opaque than the rest and the crystal inside the LED could be readily seen. This did not disturb us as the light output of the LEDs was our prime concern.

When we tested it, sure enough; the 3 LEDs did not light up.

On measuring across the new LEDs with a multimeter set to low ohms, the voltage drop across the crystal was slightly higher than the rest. (When we are taking a measurement like this, the swing of the needle is being taken as a voltage drop. We are using the 3v suppy in the multimeter to provide the LED with voltage and the needle tells us the characteristic voltage drop across the crystal.)

We then got three LEDs from our stock and measured the characteristic voltage drop. It was exactly the same as the majority in the display and when we fitted them, the whole screen lit up perfectly.

The reason why the LEDs failed to illuminate was due to the higher voltage needed to turn them on. Even if this is 100mV or so, the result will be the LED will not turn on at all. (See the experiment in Stage-1, P 9)

It is important that LEDs are matched according to this characteristic voltage, for situations where they are placed in parallel. The 8x8 is one example as the LEDs are effectively in parallel when the whole screen is being illuminated in a non-multiplexed situation.

DISPLAYING LETTERS AND NUMBERS

The 7-segment display is quite a unique unit. It will display all the numbers from 0 to 9 as well as many of the letters of the alphabet.

There are only about seven letters that cannot be readily displayed and for these we will have to make a compromise.

The letter M is displayed as a small 'n', with a bar over the top. This corresponds to a feature in mathematics where a dot is placed over the first and last digits in a

number to indicate the number repeats. (This is called a recurring number or recurring fraction).

The letter W is displayed as a small 'u' with a bar over the top, for the same reason. The letter 'U' is displayed as a capital letter while V is a small 'u'.

The letter 'X' is displayed as part of a cross and Z is shown as two angles in opposite corners of the display, and looks quite readable.

The only letters which require interpretation are 'K' and 'Q'.

Ten other characters have also been included such as a question mark and 'equals' as well as a reverse bracket to assist in displaying mathematical problems.

A = 6F		? =	4D
B = E6		= =	84
C = C3		¬ =	04
D = EC		! =	38
E = C7		. =	10
F = 47		" =	0A
G = E3		; =	30
H = 6E		=	20
I = 28		= =	85
J = E8		) =	0F
K = 67			
L = C2			
M = 65			
N = 6B			
O = EB			
P = 4F			
Q = 3F		1 =	28
R = 44		2 =	CD
S = A7		3 =	AD
T = 46		4 =	2E
U = EA		5 =	A7
V = E0		6 =	E7
W = E1		7 =	29
X = 26		8 =	EF
Y = AE		9 =	AF
Z = C9		0 =	EB

TESTING A BLANK 2716 FOR FF's

After erasing an EPROM, such as a 2716, it is wise to make sure it is entirely blank before reprogramming it. The program that follows does just that. It does not inform you of the location or locations that do not contain FF, but rather the screen goes blank and stays blank if a location has not been fully erased.

If all locations contain FF, the TEC resets via the MONitor program to the start-up address (either 0800 or 0900). This program can be placed anywhere in RAM and will work with either MON 1 or MON 2.

- by James Doran. 3218

```
11 00 08
21 00 10
7E
FE FF
20 07
23
1B
7A
B3
20 F5
C7
76
```

As promised, a larger photo of the robot arm. If you have built anything like this, why not take a photo and send it in.

Your ideas, combined with others, will help us to present an article.
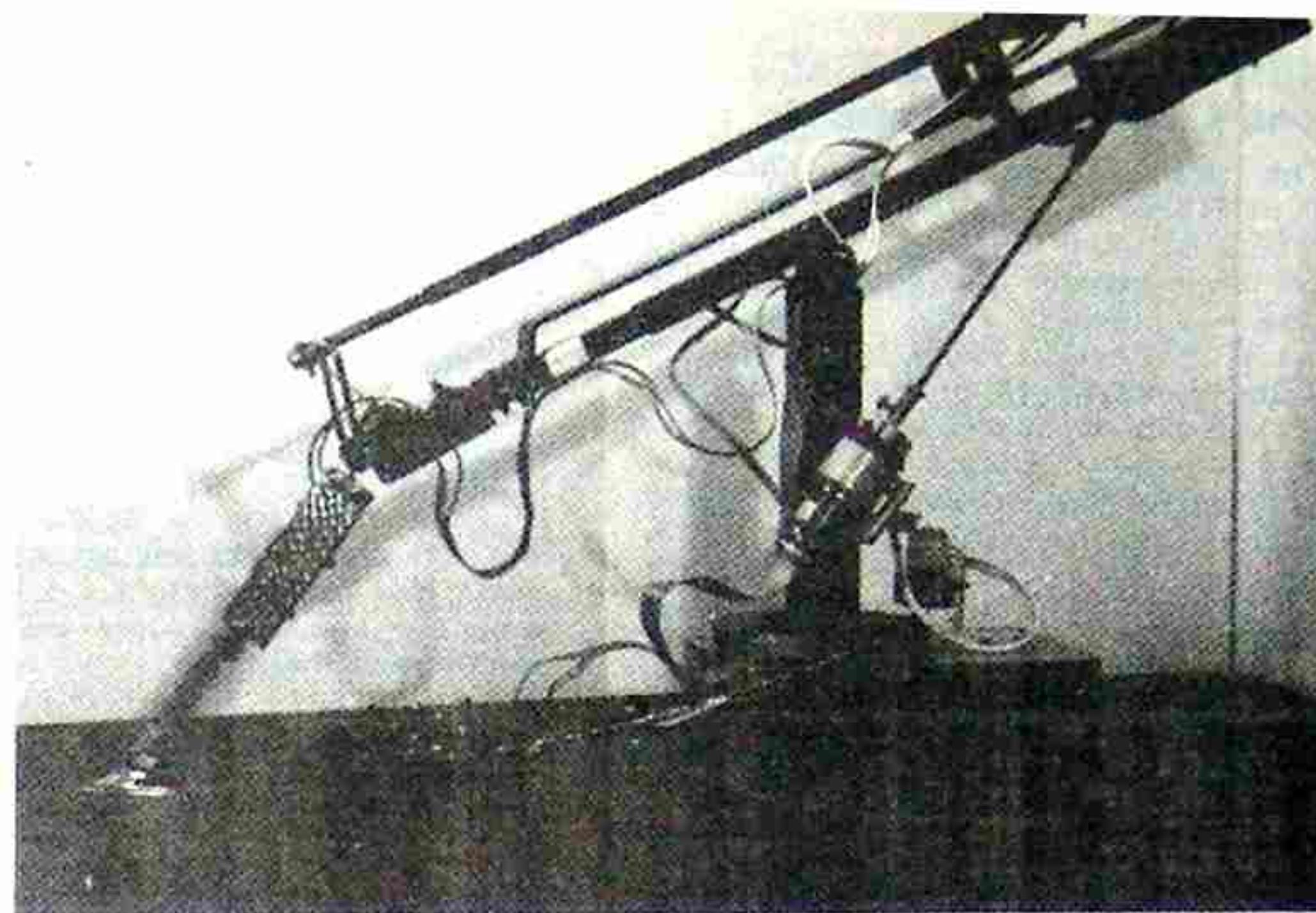
MON 2 HEX LISTING:

For those with the TEC 1B and an EPROM BURNER, here is the hex listing for the MON 2.

With this you can make your own MON 2, and save the cost of conversion.

Insert the data **0800** on the TEC, and continue through to **0D64.**

Go through the program at least once, checking each of the values to make sure a mistake has not been made. A single mistake can mean the difference between perfection and failure.

MON 2 HEX LISTING FOR TEC 1B:

```
0000  C3 00 02 FF      0114  1A 96 1C 8E      0228  FF FF FF FF      033C  01 06 20 10      0450  C3 7D 03 FF
0004  FF FF FF FF      0118  1E 86 20 7F      022C  FF FF FF FF      0340  FE AF D3 01      0454  FF 57 21 DF
0008  2A C0 08 E9      011C  22 77 24 71      0230  FF FF FF FF      0344  C1 D1 E1 F1      0458  08 CB 9E CB
000C  FF FF FF FF      0120  26 6A 28 64      0234  FF FF FF FF      0348  C9 FF FF FF      045C  66 20 08 01
0010  2A C2 08 E9      0124  2A 5F 2D 59      0238  FF FF FF FF      034C  FF FF FF FF      0460  00 00 CD 90
0014  FF FF FF FF      0128  2F 54 32 50      023C  FF FF FF FF      0350  21 80 00 1A      0464  04 CB E6 CD
0018  2A C4 08 E9      012C  35 4B 38 47      0240  31 C0 08 AF      0354  85 6F 7E 13      0468  89 02 78 07
001C  FF FF FF FF      0130  3C 43 3F 3F      0244  D3 01 D3 02      0358  21 DF 08 C9      046C  07 07 07 E6
0020  1A C6 08 E9      0134  43 3C 47 38      0248  21 B0 00 11      035C  FF FF FF FF      0470  F0 5F 79 07
0024  FF FF FF FF      0138  4B 35 50 32      024C  D8 08 01 05      0360  F5 E5 21 E0      0474  07 07 07 E6
0028  2A C8 08 E9      013C  54 2F 59 2D      0250  00 ED B0 CD      0364  08 3E FF BE      0478  0F 83 47 79
002C  FF FF FF FF      0140  5F 2A 64 28      0254  70 02 3E 08      0368  28 0E 7E E6      047C  07 07 07 07
0030  2A CA 08 E9      0144  6A 26 71 24      0258  CD 70 01 3E      036C  1F CB 6E 20      0480  E6 F0 82 4F
0034  FF FF FF FF      0148  77 22 7F 20      025C  0F CD 70 01      0370  02 C6 14 C3      0484  CD 90 04 CD
0038  2A CC 08 E9      014C  86 1E 8E 1C      0260  3E 01 32 DF      0374  A8 03 FF FF      0488  70 02 C3 7D
003C  FF FF FF FF      0150  96 1A 94 19      0264  08 CD A0 02      0378  E1 F1 C9 FF      048C  03 FF FF FF
0040  FF FF FF FF      0154  A9 18 B3 16      0268  CD 60 03 18      037C  FF E1 F1 C9      0490  F5 E5 21 D8
0044  FF FF FF FF      0158  BE 15 C9 14      026C  F8 FF FF FF      0380  FF FF FF FF      0494  08 78 E6 F0
0048  FF FF FF FF      015C  D5 13 E1 12      0270  F5 E5 C5 CD      0384  CD 89 02 C5      0498  07 07 07 07
004C  FF FF FF FF      0160  EF 11 FD 10      0274  89 02 E6 F0      0388  DD E1 DD 23      049C  77 23 78 E6
0050  FF FF FF FF      0164  FF FF FF FF      0278  0F 0F 0F 0F      038C  DD E5 E1 7C      04A0  0F 77 23 79
0054  FF FF FF FF      0168  FF FF FF FF      027C  32 DC 08 0A      0390  FE 40 28 08      04A4  E6 F0 07 07
0058  FF FF FF FF      016C  FF FF FF FF      0280  E6 0F 32 DD      0394  DD 7E 00 DD      04A8  07 07 77 23
005C  FF FF FF FF      0170  C5 D5 E5 F5      0284  08 C1 E1 F1      0398  77 FF 18 EE      04AC  79 E6 0F 77
0060  FF FF FF FF      0174  A7 20 03 5F      0288  C9 21 D8 08      039C  3E 00 32 FF      04B0  E1 F1 C9 FF
0064  FF FF F5 DB      0178  18 02 1E 80      028C  7E 07 07 07      03A0  3F CD 70 02      04B4  FF FF FF FF
0068  00 32 E0 08      017C  21 00 01 87      0290  07 23 86 47      03A4  C3 78 03 FF      04B8  FF FF FF FF
006C  F1 ED 45 FF      0180  85 6F 4E 23      0294  23 7E 07 07      03A8  C6 01 CD 70      04BC  FF FF FF FF
0070  FF FF FF FF      0184  46 7B D3 01      0298  07 07 23 86      03AC  01 C3 21 04      04C0  21 DF 08 CB
0074  FF FF FF FF      0188  10 FE 46 AF      029C  4F 0A C9 FF      03B0  CD 89 02 0B      04C4  9E CB A6 FE
0078  FF FF FF FF      018C  D3 01 10 FE      02A0  F5 E5 D5 C5      03B4  DD 21 FE 3F      04C8  10 CA E0 00
007C  FF FF FF FF      0190  0D 20 F1 F1      02A4  11 D8 08 AF      03B8  DD 7E 00 DD      04CC  FE 11 CA E6
0080  EB 28 CD AD      0194  E1 D1 C1 C9      02A8  D3 01 CD 50      03BC  77 01 DD 2B      04D0  00 FE 12 CA
0084  2E A7 E7 29      0198  FF FF FF FF      02AC  03 CB 4E 28      03C0  DD E5 E1 79      04D4  0C 03 FE 13
0088  EF 2F 6F E6      019C  FF FF FF FF      02B0  02 CB E7 D3      03C4  BD 20 F1 78      04D8  CA C0 01 FE
008C  C3 EC C7 47      01A0  F5 E5 2A D6      02B4  02 3E 20 D3      03C8  BC 20 ED DD      04DC  14 CA 50 05
0090  E3 66 28 E8      01A4  08 7E FE FF      02B8  01 06 20 10      03CC  36 01 00 CD      04E0  FE 15 CA FF
0094  4E C2 2D 6B      01A8  20 03 E1 F1      02BC  FE AF D3 01      03D0  70 02 C3 78      04E4  FE 16 FE 17
0098  EB 4F 2F 4B      01AC  C9 FE FE 28      02C0  CD 50 03 CB      03D4  03 FF FF FF      04E8  FF FF FE 17
009C  A7 46 EA E0      01B0  F1 23 CD 70      02C4  4E 28 02 CB      03D8  E5 F5 DD E5      04EC  CA F2 01 FE
00A0  AC A4 AE C9      01B4  01 18 EE FF      02C8  E7 D3 02 3E      03DC  C5 AF 32 DF      04F0  18 CA 70 05
00A4  10 08 18 04      01B8  FF FF FF FF      02CC  10 D3 01 06      03E0  08 06 06 21      04F4  FE 19 CA FF
00A8  2C 00 FF FF      01BC  FF FF FF FF      02D0  20 10 FE AF      03E4  D8 08 3E 29      04F8  FF FE 1A CA
00AC  FF FF FF FF      01C0  21 DF 08 CB      02D4  D3 01 CD 50      03E8  77 23 10 FC      04FC  FF FF FE 1B
00B0  00 09 00 00      01C4  46 20 07 CB      02D8  03 CB 4E 28      03EC  2A D0 08 7E      0500  CA FF FF FE
00B4  FF FF FF FF      01C8  C6 CB 8E C3      02DC  02 CB E7 D3      03F0  FE FF 20 06      0504  1C CA 60 06
00B8  FF FF FF FF      01CC  78 03 CB 86      02E0  02 3E 08 D3      03F4  C1 DD E1 F1      0508  FE 1D CA FF
00BC  FF FF FF FF      01D0  CB CE C3 78      02E4  01 06 20 10      03F8  E1 C9 FE FE      050C  FF FE 1E CA
00C0  1B 18 1E 1D      01D4  03 FF FF FF      02E8  FE AF D3 01      03FC  28 EE DD 21      0510  FF FF FE 1F
00C4  12 17 0E 29      01D8  C5 06 80 CD      02EC  CD 50 03 CB      0400  D8 08 06 05      0514  CA FF FF FE
00C8  0B 12 29 17      01DC  A0 02 10 FB      02F0  4E 28 02 CB      0404  DD 7E 01 DD      0518  20 CA FF FF
00CC  12 0C 24 29      01E0  C1 C9 FF FF      02F4  E7 D3 02 3E      0408  77 00 DD 23      051C  FE 21 CA FF
00D0  29 29 29 29      01E4  ED 4B D2 08      02F8  10 D3 01 06      040C  10 F6 7E 32      0520  FF FE 22 FE
00D4  FE 1C 1D 18      01E8  CD 90 04 CD      02FC  20 10 FE AF      0410  DD 08 23 06      0524  FF FF FE 23
00D8  17 0E FF FF      01EC  70 02 C3 78      0300  D3 01 00 C3      0414  40 CD A0 02      0528  CA FF FF FE
00DC  FF FF FF FF      01F0  03 FF ED 4B      0304  18 03 FF FF      0418  10 FB 18 D3      052C  24 CA B0 03
00E0  CD 89 02 03      01F4  D4 08 CD 90      0308  FF FF FF FF      041C  FF FF FF FF      0530  FE 25 CA 84
00E4  18 04 CD 89      01F8  04 CD 70 02      030C  CD 89 02 C5      0420  FF D6 01 36      0534  03 FE 26 CA
00E8  02 0B CD 90      01FC  C3 78 03 FF      0310  E1 31 C0 08      0424  FF CB 67 C2      0538  FF FE FE 27
00EC  04 CD 70 02      0200  ED 73 E8 08      0314  E9 FF FF FF      0428  C0 04 CB 6F      053C  CA E4 01 C3
00F0  21 DF 08 CB      0204  31 00 09 F5      0318  CD 50 03 CB      042C  C2 C0 04 21      0540  78 03 FF FF
00F4  C6 CB 8E C3      0208  C5 D5 E5 DD      031C  46 28 02 CB      0430  DF 08 CB 46      0544  FF FF FF FF
00F8  78 03 FF FF      020C  E5 FD E5 08      0320  E7 D3 02 3E      0434  CA 55 04 57      0548  FF FF FF FF
00FC  FF FF FF FF      0210  D9 F5 C5 D5      0324  02 D3 01 06      0438  CD 89 02 21      054C  FF FF FF FF
0100  FD 10 10 FD      0214  E5 ED 57 F5      0328  20 10 FE AF      043C  DF 08 CB 5E      0550  CD 89 02 60
0104  11 EF 12 E1      0218  AF 32 CC 08      032C  D3 01 CD 50      0440  20 03 AF CB      0554  69 3A E1 08
0108  13 D5 14 C9      021C  32 CD 08 3E      0330  03 CB 46 28      0444  DE 07 07 07      0558  23 BE 20 FC
010C  15 BE 16 B3      0220  FF 32 E0 08      0334  02 CB E7 D3      0448  07 E6 F0 82      055C  44 4D CD 90
0110  18 A9 19 9F      0224  C3 40 02 FF      0338  02 3E 01 D3      044C  02 CD 70 02      0560  04 C3 53 02
                                                                                            0564  FF FF FF FF
```

HOW THE CIRCUIT WORKS
(and a general discussion.)

The circuit diagram is TALKING ELECTRONICS COMPUTER 1B (TEC 1B). It is a 9-chip, single-board computer capable of executing Machine Code commands and displaying the result on either the inbuilt display (a set of 7-segment displays) or on other displays via the expansion socket.

The expansion socket is configured identical to the RAM socket and is accessed via line Y2 of the ROM/RAM decoder 74LS138, at the top right-hand corner of the diagram.

The computer starts-up via a MONitor program contained in the 2732 and two monitor programs are in this chip.

The MON 1 select switch takes address line A11 LOW for the low half and HIGH for the upper half.

The other major change between TEC 1 and TEC 1B is the output latches. They were originally 8212's but now 74LS273's have been used. These are a modern chip and are more readily available.

STARTING UP

When the power is applied to the computer, the reset line on the Z-80 is taken low for an instant via the 100n capacitor and this resets the internal workings of the Z-80.
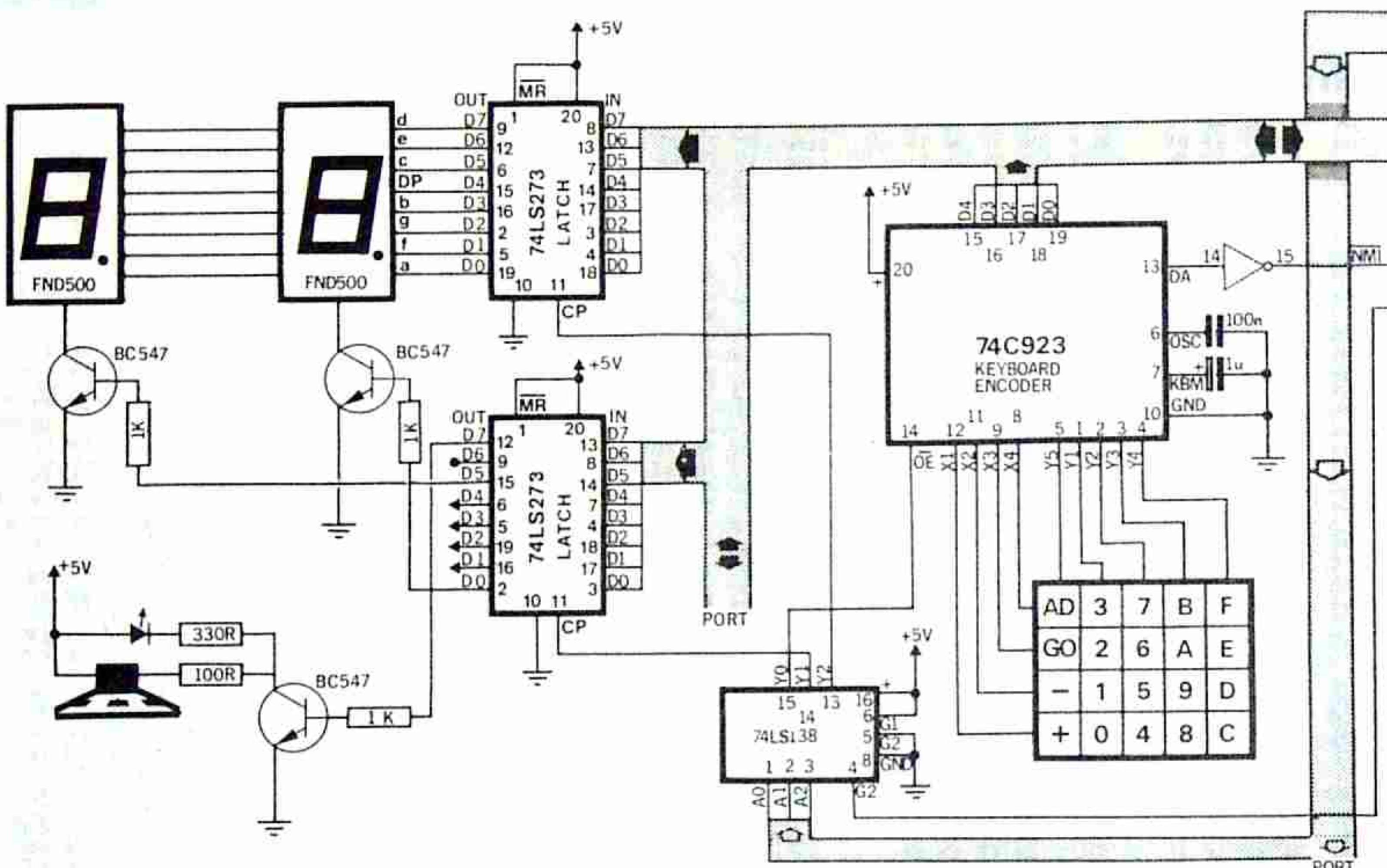
Its first operation is to look for the first byte of data at address zero, in the monitor. Depending on this being a one-

contains 11 lines while the data bus contains 8 lines. The data bus is always 8 bits wide for a Z-80 processor and this gives it the name '8-bit system'.

The address bus is a **ONE-WAY** bus in which the Z-80 activates the lines and turns them on and off using binary notation to generate an address value.

When all lines are LOW, address zero is represented. When line A0 is HIGH, address 1 is represented. The Z-80 has 16 address lines and address 1 is: 0000 0000 0000 0001. When line A1 is HIGH, address 2 is: 0000 0000 0000 0010

The address lines connect to a number of chips but only one will respond due to a 'turn-on' line called a command line being required to be activated.

TEC 1B COMPUTER CIRCUIT

When the ROM select switch is HIGH, MON-1 program is accessed and the computer displays **0800**. When the switch is LOW, the computer displays **0900** and the MON 2 program operates.

This has been done so that the TEC 1B is compatible with the original TEC 1 and it can be upgraded by adding a monitor switch and a programmed 2732 EPROM. The original TEC 1 had a 2716 EPROM but these chips are no longer manufactured and thus a 2732 is now used. When a 2732 is placed in a 2716 socket the upper half of the chip is accessed and thus MON 1 program has been placed in the upper half.

byte, two-byte or three-byte instruction, the Z-80 will execute it or request one or two more bytes.

The flow of information from the Z-80 to the other chips is via two buses. They are the **ADDRESS BUS** and **DATA BUS**. In addition, there is a set of control lines (sometimes referred to as the control bus) that activate (generally) one chip at a time.

All signals within the computer are at a level equal to rail voltage (called HIGH) or ground (called LOW). For this reason they are called digital circuits.
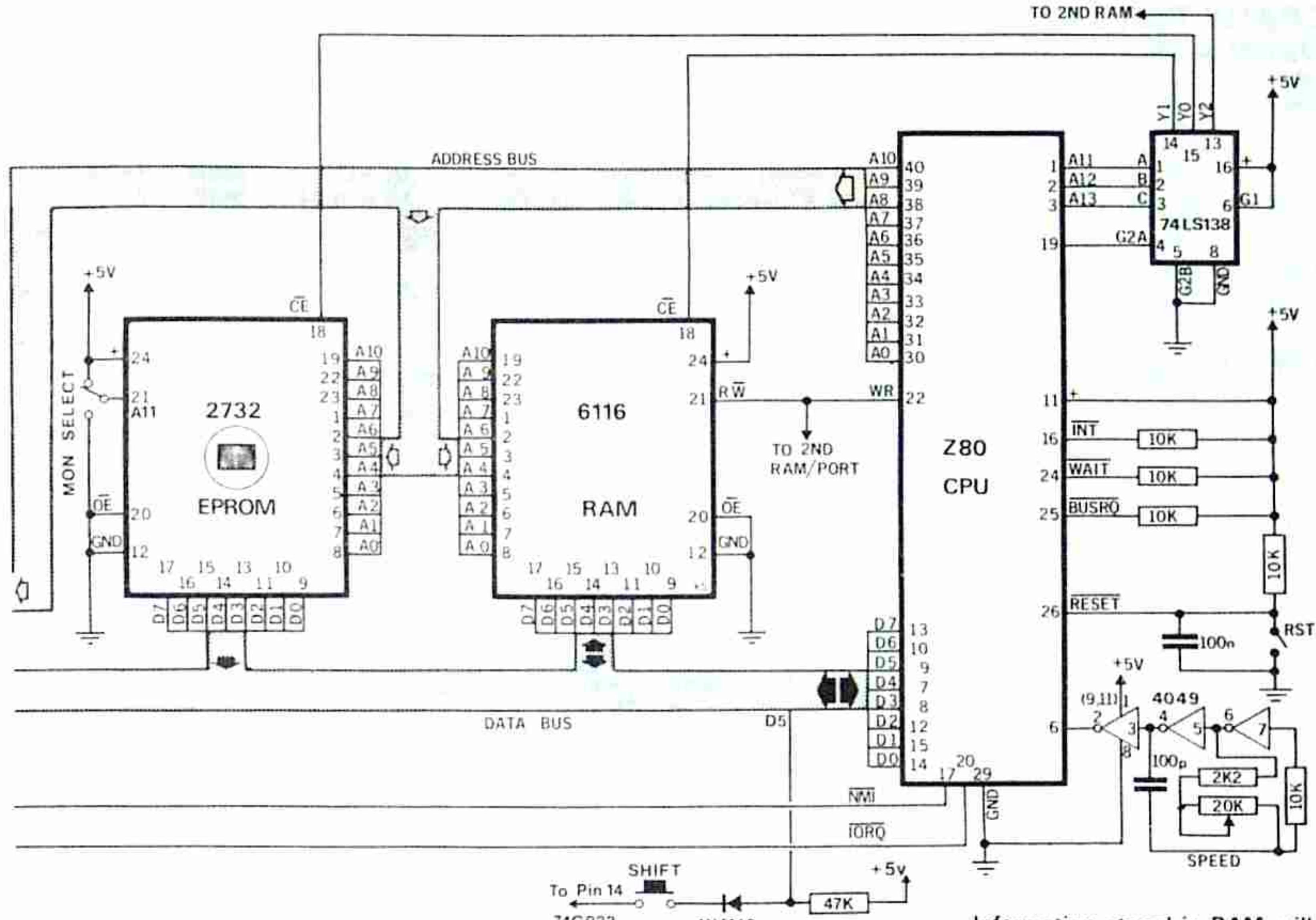
The shaded paths of the diagram represent buses and the address bus

These command lines are called chip select, chip enable or output enable and this allows only one chip to be activated at a time.

The chip select lines are the outputs of a decoder chip and this chip is 'turned on' by the Z-80 and only one of its outputs goes low at a time.

It is a 3-line to 8-line decoder and this means it has 3 input lines and depending on the HIGH-LOW values on these lines, one of the outputs will go low.

This is a form of expander so that a single line from the Z-80 (e.g. from pin 19 or 20) can control 8 devices.

TO 2ND RAM

ADDRESS BUS

+5V

74LS138

Y1 Y0 Y2
14 13
15 16 +
A11 A
A12 B
A13 C G1
G2A
G2B GND

+5V

A10 40
A9 39
A8 38
A7 37
A6 36
A5 35
A4 34
A3 33
A2 32
A1 31
A0 30

+5V

MON SELECT

CE 18

2732

EPROM

24
21
A11
OE 20
GND 12

19 A10
22 A9
23 A8
1 A7
2 A6
3 A5
4 A4
5 A3
6 A2
7 A1
8 A0

17 15 13 10
16 14 11 9
D7 D6 D5 D4 D3 D2 D1 D0

CE 18

6116

RAM

24
21 RW
OE 20
GND 12

19 A10
22 A9
23 A8
1 A7
2 A6
3 A5
4 A4
5 A3
6 A2
7 A1
8 A0

17 15 13 10
16 14 11 9
D7 D6 D5 D4 D3 D2 D1 D0

TO 2ND
RAM/PORT

WR 22

Z80
CPU

11 +
16 INT   10K
24 WAIT   10K
25 BUSRQ 10K

10K

26 RESET

+5V

RST

100n

DATA BUS

D5

D7 13
D6 10
D5 9
D4 7
D3 8
D2 12
D1 15
D0 14

17 20 29

NMI

IORQ

GND

4049
(9,11)1 2 3 4 5 6 7
100p
2K2
20K
10K

SPEED

SHIFT
To Pin 14
74C923   1N4148   47K   +5V

The top right-hand decoder is called the ROM/RAM decoder and the lower left-hand, the IN/OUT decoder.

The data from the monitor flows to the Central Processing Unit (the Z-80) along the data bus as 8 parallel bits of information AT THE SAME TIME.

This is called a BYTE of information and can have 256 different possibilities. The Z-80 knows if the byte is data or instruction by the fact that it starts at address zero looking for an instruction byte. From there the program must follow correctly and this is the responsibility of the programmer.

The data enters the Z-80 via a holding register (an instruction register) that is not available to the programmer and to keep the discussion simple, we consider the byte flows directly into the A register (called the accumulator). This is the only register capable of accepting information from the data bus. All other registers must be fed from the accumulator.

Data can also flow out of the Z-80 along the data bus and this bus is BI-DIRECTIONAL. The arrows on the bus show the direction of flow of information.

The keyboard is scanned by the 74C923 and this is called hardware scanning as the chip has inbuilt scanning circuits for a matrix of 20 keys.

When a key is pressed, a signal is generated at the Data Available pin and the Z-80 is notified via the Non-Maskable Interrupt line.

The Z-80 immediately ceases all processing and jumps to address 66 in the MONitor. Here it executes a short program and activates the input/output decoder to turn on the keyboard encoder. The encoder puts a 5-bit number on the data bus and this is stored for later use or operated upon, as required.

When the shift button is pressed, and kept pressed while one of the keys is pressed, an extra bit is added to create a 6-bit number and thus an additional set of 20 commands can be created.

The output latches are also controlled by the in/out decoder and the control line on each latch is called CP (clock pulse).

When these lines are taken LOW, then HIGH again, the data appearing on the input lines is latched into the chip and will appear on the output lines and will remain there.

This allows devices such as 7-segment displays, relays or globes etc. to be activated.

The 6116 RAM is RANDOM ACCESS MEMORY and as the name suggests, bytes of information can be placed anywhere in its matrix of cells. These bytes are generally data however programs can be stored and run in RAM and these are usually developmental programs.

Information stored in RAM will only be maintained as long as the power is applied as the flip flops storing the data will not hold their state when power is removed.

'ADD-ONs'

This computer is only a baby in the computer world however it does have the facility for expansion and already a number of 'add-ons' have been produced.

Possibly the most important add-on is the NON-Volatile RAM. This consists of a battery backed-up 6116, into which programs can be placed.

Other devices can be connected to the system via the expansion port and this includes an IN/OUT module, an OUTPUT module, a display module and a controller module (to come).

The clock oscillator is adjustable via a speed control pot and allows programs to be run at different speeds for assessment. If a real-time situation is required, a crystal oscillator can be fitted and this will allow time to be programmed accurately.

The main intention of this computer is to provide the starting point for an understanding into computer operations. For this reason, machine code programming has been employed. This means you will be able to create your own systems for such applications as controllers and timers for industry and home and be able to produce the project from the ground up, without requiring any external operating system.

PROGRAMS FOR THE TEC DISPLAYS and a sound Program:

Here are three programs for the TEC and TEC displays. The effects that can be produced on a set of 7-segment displays is quite amazing. I thought we had run out of ideas and yet they still keep coming.

The first program is a Space Invaders sound effect using button 4 as the firing button. The other two programs use the displays.

SPACE INVADERS 'SHOOTING'

Phillip Barns.   2118

Computer sounds and effects are always impressive, especially when we have control over them.

This program does just that.

It is a Space Invaders sound effect and you can control it via button 4.

The point to note with this program is the way the delay is increased by inserting a varying value into a delay loop. In the latter half of the program the OFF time is gradually increased by placing another varying value into a delay loop.

The resulting ON-OFF values outputted to the speaker produce the changing tone.

The program only accepts the press of button '4' (determined by **CP 04**) and by pressing this button repeatedly, a firing sound will be produced.

LD A,12	800	3E 12
LD I,A	802	ED 47
LD H,FF	804	26 FF
LD B,01	806	06 01
INC B	808	04
LD A,80	809	3E 80
OUT (01),A	80B	D3 01
CALL 0828	80D	CD 28 08
XOR A	810	AF
OUT (01),A	811	D3 02
CALL 0828	813	CD 28 08
LD A,I	816	ED 57
CP 04	818	FE 04
JP Z 0800	81A	CA 00 08
DEC H	81D	25
JP NZ 0808	81E	C2 08 08
CP 04	821	FE 04
JR NZ 0821	823	20 FC
JP 0800	825	C3 00 08
LD C,B	828	48
DEC C	829	0D
JR NZ 0829	82A	20 FD
RETURN	82C	C9

THE BOX   G.L Dunt   3219.

This program is an extension of the techniques we have been discussing in issue 12, P 18, covering the control of two or more pixels at the same time.

It produces an interesting piece of animation in which a box with lid is displayed and moved across the screen in a 'chase scene'.

Again we won't say much about the effect, except to say that you can get quite involved with it and find it very easy to improve upon.

The program consists of 25 'frames' and each frame requires 4 bytes of the table to produce the necessary effects. Each time you increase the table (by 4 bytes) you must also increase the counter register by one (for each frame).

By using 4 bytes we gain the ability to control two pixels at the same time. If only one display is required, the two pairs of bytes will be identical.

LD IX 0840	0800	DD 21 40 08
LD D,10	0804	16 19
LD C,40	0806	0E 40
LD A(IX + 00)	0808	DD 7E 00
OUT (01),A	080B	D3 01
LD A(IX + 01)	080D	DD 7E 01
OUT (02),A	0810	D3 02
DJNZ	0812	10 FE
XOR A	0814	AF
OUT (02),A	0815	D3 02
LD A(IX + 02)	0817	DD 7E 02
OUT (01),A	081A	D3 01
LD A(IX + 03)	081C	DD 7E 03
OUT (02),A	081F	D3 02
DJNZ	0821	10 FE
DEC C	0823	0D
JR NZ 0808	0824	20 E2
INC IX	0826	DD 23
INC IX	0828	DD 23
INC IX	082A	DD 23
INC IX	082C	DD 23
DEC D	082E	15
JR NZ 0806	082F	20 D5
JP 0800	0831	C3 00 08

at 0840:

01	01	01	20
E4	E4	80	E4
01	01	10	20
E4	E4	C4	E4
01	01	01	10
E8	E1	80	E4
01	01	20	10
E8	E1	E0	E4
01	01	02	08
E4	01	80	E4
01	02	20	08
E4	E0	E0	E4
01	01	04	04
E2	04	80	E0
01	02	20	08
E2	E0	E0	04
01	01	08	02
E4	80	80	E0
01	02	20	08
E4	E0	E0	04
01	01	10	01
E4	80	04	E0
01	04	20	04
E4	A4	E0	04
01	01	20	01
E2	80	E1	E0
01	08	20	02
E2	64	E1	04

Halilovic's Piano:

This program has been designed by BOB Halilovic and gives a piano effect when one of the 20 keys is pressed. The notes have a pre-determined length, and this distinguishes it from the organ programs we have previously presented.

Data	0800	00
Data	0801	09
LD A,1F	0802	3E 1F
LD (0901),A	0804	32 01 09
CALL 01B0	0807	CD B0 01
HALT	080A	76
CP 10	080B	FE 10
JR NC	080D	30 07
ADD A,05	080F	C6 05
LD (0900),A	0811	32 00 09
JR 0807	08014	18 F1
SUB A,0F	0816	D6 0F
JR 0811	0818	18 F7

G. Sheehan &
D. Svendsen. 3175

BOOMERANG

Boomerang is a program for the TEC displays. The effect you get is so clever that we are not going to spoil it by telling you what happens.

The only point we will mention is the composition of the byte table.

Each pass of the program uses two bytes from the table and the end of the program is detected by looking for address **0844**. Register L will be 44 at the end of the table.

By using the table two bytes at a time, we can specify the display we wish to access and the segment to be lit.

Also, using a byte table like this requires less program and fewer registers. It is one of the tricks of compact programming.

The delay at 0900 produces the speed of execution.

Try altering and modifying the program and you will learn a lot about what each instruction does. You can also lengthen it by adding more frames. It'll be like creating your own cartoon.

LD HL,0820	0800	21 20 08
LD A(HL)	0803	7E
OUT (01),A	0804	D3 01
INC HL	0806	23
LD A,(HL)	0807	7E
OUT (02),A	0808	D3 02
INC HL	080A	23
CALL 0900	080B	CD 00 09
LD A,L	080E	7D
CP 44	080F	FE 44
JP NZ 0803	0811	C2 03 08
JP 0800	0814	C3 00 08

at 0820:

01	20	20
09	C0	6F
02	10	10
03	A0	EA
04	08	08
06	24	A7
08	04	04
0C	44	A7
10	02	02
09	C0	28
20	01	01
03	A0	C7

Delay at 0900:

	900	11 FF 0A
	903	1B
	904	7B
	905	B2
	906	C2 03 09
	909	C9

PROGRAMS FOR THE 8x8 DISPLAY:

The 8x8 has remained a popular 'add-on' and we still get requests for more programs for it. Here are some recent submissions:

If you have written a program equal to these, send it in for inclusion in the next issue:

FAN OUT Mk III

Dean Svendsen 3175.

FAN OUT Mk III produces symmetry on the displays and can be seen by the same byte being outputted to both ports 3 and 4. The end of the table is detected by looking at the value of L and starting again when it equals the address of the end of the table.

LD HL 0815	21 15 08
LD A(HL)	7E
OUT (03),A	D3 03
OUT (04),A	D3 04
INC HL	23
CALL 0900	CD 00 09
LD A,L	7D
CP 20	FE 20
JP NZ 0803	C2 03 08
JP 0800	C3 00 08

at 0815:

18	81
3C	C3
7E	E7
FF	FF
E7	7E
C3	3C

900	11 FF 0A
903	1B
904	7B
905	B2
906	C2 03 09
909	C9

BOUNCING BALL AND ROLLING BALL.

G.L. Dunt, 3219.

This program is an extension and improvement over the Bouncing Ball program in issue 12, P. 26.

If you look at P.26, you will notice the program is fairly long.

This is because it is necessary to specify the start address of the ball, each time it changes direction.

Much of the program is a repetition of similar or nearly similar codes and to reduce its length we need to look at any part(s) that repeat.

At first they may not be obvious but one can be found that starts at the base of a column, up the column, across to the next and down to the base again. The sequence ends with the LED jumping to the start of the next column.

If we repeat this 4 times, the whole of the board will be covered. This will reproduce

the effect as described on P. 26 of issue 12. Using the same technique, we can travel across the display and back again, to produce a weaving effect as the LED advances up the display. To complete the travel we need to move the LED from the top right hand corner to the lower left hand corner, ready for the start of the next sequence.

By using efficient programming as covered in this program, we can produce twice the effect with about half the program.

Most of the reduction is done by defining the co-ordinates of the ball only once. This is done at the beginning of the program and from there the ball position is kept in the C and D registers. They act as the x and y values in co-ordinate geometry.

To move the LED across or up and down the screen, the C and D registers are rotated left or right. Each register contains only one bit and when this moves out the end of the register, it either "sits in the carry box" or passes it and enters the other end of the register. In either case the carry flag is affected and we look for this to let us know the end of the display has been reached.

As you can see, the LED is either "off the end of the board" or at the other side of the display, when the carry is detected and we must shift it back one location, ready for the next run. This way the LED appears to be darting back and forth from one side to the other, and we are not aware of the 'corrections' that take place.

LD C,01	0800	0E 01
LD D,01	0802	16 01
LD A,C	0804	79
OUT (03),A	0805	D3 03
LD A,D	0807	7A
OUT (04),A	0808	D3 04
CALL 0900	080A	CD 00 09
RLC D	080D	CB 02
JR NC 0807	080F	30 F6
RR D	0811	CB 1A
RLC C	0813	CB 01
LD A,C	0815	79
OUT (03),A	0816	D3 03
LD A,D	0818	7A
OUT (04),A	0819	D3 04
CALL 0900	081B	CD 00 09
RR D	081E	CB 1A
JR NC,0818	0820	30 F6
RL D	0822	CB 12
RLC C	0824	CB 01
JR NC,0804	0826	30 DC
RRC C	0828	CB 09
LD A,D	082A	7A
OUT (04),A	082B	D3 04
LD A,C	082D	79
OUT (03),A	082E	D3 03
CALL 0900	0830	CD 00 09
RRC C	0833	CB 09
JR NC,082D	0835	30 F6
RL C	0837	CB 11
RLC D	0839	CB 02
LD A,D	083B	7A
OUT (04),A	083C	D3 04
LD A,C	083E	79
OUT (03),A	083F	D3 03
CALL 0900	0841	CD 00 09
RLC C	0844	CB 01
JR NC,083E	0846	30 F6

RRC C	0848	CB 09
RLC D	084A	CB 02
JR NC,082A	084C	30 DC
RRC D	084E	CB 0A
RRC D	0850	CB 0A
LD A,D	0852	7A
OUT (04),A	0853	D3 04
CALL 0900	0855	CD 00 09
RR D	0858	CB 1A
JR NC,0852	085A	30 F6
RRC C	085C	CB 09
LD A,C	085E	79
OUT (03),A	085F	D3 03
CALL 0900	0861	CD 00 09
RRC C	0864	CB 09
JR NC,085E	0866	30 F6
JP 0800	0868	C3 00 08

At 0900:

LD HL,06FF		21 FF 06
DEC HL		2B
LD A,L		7D
OR H		B4
JP NZ 0903		C2 03 09
Return		C9

RAIN DROPS:

Jim Robertson,

This program produces a very effective pattern, similar to falling rain. The random number generator is the interesting part as it is very difficult to produce random numbers in a program that loops.

CALL Random Nos.		CD 00 0A
AND 07		E6 07
LD H,0B	0805	26 0B
LD L,A	0807	6F
RLC (HL)	0808	CB 0E
LD DE,0006	080A	11 06 00
CALL SCAN	080D	CD 00 09
DEC DE	0810	1B
LD A,D	0811	7A
OR E	0812	B3
JRNZ	0813	20 F8
JR START	0815	18 E9

at 0900:

SCAN

LD HL 0B00	0900	21 00 0B
LD B,01	0903	06 01
LD A(HL)	0905	7E
OUT (03),A	0906	D3 03
LD A,B	0908	78
OUT (04),A	0909	D3 04
LD B,20	090B	06 20
DJNZ	090D	10 FE
INC HL	090F	23
LD B,A	0910	47
XOR A	0911	AF
OUT (04),A	0912	D3 04
RLC B	0914	CB 00
JR NC	0916	30 ED
RETURN	0918	C9

at 0A00:

RANDOM NUMBERS:

LD A,R	0A00	ED 5F
LD B,A	0A02	47
LD A,R	0A3	ED 5F
RLA	0A05	17
LD R,A	0A06	ED 4F
DJNZ	0A08	10 FB
RETURN	0A0A	C9

PHONE DIALLER

TURNING THE TEC INTO A PHONE DIALLER

The following three or four pages examine the development of an idea. It is a Telephone Dialler capable of storing up to 30 or 40 names and phone numbers with a dialling facility and auto re-dial.

It is only a program of ideas as the output appears on a speaker in the form of tones.

Since this is a fiarly ambitious concept, it has been divided into 3 sections. Each section describes a program that is complete in itself and increases in complexity with complete design in section 3.

The first program is fairly simple. It shows how to get figures from the keyboard and display them on the screen. The second contains two function buttons, C and E. The 'C' key clears the screen and 'E' indicates the end of a phone number.
The third program is much more complex. It has more features and is keeping track of more things.

Each program has been created from scratch as it is almost impossible to 'add onto' an existing program.

Type each of these programs into the TEC and study them. This way you will learn how they operate.

PHONE DIALLER PROGRAM 1.

This program is limited to displaying 6 digits on the TEC screen as no scrolling feature is present.   As the keys are pressed, the numbers fill the screen from left to right. When the screen is full, the capability of the program is reached.

The screen buffer is located at 0900 and the scan rate is determined by the value of B (at 082E and 082F). We can increase or reduce the scan rate by altering the value of B and by adjusting the TEC clock speed.

No other features are available in this program. The TEC must be reset and 'GO' pushed to clear the screen so that a new number can be keyed in.

This simple program shows how to get numbers from the keyboard and onto the screen.

The only instruction that will be unfamiliar is **JRNC**. It effectively divides the keyboard in two, allowing keys 0-9 to be accepted and A-F to be disregarded.

JRNC means Jump Relative if the Carry flag is NOT SET.   When the previous instruction is a 'COMPARE', it is best to substitute the word 'BORROW' for carry, and the instruction will be much easier to understand. This is because the compare instruction subtracts the data byte from the accumulator and if a borrow is required, the carry flag is SET.

PHONE DIALLER · Part 1

LD D, 08	0800	16 08	The first 8 memory locations are cleared so that the program will come on with a blank screen. We need only 6 locations. The 7th location is explained in the text. Register A is zeroed and this value is inserted into **0900 - 0907** via the HL register being the pointer register.
XOR A	0802	AF	
LD HL,0900	0803	21 00 09	
LD (HL),A	0806	77	
INC HL	0807	23	
DEC D	0808	15	
JR NZ	0809	20 FB	
LD A,I	080B	ED 57	The Index register contains the value of the key.
CP 0A	080D	FE 0A	Compare the accumulator with **0A**.
JR NC	080F	30 12	Jump relative if the key is A or higher.
LD DE 0880	0811	11 80 08	Load DE with the start of the DISPLAY TABLE.
ADD A,E	0814	83	Add **80** to the key value.
LD E,A	0815	5F	Load the result back into E. DE will point to a table-byte.
LD HL,0900	0816	21 00 09	Load HL with the start of memory.
LD A,(HL)	0819	7E	Look for the first blank memory location by loading the value pointed to by HL into the accumulator and comparing with zero until a blank location is found.
CP 00	081A	FE 00	
JR Z	081C	28 03	
INC HL	081E	23	
JR	081F	18 F8	
LD A(DE)	0821	1A	When found, load A with the byte pointed to by DE.
LD (HL),A	0822	77	Load the table value into the blank memory location.
LD A,FF	0823	3E FF	Change the value of the index register by loading it with FF so that we can detect the same or another button.
LD I,A	0825	ED 47	
LD C,20	0827	0E 20	start the scan at the left hand end of the display.
LD HL,0900	0829	21 00 09	Load HL with start of memory.
LD D,06	082C	16 06	Load D with 06 for 6 loops of the program.
LD B,00	082E	06 00	Load B with delay value for turning ON each digit.
LD A(HL)	0830	7E	Load the data at the first memory location into A.
OUT (02),A	0831	D3 02	Output to the segment port.
LD A,C	0833	79	Load C into A.
OUT (01),A	0834	D3 01	Output to the cathode port.
RRC C	0836	CB 09	Rotate register C right, to access the 2nd display.
DJNZ	0838	10 FE	Create a short delay to display the digit.
XOR A	083A	AF	Zero A
OUT (01),A	083B	D3 01	Output to the cathode port to turn display OFF.
INC HL	083D	23	Increment to the next location.
DEC D	083E	15	Decrement the loop register.
JR NZ	083F	20 ED	Jump to start of loop if D not zero.
JR	0841	C3 0B 08	Jump to start of program if D zero and look for new key.

In our program, **CP 0A** causes the Z-80 to substact **0A** from the accumulator (it will hold the value of the key). When any key below **A** is pressed, the subtraction operation creates a borrow and this sets the carry flag. If we push key **6**, the operation will be 6 - A and the answer will require a borrow. Thus the carry flag will be SET. If we go to the program, we can see the Z-80 will continue down the program and NOT JUMP as the instruction says: JUMP RELATIVE NO BORROW.

To fully understand these instructions you have to comprehend the double negative. For instance: I am NOT, NOT going to jump means I AM going to jump.

Type the program at **0800** and the display conversion table at **0880.**

Push RESET, GO and the displays will blank. Press any combination of keys and notice that only number keys respond.

Modify the value of B in the scan section to increase the scan rate.

Some ideas for experimenting include: scanning from the opposite direction, scanning only 5 displays, allowing letters to appear on the screen, and changing the output to a CODE, so that you can turn it into a CODE-BREAKING game.

at 0880:

EB
28
CD
AD
2E
A7
E7
29
EF
AF

PHONE DIALLER · Part 2

The second part of the Phone Dialler program uses a different approach. As we have said, each must start afresh as it is more difficult to adapt an existing program.

This program accepts a string of digits of any length and will remember them for recall after key E (for END) has been pressed.

The **C** button clears the display and can be pressed at any time. When the desired number has been entered, button E is pressed. The display is blanked and the numbers emerge from the right hand end of the display and shift across to the left. Three empty spaces are created before the numbers start again.

This program introduces the concept of control keys and also the need for subroutines for any sequence that is required more than once.

Programs increase in length as more and more housekeeping is called for. Housekeeping is looking for button presses or detecting the end of a sequence etc.

The prime requirement of the program is to keep the displays illuminated. This means we must be calling SCAN for most of the time and as you will see, the SCAN routine is a favourite place to put housekeeping.

If you want a key to be immediately responsive, it must be checked during the SCAN loop. To be more precise, it must be checked during the inner-most loop as this is the loop which is being run for most of the time.

Key the program into the TEC and run it. Try changing some of the locations and see the result. This is the best way to following what is happening, especially at specific locations.

HOW THE PROGRAM WORKS

The program generates 2 memory areas. One is made up of 6 locations, from **0900** to **0905** and is called the **DISPLAY BUFFER**. The other is from **0907** onwards and is called **MEMORY AREA**.

The SCAN ROUTINE (at 0877) looks at the Display Buffer locations and outputs their value onto the displays.

The remainder of memory, starting at 0907 holds any number of digital as required and is open-ended.

One location, **0906**, is left blank and its purpose will be explained later.

As each number is keyed in, it is stored in memory, from 0907 onwards, and the HL register pair keeps track of the next available location.

The number is also outputted onto the display but firstly a SHIFT ROUTINE is called. The function of this routine is to take the value corresponding to the left-hand digit and drop it out of the buffer zone. The second location is then transferred to the first, the third to the second etc until all the digits have been shifted one place to the left. This leaves an empty hole at the right-hand end of the display.

The way in which this empty space is generated is quite clever. The '00' in **0906** is shifted into the 6th buffer location.

The program then loads the present key value in the buffer zone, position six, and reverts to a scan situation in which it is looking for an 'end of number' via button E.

When this is detected, memory is incremented one location and E is inserted.

The displays are cleared and the program picks up the first digit at 0907and places it in the 6th position of the buffer area.

The shift routine is called then the next memory value is placed in the 6th buffer location.

Before each new value is loaded into the buffer area, it is compared with **0E** to detect the 'end of message.'

When E is detected, three blank locations are produced and the message starts again.

The CLEAR function is included in the SCAN routine. This has been done so that CLEAR can be detected instantly, as the display scan must be running at all times to keep the displays illuminated.

DIALLER Part 2 listing: Main Program:

```
LD D,20          0800    16 20
CALL CLEAR       0802    CD 5B 08
LD HL, 0907      0805    21 07 09
LD A,I           0808    ED 57
CP 0A            080A    FE 0A
JR NC,0820       080C    30 12
INC HL           080E    23
LD DE,08A5       080F    11 A5 08
ADD A,E          0812    83
LD E,A           0813    5F
CALL SHIFT       0814    CD 65 08
LD A,(DE)        0817    1A
LD (HL),A        0818    77
LD (0905),A      0819    32 05 09
LD A,FF          081C    3E FF
LD I,A           081E    ED 47
CP 0E            0820    FE 0E
LR Z,002A        0822    28 05
CALL SCAN        0824    CD 77 08
JR 0808          0827    18 DF
INC HL           0829    23
LD (HL),A        082A    77
LD D,06          082B    16 06
CALL CLEAR       082D    CD 5B 08
LD HL,0907       0830    21 07 09
LD A,(HL)        0833    7E
LD D,20          0834    16 20
INC HL           0836    23
CP 0E            0837    FE 0E
JR Z,0849        0839    28 0E
LD (0905),A      083B    32 05 09
CALL SCAN        083E    CD 77 08
DEC D            0841    15
JR NZ,083E       0842    20 FA
CALL SHIFT       0844    CD 65 08
JR 0833          0847    18 EA
LD E,02          0849    1E 02
LD D,20          084B    16 20
CALL SCAN        084D    CD 77 08
DEC D            0850    15
JR NZ,084D       0851    20 FA
CALL SHIFT       0853    CD 65 08
DEC E            0856    1D
JR NZ,084B       0857    20 F2
JR 0830          0859    18 D5
```

Clear:

```
XOR A            085B    AF
LD HL,0900       085C    21 00 09
LD (HL),A        085F    77
INC HL           0860    23
DEC D            0861    15
JR NZ, 085F      0862    20 FB
RETURN           0864    C9
```

Shift:

```
LD B,07          0865    06 07
LD IX,08FF       0867    DD 21 FF 08
LD A,(IX + 01)   086B    DD 7E 01
LD (IX + 00),A   086E    DD 77 00
INC IX           0871    DD 23
DEC B            0873    05
JR NZ,086B       0874    20 F5
RETURN           0876    C9
```

Scan:

```
PUSH HL          0877    E5
PUSH DE          0878    D5
LD C,20          0879    0E 20
LD HL,0900       087B    21 00 09
LD D,06          087E    16 06
LD B,80          0880    06 80
LD A,(HL)        0882    7E
OUT (02),A       0883    D3 02
LD A,C           0885    79
OUT (01),A       0886    D3 01
RRC C            0888    CB 09
DJNZ 088A        088A    10 FE
XOR A            088C    AF
OUT (01),A       088D    D3 01
INC HL           088F    23
LD A,I           0890    ED 57
CP 0C            0892    FE 0C
JR Z,089C        0894    28 06
DEC D            0896    15
JR NZ,0880       0897    20 E7
POP DE           0899    D1
POP HL           089A    E1
RETURN           089B    C9
POP DE           089C    D1
POP HL           089D    E1
LD A,FF          089E    3E FF
LD I,A           08A0    ED 47
JP 0800          08A2    C3 00 08
```

at 08A5:

```
0 = EB
1 = 28
2 = CD
3 = AD
4 = 2E
5 = A7
6 = E7
7 = 29
8 = EF
9 = AF
0 =
```

PHONE DIALLER · Part 3

The third and final part of the Phone Dialler program is the longest and most impressive. It looks complicated because it is looking after a lot of things.

The program accesses memory and when using the 2k onboard RAM, it is capable of holding up to 36 names and numbers, each fitting into a block of memory 20H bytes long. The program allows up to 27 characters for the name and number and this should be sufficient for any situation.

The program uses a lot of sub-routines and they perform most of the work.

As the processor goes through the MAIN program, it CALLS the sub-routines and they do all the displaying, shifting, display converting etc.

Any operation that is required more than once is put into the form of a sub-routine. This reduces the length of the program and allows the sub-routines to be called as many times as required.

USING THE PROGRAM

Basically the program is self explanatory as the instructions for its use are displayed on the screen after the GO button is pressed.

The first instruction is to select an INDEX NUMBER from 00 to 36 (decimal) into which the telephone number is placed.

Push button E and the screen will blank so that the index number can be inserted.

The index number will remain on the screen for about one second and then the second set of instructions will appear. After reading the instructions, push E. This will cause the screen to blank so that you can type the name corresponding to the phone number.

After the end of the name, insert a space by typing F and the program will convert to displaying a digit for each key pressed.

At the end of the phone number type E and the program will scroll the contents of memory.

To dial the phone number push D. The program will pause for 5 seconds then dial the number.

At the completion of dialling, the screen will scroll the name and number again.

You can redial the same number at any time by pressing D.

To re-load the memory BLOCK, push C. This will re-start the program and allow a new name and number to be inserted.

Once a name and number has been inserted into memory at a particular index value, it can be dialled very quickly. You can push either button C or RESET. If the Reset button is pushed, the GO button must be pushed for the first set of instructions to appear.

Push E and insert the index number; then push D. The computer will dial the number. A constant beeping will indicate the location is not filled and you should try another index.

At the end of dialling, the name and number will scroll and you can confirm it to be correct.
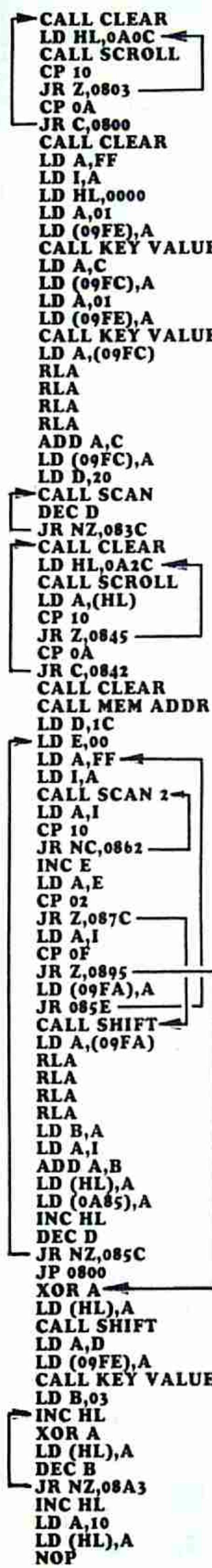
A SUMMARY OF THE PROGRAM

The program creates a display buffer area at **0A80** to **0A85** and the values placed at these 6 locations are directly transferred to the TEC display via the SCAN routine.

The CLEAR routine zeros each of these locations and also the next location. This is one of the clever tricks of the program, and it is cleared for the following reason:

The SHIFT routine starts at a location that is one lower than **0A80**, (namely **0A7F**) and places the data at **0A80** into

PHONE DIALLER PROGRAM:

Instruction	Address	Hex	Comment
CALL CLEAR	0800	CD 20 09	The first 7 lines of the program displays "Enter Index
LD HL,0A0C	0803	21 0C 0A	. . . ." etc and looks for the value 10 at the end of the
CALL SCROLL	0806	CD C0 09	table to repeat the sequence. The program also looks
CP 10	0809	FE 10	for an input value above 9 to jump out of the loop.
JR Z,0803	080B	28 F6	
CP 0A	080D	FE 0A	
JR C,0800	080F	38 EF	
CALL CLEAR	0811	CD 20 09	The screen is cleared and the index register is loaded
LD A,FF	0814	3E FF	with FF so that we can detect when a button has been
LD I,A	0816	ED 47	pushed.
LD HL,0000	0818	21 00 00	Memory is set to zero by loading HL with 00 00.
LD A,01	081B	3E 01	Location **09FE** stores the value 01 so that key value is
LD (09FE),A	081D	32 FE 09	called once. The requirement of the next 12 lines is to
CALL KEY VALUE		CD 30 09	get a double decimal number into location **09FC**.
LD A,C	0823	79	C will contain the key value and this is loaded into
LD (09FC),A	0824	32 FC 09	memory location **09FC** (first figure)
LD A,01	0827	3E 01	Repeat the sequence and call KEY VALUE once more.
LD (09FE),A	0829	32 FE 09	
CALL KEY VALUE	082C	CD 30 09	
LD A,(09FC)	082F	3A FC 09	Load the first figure into A and rotate the accumulator
RLA	0832	17	4 places to the left to shift the number into the upper
RLA	0833	17	half of the register.
RLA	0834	17	
RLA	0835	17	
ADD A,C	0836	81	Add the second figure to the accumulator and store
LD (09FC),A	0837	32 FC 09	the result into **09FC** as a two figure decimal number.
LD D,20	083A	16 20	Create a delay with register D and call SCAN for 20H
CALL SCAN	093C	CD 80 09	loops. (32 loops).
DEC D	083F	15	
JR NZ,083C	0840	20 FA	
CALL CLEAR	0842	CD 20 09	Clear the display and load the pointer register with the
LD HL,0A2C	0845	21 2C 0A	start address of the second table. Display "Enter
CALL SCROLL	0848	CD C0 09	name . . . . .etc" Look for the end of the table (**10**) and
LD A,(HL)	084D	7E	loop, unless a key 0-9 has been pressed.
CP 10	084C	FE 10	
JR Z,0845	084E	28 F5	
CP 0A	0850	FE 0A	
JR C,0842	0852	38 EE	
CALL CLEAR	0854	CD 20 09	Call CLEAR to clear the display.
CALL MEM ADDR	0857	CD 60 09	Read MEMORY ADDRESS notes.
LD D,1C	085A	16 1C	Register D counts up to 28 characters (max allowed).
LD E,00	085C	1E 00	Register E counts to 2. Two key presses for a char.
LD A,FF	095E	3E FF	Fill the I register via the accumulator so that we can
LD I,A	0860	ED 47	detect when a key is pressed.
CALL SCAN 2	0862	CD D0 0A	Scan the display looking for a key press 0-F.
LD A,I	0865	ED 57	
CP 10	0867	FE 10	
JR NC,0862	0869	30 F7	
INC E	086B	1C	Increment the E register.
LD A,E	086C	7B	Load E into A.
CP 02	086D	FE 02	Compare the accumulator with 02 and jump if the two
JR Z,087C	086F	28 0B	are the same. If not, go to the next instruction.
LD A,I	0871	ED 57	Look to see if a space is required as this will indicate
CP 0F	0873	FE 0F	the end of names and the beginning of numbers.
JR Z,0895	0875	28 1E	Jump relative if F has been pressed.
LD (09FA),A	0877	32 FA 09	Store the value of A at **09FA** and loop for second press
JR 085E	087A	18 E2	of button.
CALL SHIFT	087C	CD E1 09	Call SHIFT to get display ready for next number.
LD A,(09FA)	087F	3A FA 09	Load the first number into the accumulator and shift
RLA	0882	17	4 places to the left to occupy the upper half of the
RLA	0883	17	register.
RLA	0884	17	
RLA	0885	17	
LD B,A	0886	47	Save the result in B.
LD A,I	0887	ED 57	Put second number into the accumulator.
ADD A,B	0889	80	Combine the two to create a 2-digit number.
LD (HL),A	088A	77	Load this value into the location looked at by HL.
LD (0A85),A	088B	32 85 0A	Also load it into the first display location.
INC HL	088E	23	Increment HL.
DEC D	088F	15	Decrement D and
JR NZ,085C	0890	20 CA	Jump if **1C** locations not filled.
JP 0800	0892	C3 00 08	Jump to start if overflow occurs.
XOR A	0895	AF	Zero A and load it
LD (HL),A	0896	77	into the location looked at by HL to create a space.
CALL SHIFT	0897	CD E1 09	Shift the display digits one place to the left.
LD A,D	089A	7A	Load the remaining locations into A and store at **09FE**
LD (09FE),A	089B	32 FE 09	for use by the CALL KEY routine.
CALL KEY VALUE	089E	CD 30 09	Call KEY VALUE. This will put Nos onto the display.
LD B,03	08A1	06 03	Create 3 blank locations after the numbers have been
INC HL	08A3	23	inserted, to produce a space between the end of the
XOR A	08A4	AF	message and the start so that it can be scrolled across
LD (HL),A	08A5	77	the display.
DEC B	08A6	05	
JR NZ,08A3	08A7	20 FA	
INC HL	08A9	23	
LD A,10	08AA	3E 10	Increment HL and load last location with **10** so that
LD (HL),A	08AC	77	program will loop name and telephone number.
NOP	08AD	00	

this lower location. As can be seen from the program, this lower location is not displayed on the TEC and thus the data shifts off the screen. The data for the second location is shifted to the location for the first display and this repeats for the 6 locations. The result is the data in the blank location at **0A86** is shifted into the last display location and thus an empty space is produced on the display.

It is important for **0A86** to be empty for this to work.

The MEMORY ADDRESS routine creates areas that are 20H bytes long and starts at **0B00**.

The program stores the Index number at location **09FC** and as each memory area is created, it decrements the Index number and the program exits when the count register is zero.

The HL register will contain the start of this address. It is not used for any other purpose and thus it will not be destroyed during the running of the program and will hold the current value for re-dial, if required.

The SCROLL routine picks up the first byte from the table and places it at **0A85** and then calls SCAN for 20H loops (32 passes of the display).

The SHIFT routine is then called and all the bytes (including the blank locations) are transferred one position to the left.

The scroll program then loops and repeats the sequence until the end of the table is reached. It detects this by looking for 10H (we could have chosen any value) and the message re-starts.

When the 'Dial key' '**D**' is pressed, a BEEP routine and PAUSE routine are called. These produce a suitable ON-OFF tone to the speaker and the program converts the values in memory to a string of beeps.

The program ignores the name at the beginning of memory and looks for the first location containing zero.

The end of the phone number is detected by also looking for a location containing zero.

The program then jumps back to calling the start of memory and scrolls the message across the screen.
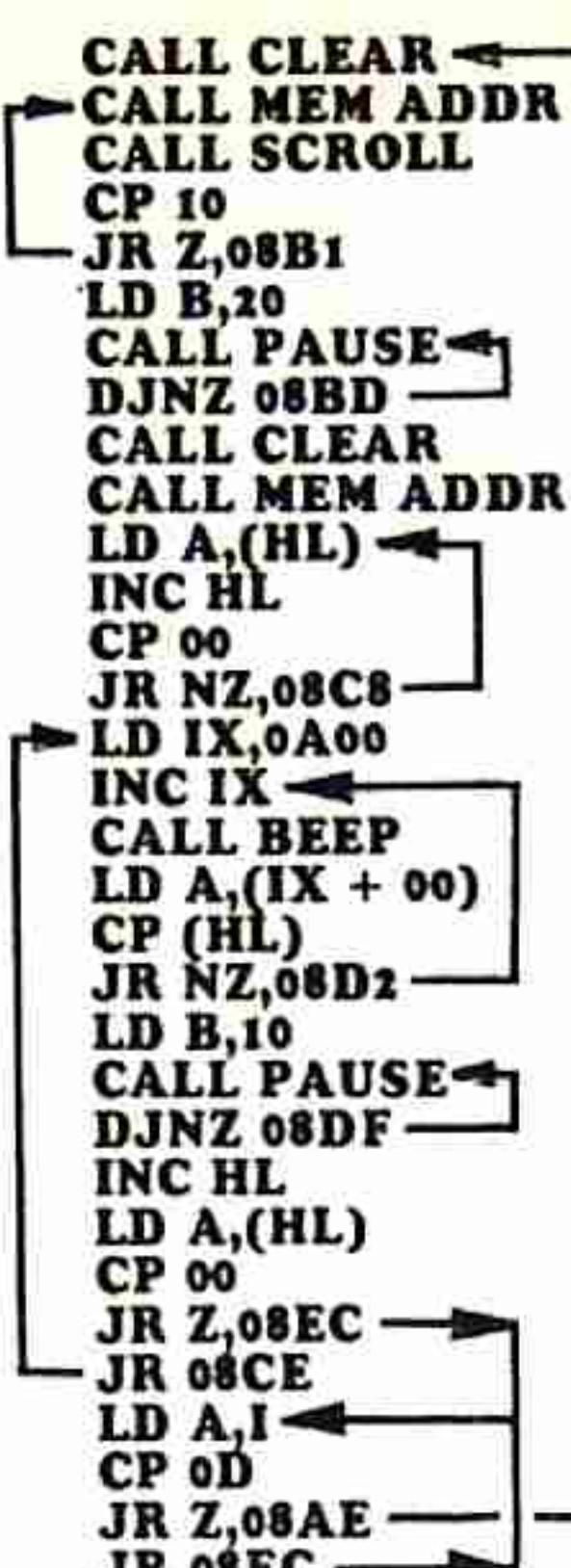
SUGGESTIONS

The program can be keyed into the TEC and fills about 3 pages, from **0800** to **0AEE**.
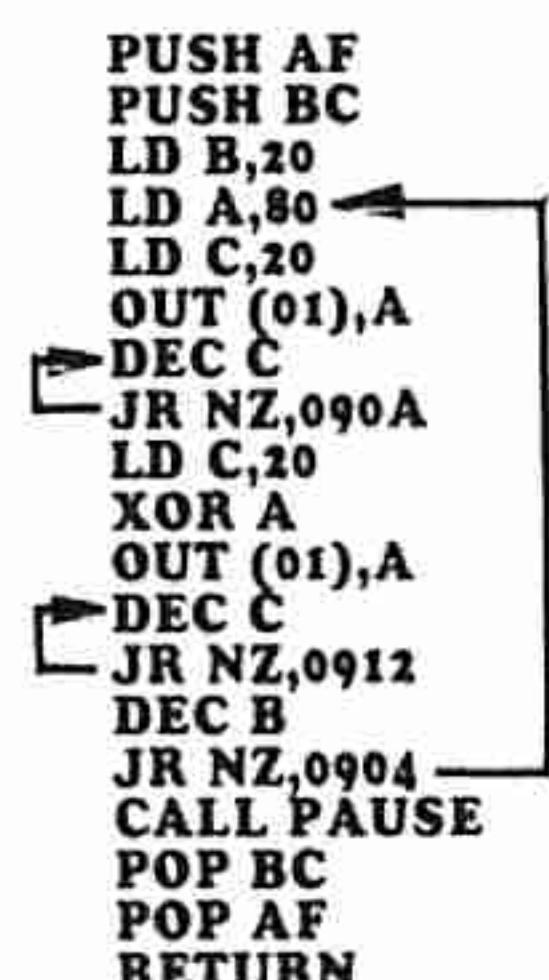
After this is done, it is wise to save a copy of the program in non-volatile RAM so that it is not lost.

To save the program, type the following dump routine at **0F80**:
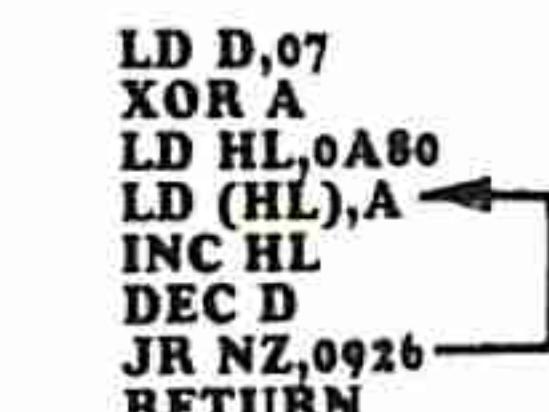
```
11 00 10
21 00 08
01 90 07
ED B0
C7
```
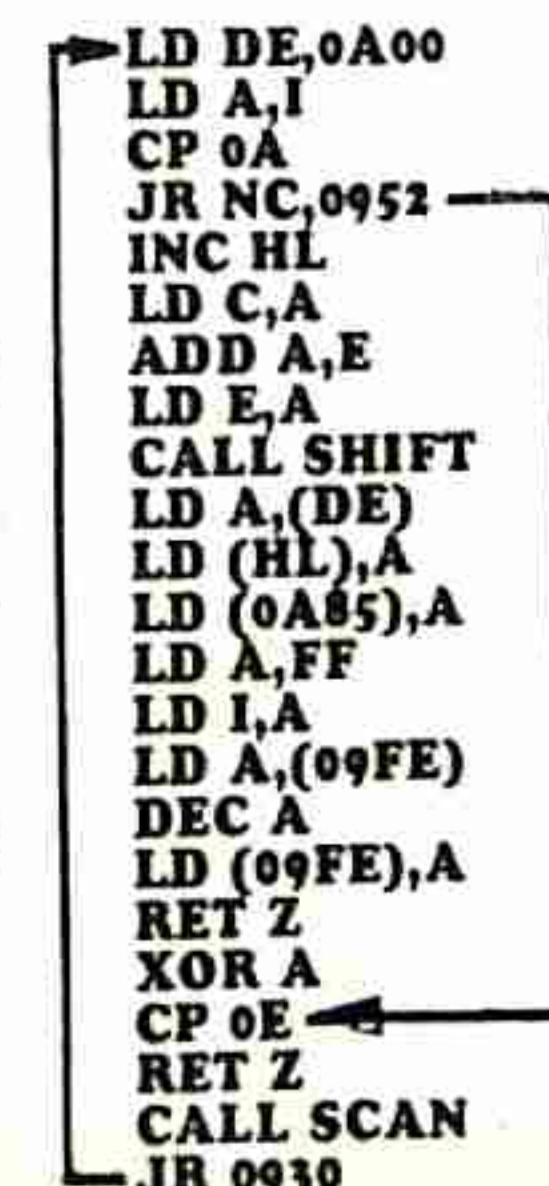
Mnemonic	Addr	Code	Comment
CALL CLEAR	08AE	CD 20 09	Clear the screen.
CALL MEM ADDR	08B1	CD 60 09	Get start of BLOCK via **09FC** (36 blocks available).
CALL SCROLL	08B4	CD C0 09	Scroll name and number across screen.
CP 10	08B7	FE 10	Look for end of message. If another key is pressed,
JR Z,08B1	08B9	28 F6	jump out of loop.
LD B,20	08BB	06 20	Create a pause before dialling by loading B with 20
CALL PAUSE	08BD	CD 72 09	and calling pause 32 times. This creates approx 2
DJNZ 08BD	08C0	10 FB	second delay.
CALL CLEAR	08C2	CD 20 09	Clear the screen of any junk etc.
CALL MEM ADDR	08C5	CD 60 09	Get start of block (00-36).
LD A,(HL)	08C8	7E	Look for space between name and phone number by
INC HL	08C9	23	comparing the contents of each location with 00 and
CP 00	08CA	FE 00	incrementing until 00 is found.
JR NZ,08C8	08CC	20 FA	The next 6 lines create the dialling pulses by loading
LD IX,0A00	08CE	DD 21 00 0A	IX with the start of the number table and calling BEEP
INC IX	08D2	DD 23	routine. (The beep calls a pause). The program then
CALL BEEP	08D4	CD 00 09	compares the byte in the table with the byte in the
LD A,(IX + 00)	08D7	DD 7E 00	block and loops until a comparison is found. Note: we
CP (HL)	08DA	BE	go into the routine 'blind' and beep before a CP!!
JR NZ,08D2	08DB	20 F5	Create a short pause at the end of each digit so that
LD B,10	08DD	06 10	the phone system detects the end of a digit.
CALL PAUSE	08DF	CD 72 09	Increment to next digit, look to see if end of phone
DJNZ 08DF	08E2	10 FB	number has been reached and return to above routine
INC HL	08E4	23	for next set of pulses.
LD A,(HL)	08E5	7E	
CP 00	08E6	FE 00	
JR Z,08EC	08E8	28 02	
JR 08CE	08EA	18 E2	
LD A,I	08EC	ED 57	If no buttons have been pressed during dialling, I will
CP 0D	08EE	FE 0D	still contain 0D (from above) and program will scroll
JR Z,08AE	08F0	28 BC	name and number. If any other key has been pressed,
JR 08EC	08F2	18 F8	program will loop with blank screen until D pressed.

This is the end of the MAIN PROGRAM. The subroutines below are called by the main program.

BEEP

Mnemonic	Addr	Code	Comment
PUSH AF	0900	F5	Registers A, B and C are used in this sub-routine and
PUSH BC	0901	C5	thus they must be pused onto the stack and saved.
LD B,20	0902	06 20	Reg B holds the number of cycles for the beep routine
LD A,80	0904	3E 80	Register A turns on the speaker bit.
LD C,20	0906	0E 20	Reg C holds the turn-on cycles for the spkr.
OUT (01),A	0908	D3 01	The spkr is turned on via **OUT (01),A**
DEC C	090A	0D	and a delay created via register C for
JR NZ,090A	090B	20 FD	32 loops.
LD C,20	090D	0E 20	The same OFF delay period is created via register C
XOR A	090F	AF	for an even 'mark-space' ratio for the speaker.
OUT (01),A	0910	D3 01	
DEC C	0912	0D	
JR NZ,0912	0913	20 FD	
DEC B	0915	05	The count register (register B) is decremented and the
JR NZ,0904	0916	20 EC	program loops until B is zero.
CALL PAUSE	0918	CD 72 09	The program calls pause to produce silence.
POP BC	091B	C1	Registers A,B and C are popped off the stack and will
POP AF	091C	F1	contain the original values and before the routine.
RETURN	091D	C9	Return to the main program.

CLEAR

Mnemonic	Addr	Code	Comment
LD D,07	0920	16 07	This routine clears the 6 display locations **0A80** to
XOR A	0922	AF	**0A85** and also **0A86** by zeroing A and
LD HL,0A80	0923	21 80 0A	loading HL with start address of buffer zone
LD (HL),A	0926	77	and loading zero into the location pointed to by HL
INC HL	0927	23	INC HL
DEC D	0928	15	DEC D
JR NZ,0926	0929	20 FB	and jump for 7 loops.
RETURN	092B	C9	Return to main program.

KEY VALUE

Mnemonic	Addr	Code	Comment
LD DE,0A00	0930	11 00 0A	Load DE to point to beginning of number table.
LD A,I	0933	ED 57	Load key value into accumulator.
CP 0A	0935	FE 0A	Compare with 0A and jump if the key value is A-F or
JR NC,0952	0937	30 19	not pressed or go to next instruction if 0-9.
INC HL	0939	23	INC HL (used when creating phone number)
LD C,A	093A	4F	Save A in C.
ADD A,E	093B	83	ADD the start of table to A (table may start at 0A03!).
LD E,A	093C	5F	Make DE ready to point at value in table.
CALL SHIFT	093D	CD E1 09	SHIFT display contents one place to left.
LD A,(DE)	0940	1A	Load byte from number table into accumulator.
LD (HL),A	0941	77	Load number byte into loaction in BLOCK.
LD (0A85),A	0942	32 85 0A	and also into right hand display.
LD A,FF	0945	3E FF	Load A with FF and then into I to detect when another
LD I,A	0947	ED 47	key has been pressed.
LD A,(09FE)	0949	3A FE 09	**09FE** caontains 01 via beginning of of main program
DEC A	094C	3D	and KEY VALUE is called once. Or **09FE** contains **1C**
LD (09FE),A	094D	32 FE 09	to keep track on the number of locations being filled in
RET Z	0950	C8	the BLOCK.
XOR A	0951	AF	Zero A.
CP 0E	0952	FE 0E	Compare accumulator with E and RETURN if E key is
RET Z	0954	C8	pushed. Otherwise call SCAN and display the
CALL SCAN	0955	CD 80 09	contents of the 6 memory locations. Jump to stat of
JR 0930	0958	18 D6	KEY VALUE sub-routine and loop until 0-9 pressed.

Decrement to **0F80** and push GO. Make sure the non-volatile RAM switch is on RAM (read/write) so that the data will be accepted. Check that the program has been dumped by addressing **1000** and compare the data with the listing.

If you have inserted names and numbers into index locations and want to save them, address **0F80** and push GO. Make sure the RAM card is in read/write mode and everything will be saved.

Switch to ROM mode and everything will be preserved.

You can now turn the TEC off.

To transfer the program back to **0800**, address **1780** and change 2 of the bytes to the following:

11 00 08 ◄ *these two bytes*
21 00 10 ◄ *are changed*
01 90 07
ED B0
C7

Decrement to **1780** and push GO. The RAM card should be in ROM MODE for this operation.

Push GO again and the program will run.

All names and numbers will be available.

AUTO REDIAL

An automatic re-dial facility can also be included so that the number automatically re-dials after say 5 or 10 minutes; if the number was originally engaged. This is very handy for those occasions when you particularly want to contact a person and their number is busy. By the time you get around to calling again, they have gone!

A simple addition to the program can be fitted in at **08BE** and this will create a delay by counting the number of times the name and phone number scroll past the display. This is only a suggestion and we have not actually produced the program for re-dial.

Register E is the 'count register' and the remainder of the program remains the same. The only bytes you will have to change are jump relative values as well as the jump value at **09B4**. You may also need a subroutine and a flag to pick up redial mode.

Here is a suggested AUTO RE-DIAL program for insertion at **08B4**:

LD E,40
DEC E
JR Z
CALL CLEAR
CALL MEMORY ADDR
CALL SCROLL
CP 10
JR Z
CALL CLEAR

JR

MEMORY ADDRESS

LD HL,0B00	0960	21 00 0B
LD A,(09FC)	0963	3A FC 09
LD D,20	0966	16 20
CP 00	0968	FE 00
RET Z	096A	C8
INC HL	096B	23
DEC D	096C	15
JR NZ,096B	096D	20 FC
DEC A	096F	3D
JR 0966	0970	18 F4

PAUSE

XOR A	0972	AF
OUT (01),A	0973	D3 01
LD DE,02FF	0975	11 FF 02
DEC DE	0978	1B
LD A,E	0979	7B
OR D	097A	B2
JR NZ,0978	097B	20 FB
RETURN	097D	C9

SCAN 1

PUSH HL	0980	E5
PUSH DE	0981	D5
LD C,20	0982	0E 20
LD HL,0A80	0984	21 80 0A
LD D,06	0987	16 06
LD B,20	0989	06 20
LD A,(HL)	098B	7E
OUT (02),A	098C	D3 02
LD A,C	098E	79
OUT (01),A	098F	D3 01
RRC C	0991	CB 09
DJNZ 0993	0993	10 FE
XOR A	0995	AF
OUT (01),A	0996	D3 01
INC HL	0998	23
LD A,I	0999	ED 57
CP 0C	099B	FE 0C
JR Z,09A9	099D	28 0A
CP 0D	099F	FE 0D
JR Z,09B2	09A1	28 0F
DEC D	09A3	15
JR NZ,0989	09A4	20 E3
POP DE	09A6	D1
POP HL	09A7	E1
RETURN	09A8	C9
POP DE	09A9	D1
POP HL	09AA	E1
LD A,FF	09AB	3E FF
LD I,A	09AD	ED 47
JP 0800	09AF	C3 00 08
POP DE	09B2	D1
POP HL	09B3	E1
JP 08BB	09B4	C3 BB 08

SCAN 2

PUSH HL	0AD0	E5
PUSH DE	0AD1	D5
LD C,20	0AD2	0E 20
LD HL,0A80	0AD4	21 80 0A
LD D,06	0AD7	16 06
LD B,20	0AD9	06 20
LD A,(HL)	0ADB	7E
OUT (02),A	0ADC	D3 02
LD A,C	0ADE	79
OUT (01),A	0ADF	D3 01
RRC C	0AE1	CB 09
DJNZ 0AE3	0AE3	10 FE
XOR A	0AE5	AF
OUT (01),A	0AE6	D3 01
INC HL	0AE8	23
DEC D	0AE9	15
JR NZ,0AD9	0AEA	20 ED
POP DE	0AEC	D1
POP HL	0AED	E1
RETURN	0AEE	C9

Memory Address sub-routine locates the beginning of the name and phone number block. Each block is 20H bytes long (32 bytes) and memory starts at **0B00**. The BLOCK No is stored at **09FC** and the program increments 20H loops for each block by decrementing register D to zero, then decrementing register A by ONE. This is repeated until A is zero. The sub-routine then exits. HL pair is constantly incremented during this program and will point to the start of the block we want.

Pause produces a silence from the speaker by outputting zero to port 01. Register DE is decremented and 'wastes computer time' for about 1/10th second. This sub-routine then returns to where it has been called.

The SCAN routine uses H, L and D registers and thus they must be pushed onto the stack and saved. Load HL with start of display buffer. The routine displays 6 locations. The left-hand display is accessed via line '20'. Load B with a short delay value. Load the byte at the first location into A. Output to port 02. Load C into A, and output to port 01. This will turn on left-hand display. Rotate register C to the right for the next display. Short delay via register B. Zero A, and output to port 01. Look at next memory location. Load the keyboard value into A. Look to see if CLEAR has been pressed. Jump if it has. DEC D ready for outputting to the next display. Jump relative if D is not zero. Pop DE and HL register pairs off the stack.

and RETURN to the main program. If CLEAR has been pressed, pop DE and HL and load the I register with **FF** so that the program will detect when another key has been pressed.

Jump to **0800**.
POP DE and HL and jump to **08BB** if D (DIALS) has been pressed.

SCAN 2 is identical to SCAN 1 in the scanning section. The only difference is the 'checking' instructions, to see if a particular key is pressed. SCAN 1 above checks to see if a function key is pressed, whereas SCAN 2 performs the scan without any checks.
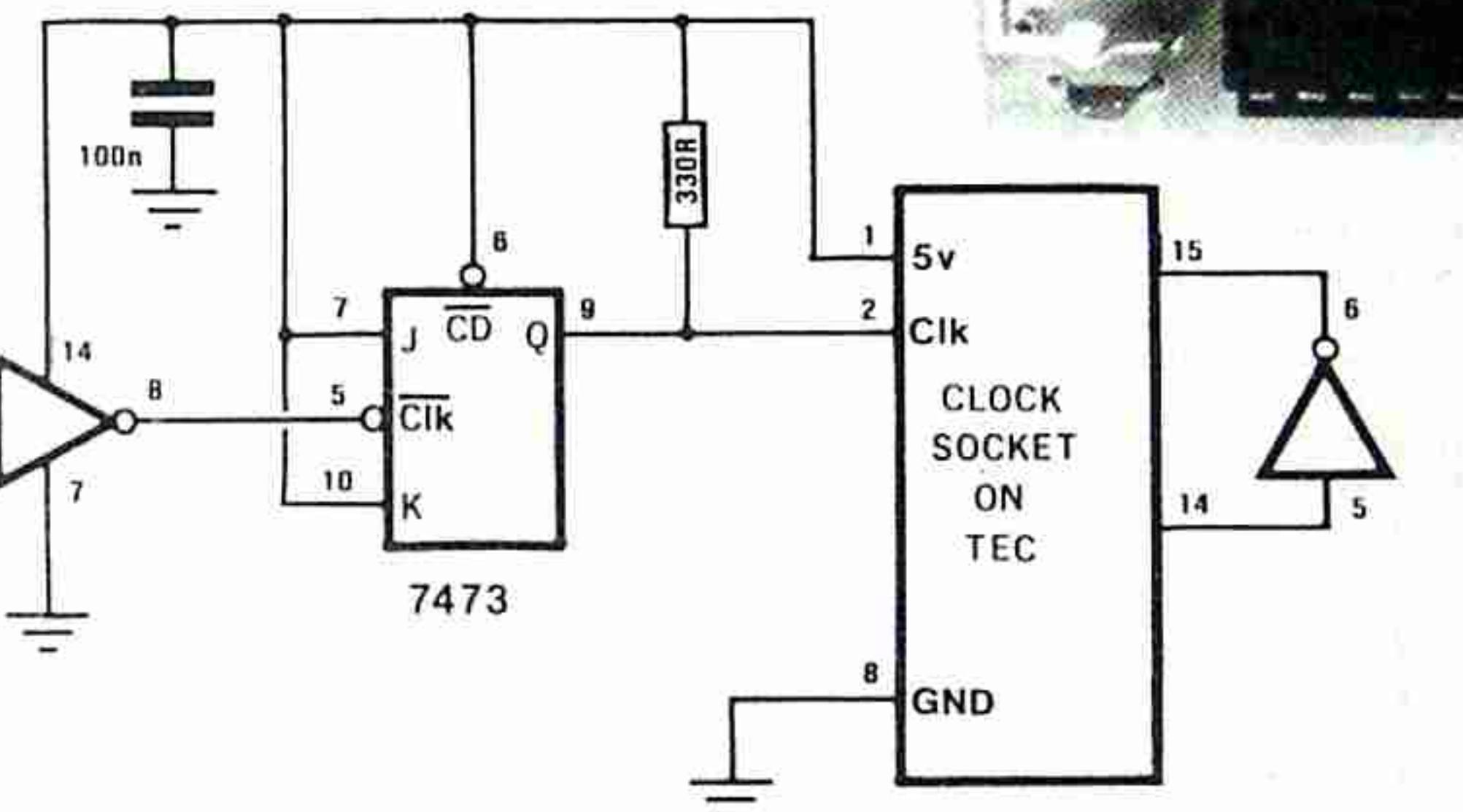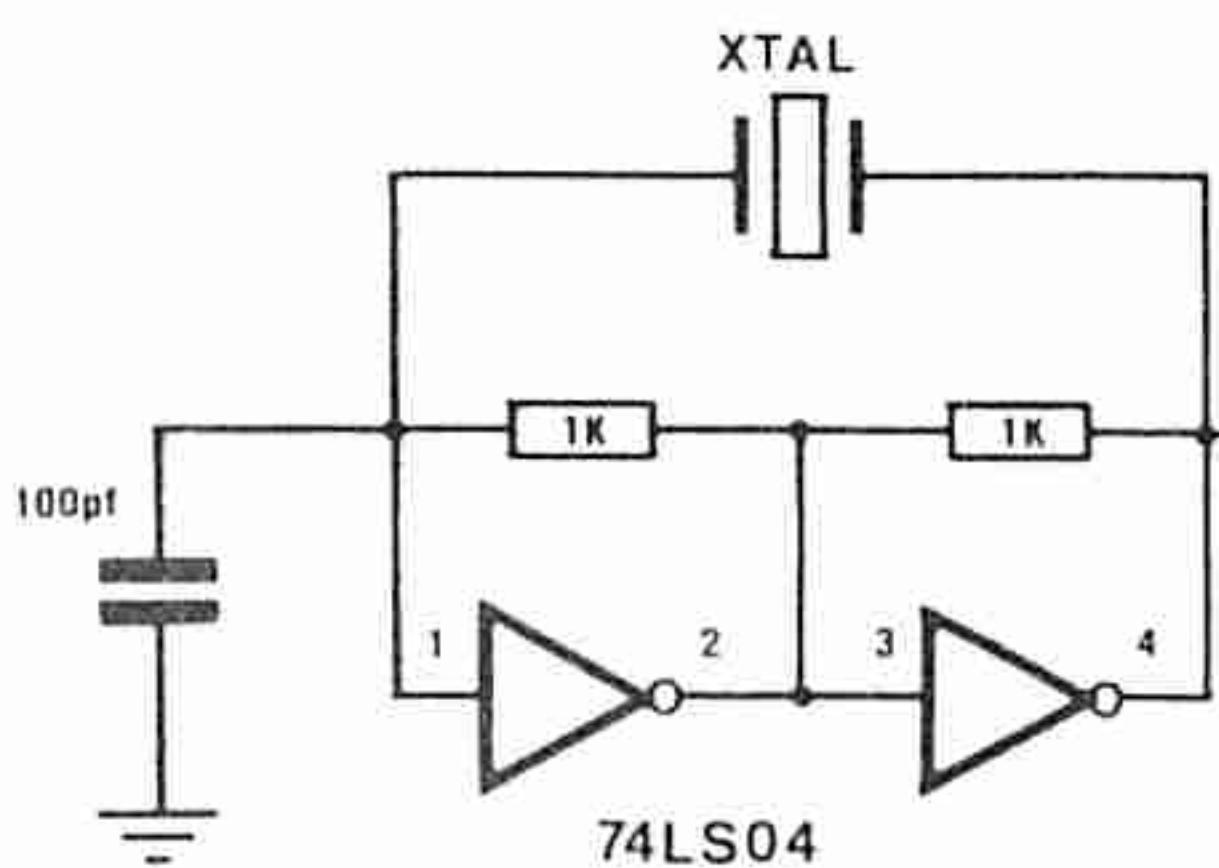
By careful programming both routines could be incorporated into one. This would require a 'check bit' and if 'set', the sub-routine would check the function keys.

CRYSTAL OSCILLATOR

CONVERTS THE TEC TO REAL-TIME CAPABILITY

CRYSTAL OSCILLATOR CIRCUIT

This project is a crystal oscillator for the TEC. It turns the TEC into a fixed-frequency computer in which each of the Machine Codes takes up a precise period of time.

This means programs such as controller programs or timing programs will run for a precise time span and will not vary from one day to the next due to speed control adjustments.

As you know, the TEC was originally designed with an adjustable clock and its frequency could be altered by turning the speed control.

This served a valuable purpose as the games of skill (contained in the MONitor ROM) could be adjusted according to the skill of the player.

It also proved that the Z-80 could be run at very low speeds and even adjusted while operating and still execute the programs correctly.

The only disadvantage of a variable speed control is its inability to create accurate REAL-TIME programs.

This is highlighted by the clock program (as presented in issue 12). Everyone expects a clock to keep accurate time as even 'two dollar' watches are accurate to two seconds a month. The clock program could only approach this accuracy as it had to be manually adjusted via the speed control.

To remedy this situation Paul has produced a crystal oscillator module that plugs into the 4049 socket.

It contains an inverter chip (74LS04) and a divider chip so that a 4MHz crystal or colour-burst (3.5795MHz) can be used (because they are cheap) and a divider chip (7473) to divide the frequency by two so that the TEC will run at about the maximum speed permissible for a Z-80 CPU.

The 7473 is wired in TOGGLE mode to provide a divide-by-two output.

Some of the earlier model TEC's used a Z-80 CPU (later models used a Z-80A as these were cheaper than the Z-80!!) and the maximum operating speed for a Z-80 is about 2.5MHz.

Almost any crystal can be used in this circuit providing it is in the range 1MHz to 5MHz for a Z-80 or up to 8MHz for a Z-80A. If a crystal other than 4MHz or colour-burst is used, it will be necessary for you to carry out your own conversion for timing etc, if a real-time situation is required.

An inverter is also necessary to invert the Data Available line from the keyboard encoder to the NMI line of the Z-80 so that the NMI line goes low when data is available from the keyboard encoder. This is provided via one of the unused inverters of the 74LS04.

The oscillator circuit is a simple twin inverter using feedback resistors.

A 100pf capacitor at the front end provides guaranteed start-up and the crystal provides a capacitive feedback that is a maximum at the fundamental frequency of the crystal.

This is why the oscillator circuit operates at the frequency as specified on the crystal.

A 100n capacitor on the oscillator module reduces noise on the power rails and a 330R pull-up resistor in the clock line guarantees a full amplitude waveform for the Z-80.

To convert the TEC to crystal control, remove the 4049 and plug in the crystal oscillator board. The speed control pot will have no effect and the speed of execution of the monitor will be about double.

This will too fast for many of the games and you may have to convert back to the adjustable speed by replacing the 4049 by pressing the reset button and keeping it pressed while changing over the clocks.

PARTS

1 - 330R
2 - 1k

1 - 100pf ceramic
1 - 100n monoblock

1 - 3.5795MHz crystal

1 - 74LS04 IC

1 - 7473 IC

2 - 14 pin IC sockets
1 - 16 pin dip header

1 CRYSTAL OSCILLATOR PC BOARD

All future programs will have to be written especially for the new speed and this will mean delay values etc will have to be lengthened accordingly.

ASSEMBLY

Assembly is very simple and we suggest, as always, that the two chips be fitted via IC sockets. The two 1k resistors stand upright and the 330R lays flat against the PC board. The leads of the crystal must be left long enough to allow the crystal to lay over after it has been soldered and a wire strap placed over the body to prevent it being damaged, as the leads are very thin.

The 100pf and 100n are fitted against the PC board and soldered in the positions shown. Don't get them swapped over or the oscillator won't work!

The module is connected to the TEC via a 16 pin dip header soldered under the board.

If the cermet pot on the TEC is a stand-up version, it will be necessary to include a wire-wrap socket between the dip header and the board to create additional clearance for the pot. This is not supplied in the kit as you can fold the cermet pot over slightly to allow the clock board to fit.

When you have the new board in place, the first program you can try is the Clock in issue 12, P.23. The best idea is to type

it into the non-volatile RAM at **1000** and down-load it to **0900** via a block-transfer program:

```
11  00  10
21  00  09
01  A0  00
ED  B0
C7
```

To convert the program to operate with 4MHz crystal, two of the inbuilt delay values must be altered and a 'fine tune' delay added to the end of the program. This will create a clock that is accurate to within a second a day.
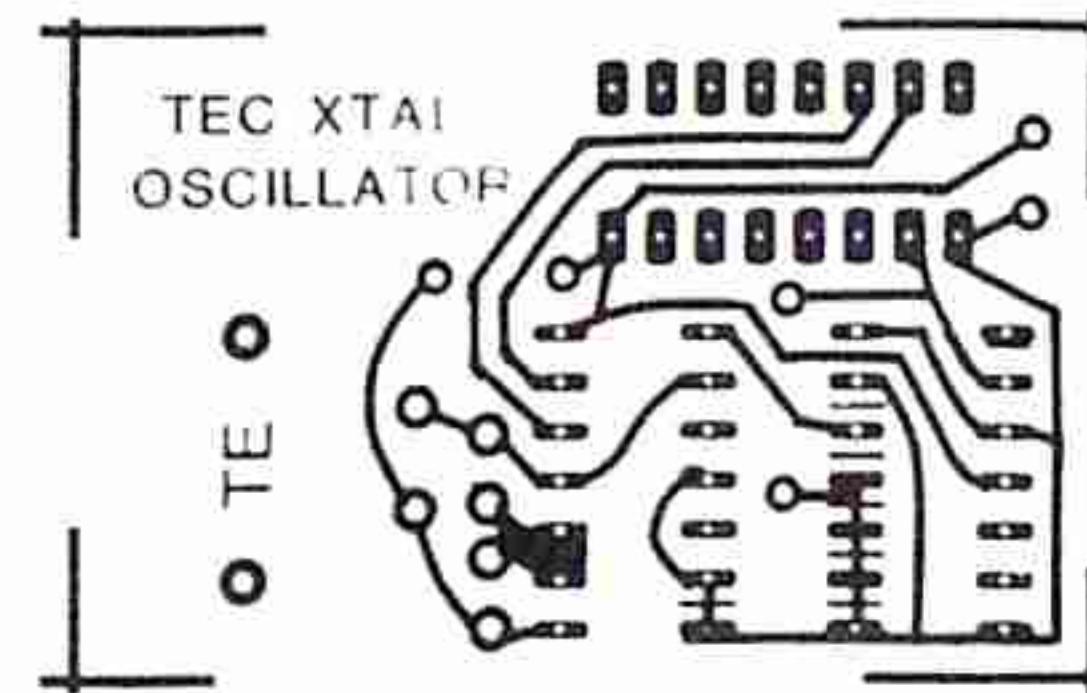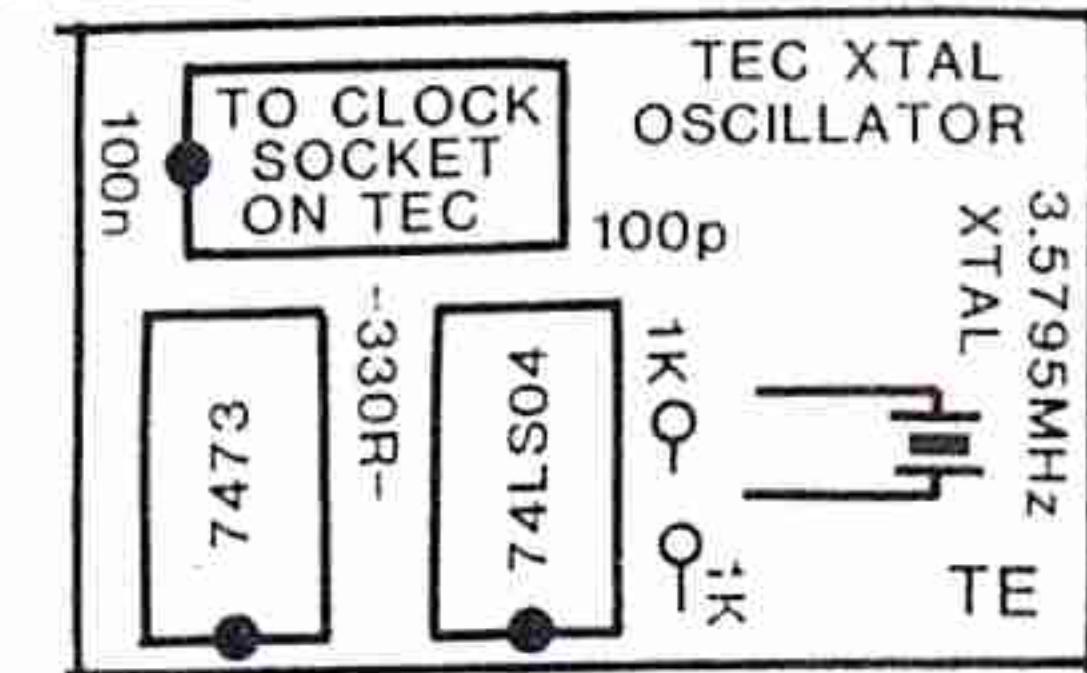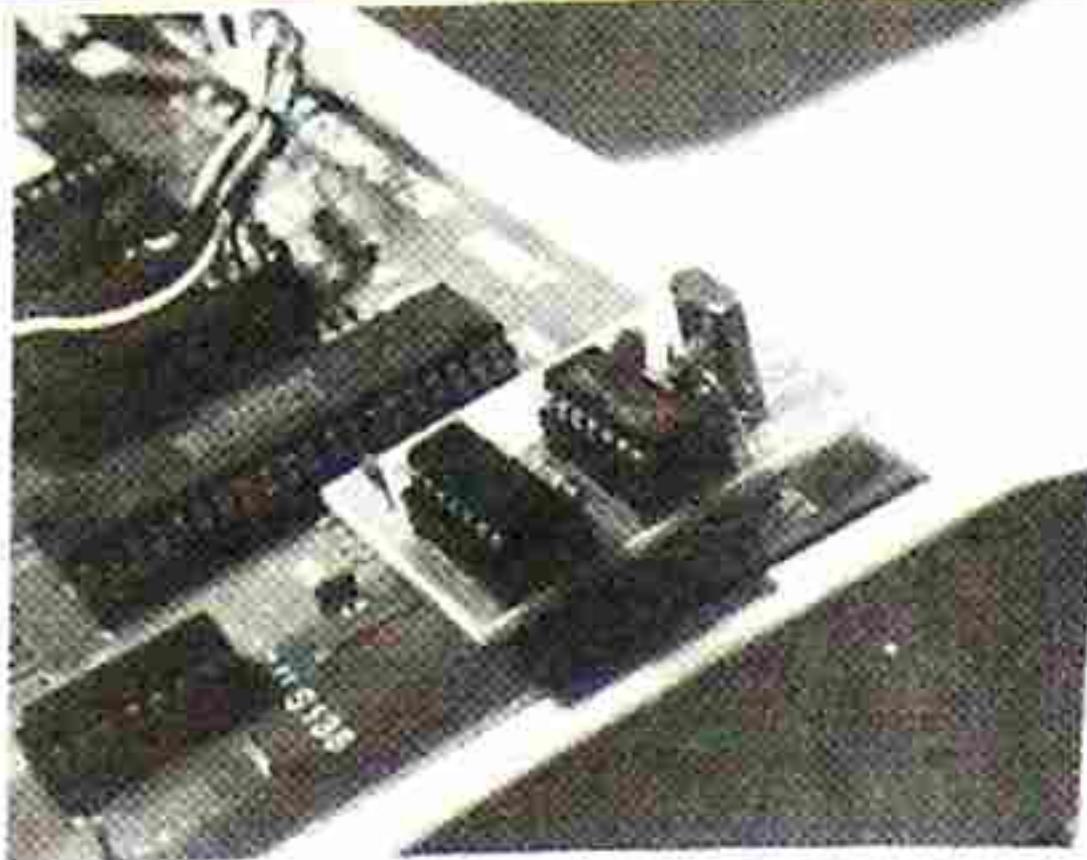
Type the complete program as per issue 12 then change the following locations and also add the extra 7 bytes:

For a 4MHz crystal:

```
94C   06 FA
962   1E 41
970   C3 93 09
993   06 55
995   10 FE
997   C3 00 09
```
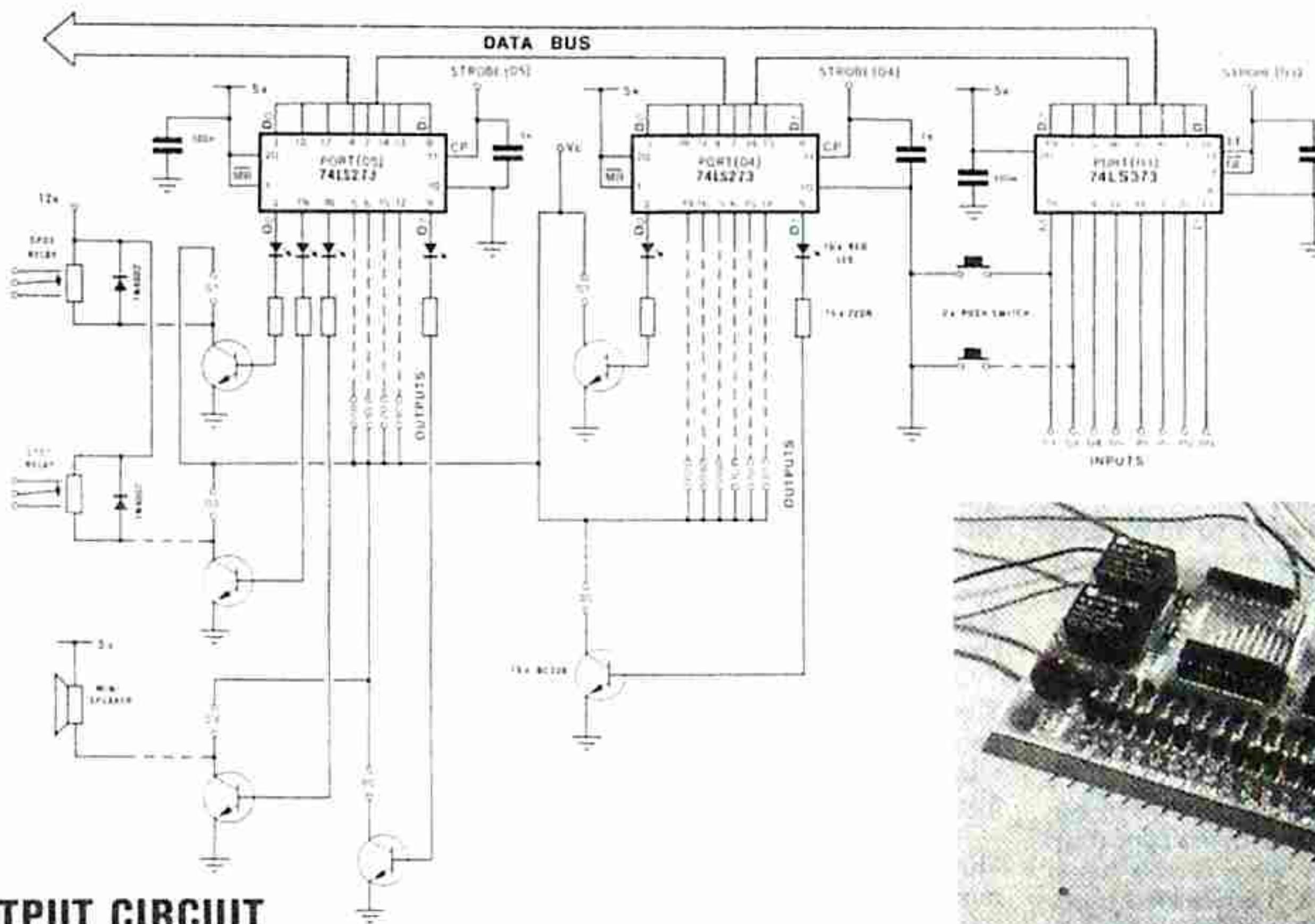
For a 3.5795MHz crystal:

```
94C   06 FC
962   1E 39
970   C3 93 09
993   06 37
995   10 FE
997   C3 00 09
```


TEC XTAL OSCILLATOR

TO CLOCK SOCKET ON TEC

100n 100p

7473 -330R- 74LS04 1KΩ ¼K

3.5795MHz XTAL TE

TEC XTAL OSCILLATOR

TE

INPUT/OUTPUT MODULE

Kit of parts: $33.80
PC Board: $5.00
Complete: $38.80

DATA BUS

INPUT/OUTPUT CIRCUIT

This project allows the TEC to talk to the outside world and also accept information from the outside. It is the first interface we have described that brings the possibility of robotics to the TEC.

The INPUT/OUTPUT MODULE has one input port and two output ports. This means it will input 8 bits (8 lines) and output 16 bits (16 lines).

To allow the module to be functional as soon as it is constructed we have included two input switches and three output devices so that a simple program can be written and seen in operation. The output devices are two relays and a mini speaker. These will allow you to test the board and see how it operates, before adding any other devices.

We have included some test programs in the article and they will show the indicator LEDs in operation.

These LEDs indicate when a particular output is high and will be invaluable when trouble-shooting a fault in either a program or in hardware.

The 5 flying leads on the module are clearly marked and you will see the input port is controlled via strobe line 03 and output ports via strobe lines 04 and 05.

Each of the 8 input and 16 output lines is further identified by a hex value on the PC overlay and this will assist you when writing a program.
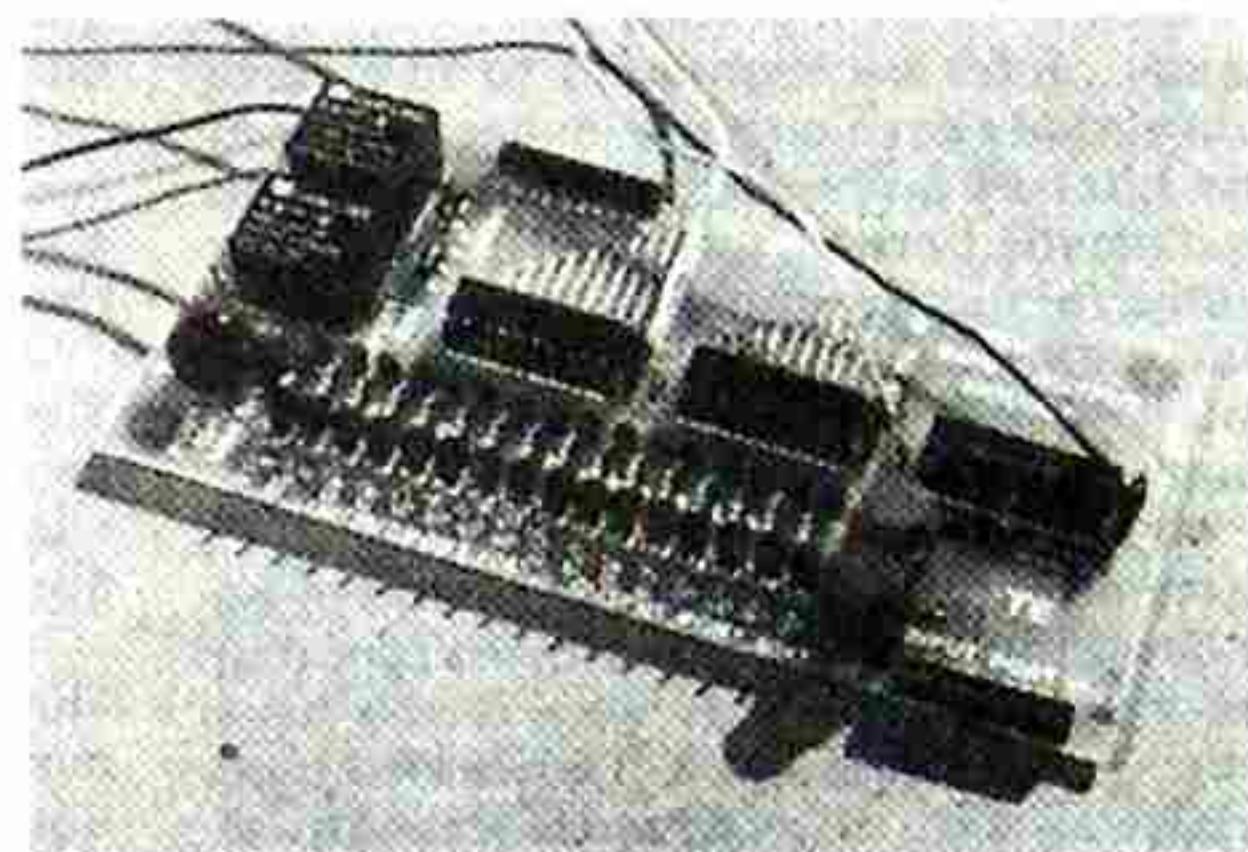
The most interesting use for the board will undoubtedly be for robotics and when designing in this field, a whole new world of mechanical and electromechanical terms will be encountered.

Before embarking on a design, it is important to have some idea of what you are going to create. It may be an arm, a wheeled vehicle or a mechanical controller such as a door opener, a lift, crane or remote controlled boat or plane.

No matter what the project, begin by collecting articles and notes describing similar or related devices and study how other designers have puts things together. Combine the features you like and make sketches and diagrams of how you indend yours to look.

The most important point is not to be too ambitious on your first attempt. Aim for a simple design, using maybe a single motor and gearbox with say one or two flashing lights and a speaker.

You will have sufficient interfacing problems with these to keep your inventive skills at work for a while.

The other point to remember is to select materials that you can readily obtain and don't choose thick material as this will be very difficult to work with.

PARTS

16 - 220R ¼watt

3 - 1n greencap
2 - 100n

2 - 1N 4002 diodes
16 - 3mm red LEDs
16 - BC 338 transistors

2 - 74LS273 IC
1 - 74LS373 IC

3 - 20 pin IC sockets
2 - PC mount push buttons
1 - Mini Speaker 80R
2 - SPDT relays

50cm tinned copper wire
5 - PC matrix pins
5 - Matrix connectors
10cm - Heatshrink tubing
15 - 20cm lengths of hook-up flex

20cm - 10 core ribbon cable
1 - 12 key telephone pad

1 - INPUT/OUTPUT MODULE PC

3mm clear plastic sheet is the best choice as it can be cut, bent, folded and even heated into shape. It also looks appealing and being clear, you can see through it and this makes the project look more complex!

Equally suitable is PC board as it has a copper surface that can be soldered to and thus small brackets can be added for shafts etc.
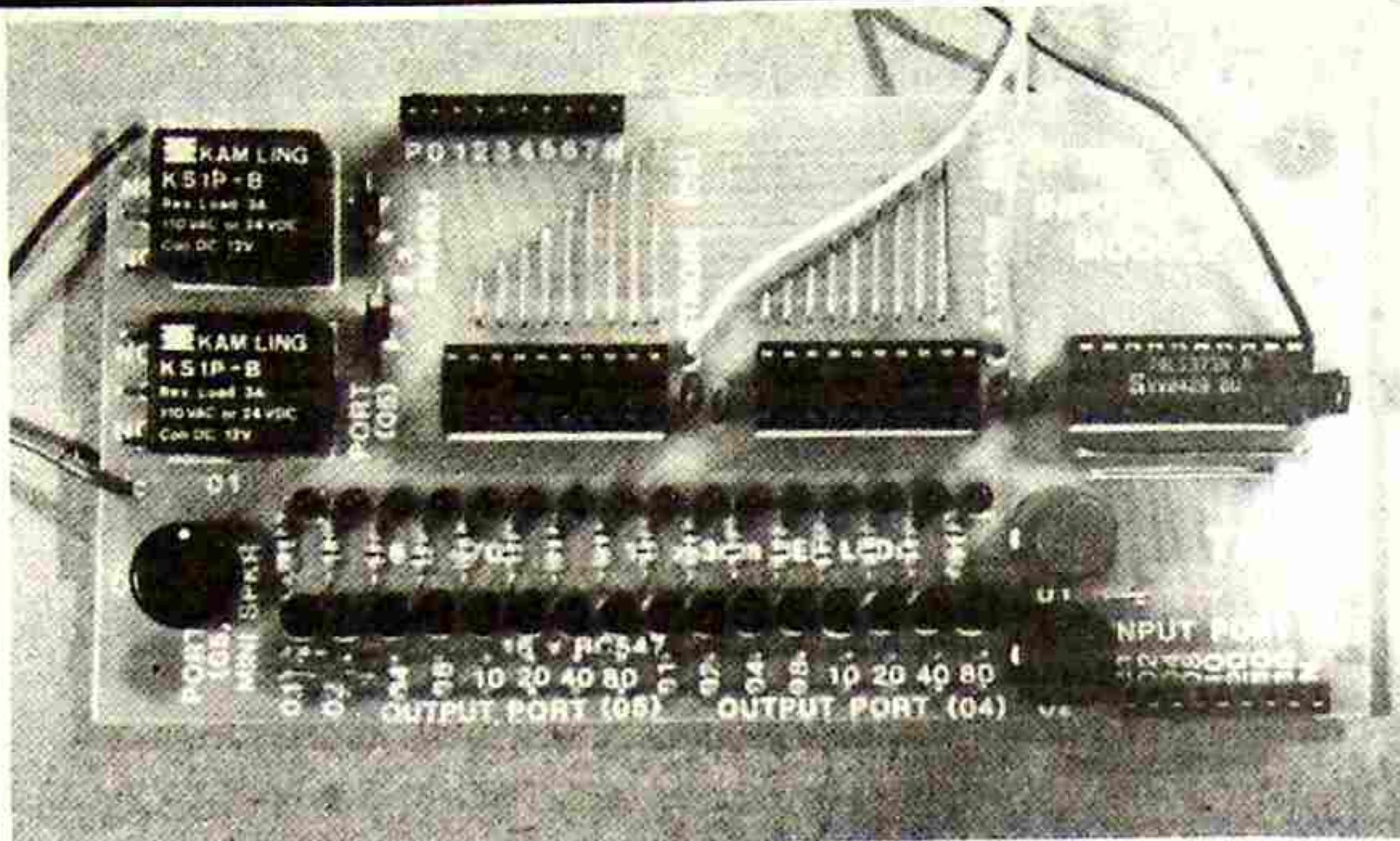
The only material I would avoid is sheet metal. Even though it has good strength, the same can be provided via plastic with the use of a few strengthening pieces, without the difficulty of cutting folding and drilling. For tinplate to have any strength it must be reasonably thick and you will require heavy duty tools etc to shape it.

Another handy medium is wood, however this should be restricted to base panels and platforms, where a number of items need to be screwed into position. You should only use soft wood, as it will be lighter and easier to drill and screw into. Don't use nails for fixing or joining as they tend to work lose.

Lastly, don't be frightened to use parts you already have on hand, especially from the kitchen and laundry where you will find plastic bottles, lids and boxes ideally suited for turning into pulleys and wheels. Use all your imagination and initiative - you will need it as you are basically breaking new ground!

In robotics, lots of new terms need to be understood to make the project function properly. But the best way is the hard way. By trial and error. Terms like gear ratios, torque, drive speeds, strength of beams, can involve an enormous amount of mathematics. That's why it's best to look through articles and see how it has been done by others.

At the time of writing, only a very limited range of motors and gearboxes are available at the low end of the market and the best of these we found at Dick Smith Electronics.

The gearboxes are in kit form and require a small amount of assembly to fit the gears onto the shafts to produce a gearbox known as a compound gearbox.

A gearbox reduces the rotational speed of a motor and at the same time increases the torque.

Torque is the twisting or turning force of a shaft and after 3 or 4 gear reductions, a shaft will have a considerable turning force.

This will be sufficient to turn wheels or move a robot arm or lift a weight. Sometimes it is necessary to convert rotation into straight-line motion and this can be done with a rack and pinion, winch and string, crank and arm or wheel and track.

Apart from the problems you will encounter adapting the mechanics into the available space, there will be problems interfacing the motor to the electronics.
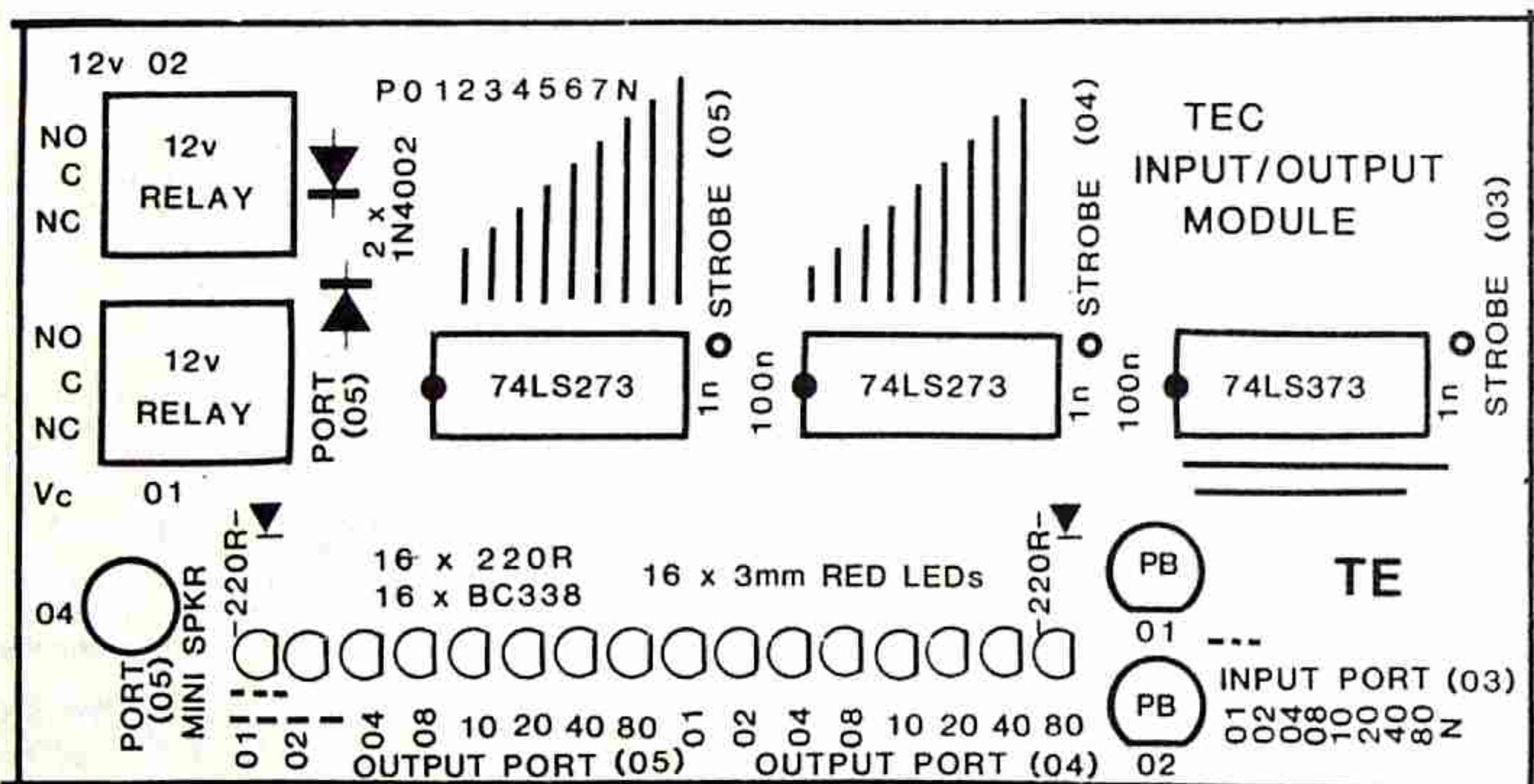
One of the major problems will be noise. Motors are inherently high noise producers and they must be kept far away from the electronics, both physically and electrically.

This may require a separate power supply so that noise and glitches from the motor do not get into the computer bus lines.

It will also be necessary to have high current available for the motor(s) as they draw a high current under load and if they stall, you must have sufficient current available to allow them to restart as soon as the load reduces.

A stalled motor can create a virtual short circuit and if connected to the computer 5v supply, the computer may drop out.

This has been avoided on the INPUT/OUTPUT MODULE by providing a separate supply line for the collectors of the output transistors and also the relays.

This will allow you to select your own supply voltage, with the necessary current capability.

When you are driving a motor, there will be three functions (or commands) needed. These are: ON/OFF (one command) FORWARD and REVERSE.

To achieve this, a number of lines (bits) will be required from the output port. Depending on the circuit used to drive the motor, either 2 or 3 bits will be required.

If you require the motor to operate in the forward direction as well as reverse, it will be necessary to use a relay. For a simple ON/OFF and FORWARD direction, a transistor can be used and only one bit (1 line) will be required. You can also get speed control from this line by including it in the program.

Basically speed control consists of outputting a high for a short duration and a low for a long duration and repeating the sequence about 100 times per second. To increase the speed, the duration of the high is increased and the low decreased. The only feature that remains constant is the repetition rate. It is essential to keep the pulses above 100Hz so that the motor rotates smoothly.

ASSEMBLY

By now you will be familiar with our assembly technique. Neatness is the overall aim. No matter how you build, the final result must be as neat as possible. This means the jumper links must be straight and sitting firm against the board, the LEDs must be close to the board and likewise the transitors, resistors and diodes. I thought it would be unnecessary to mention these points but we are still getting projects for repair in which the parts are mounted high above the board, the jumper links are twisted and kinked and the soldering is rough.

On the topic of soldering, It is important to use enough solder to cover the land and the hole. Again, we are seeing the smallest amount of solder on some joints, just enough to tack the lead to the land!

This is a very dangerous situation as you can create a problem that will be very difficult to locate. Sometimes the holes in the PC board cut through the track and the circuit relies on the solder to bridge the gap.

If you don't solder all around the lead, the copper track may contain a gap and obviously the project will fail to operate. Inspect the board before starting and check your workmanship after construction and you should have no problems in this area.

Begin assembly with the jumpers. Make sure they are straight and touching the board.

Next fit the resistors, followed by the LEDs transistors and two spike-

suppressing diodes. The overlay shows how these components are placed.

The 5 spike-suppressing capacitors are next and must be fitted close to the board. The IC's are mounted in sockets and the dot on the overlay indicates pin 1. You will find one end of the IC socket has a 'cut-away' portion to match with pin 1.

Fit the relays, mini speaker and switches. Then inspect the board to make sure all leads have been soldered properly.

After adding all the parts to the board, the 5 jumper lines are added and a female matrix connector soldered to each lead. These are covered with heatshrink to prevent shorting between leads when connecting to the TEC board.

MATRIX PINS

You will notice the module in the photographs has a set of matrix pins on the output ports and also the relays. These pins are not included in the kit however you can buy some and fit them as shown in the photo if you wish.

The 5 pins included in the kit are for adding to the TEC PC board to take the 5 flying leads from the input/output board.

Paul has included a 9 pin input plug and a 10 pin plug for connecting to the TEC. These are not included in the kit but can be easily made from 18 pin and 20 pin IC sockets. They are small and delicate but will last a number of insertions and removals.

TESTING

The first program in the list is the test program. It has a short routine to flash the output LEDs so that every second LED is lit and then the others are flashed. The program repeats this a number of times then changes to detect an input from the input port. The result is indicated on the corresponding output LED.

If this sequence is not observed, the program should be double-checked. Make sure it contains the correct commands. Then check the flying leads. They must be connected to the correct outputs on the decoder chip. Refer to the line diagram for the position of each lead.

TEST PROGRAM

LD B,10	0900	06 10
LD A,AA	0902	3E AA
OUT (04),A	0904	D3 04
OUT (05),A	0906	D3 05
CALL DELAY	0908	CD 50 09
LD A,55	090B	3E 55
OUT (04),A	090D	D3 04
OUT (05),A	090F	D3 05
CALL DELAY	0911	CD 50 09
DJNZ	0914	10 EC
LD A,00	0916	3E 00
OUT (04),A	0918	D3 04
OUT (05),A	091A	D3 05
IN A,(03)	091C	DB 03
CPL	091E	2F
OUT (04),A	091F	D3 04
JR	0921	18 F9

LD DE,0000	0950	11 00 00
DEC DE	0953	1B
LD A,D	0954	7A
OR E	0955	B3
JRNZ	0956	20 FB
RET	0958	C9

The second program is a 12-note organ using a solf-touch key pad for the input and the mini speaker on the IN/OUT module as the output.

The ide of an organ may have limited possibilities in itself, but the knowledge of how to produce a tone will be very beneficial.

In robotics, for instance, a mouse can be equipped with a speaker to produce a tone when it touches an obstacle etc. The note sounds for as long as the robot touches the object.

The importance of the program is to show how a tone is produced and how the pitch can be altered by adjusting the delay value.

Follow through the program and see how this is done:

ORGAN PROGRAM

XOR A	0900	AF
OUT (01),A	0901	D3 01
OUT (02),A	0903	D3 02
OUT (04),A	0905	D3 04
OUT (05),A	0907	D3 05
LD HL,09FF	0909	21 FF 09
IN A,(03)	090C	DB 03
CP FF	090E	FE FF
JR Z,090C	0910	28 FA
LD BC,03FF	0912	01 FF 03
DEC BC	0915	0B
LD A,B	0916	78
OR C	0917	B1
JR NZ,0915	0918	20 FB
IN A,(03)	091A	DB 03
INC HL	091C	23
INC HL	091D	23
CP (HL)	091E	BE
JR NZ,091C	091F	20 FB
INC HL	0921	23
LD B,(HL)	0922	46
DJNZ 0923	0923	10 FE
LD A,04	0925	3E 04
OUT (05),A	0927	D3 05
LD B,(HL)	0929	46
DJNZ 092A	092A	10 FE
XOR A	092C	AF
OUT (05),A	092D	D3 05
IN A,(03)	092F	DB 03
CP FF	0931	FE FF
JR NZ,0922	0933	20 ED
JR 0909	0935	18 D2

at 0A00:

00	64	3C
FA	BD	CF
84	5C	34
DE	F3	AF
7C	54	2C
BE	D7	FF
74	4C	
F9	B7	
6C	44	
DD	EB	

The third program controls the 16 output lines via a 12-key phone pad.

To turn on one of the left-hand outputs (port 05), press the asterisk key then a number button from 1-8. The right-hand port (port 04), is accessed by pressing the 'hatch' key then a number from 1-8.

When a second number key is pressed, the corresponding output-line changes state. Thus a high output will go low and vice versa. To access the other latch, one of the control keys (asterisk or hatch) must be pressed.

The program is fully described beside each instruction and this will assist you to design your own programs.

An important point to remember is DEBOUNCE. The soft-touch keys require a time to settle down before a value can be read. This means a short delay must be included in the program (see address **0913** and **0914**).

The reason is the contacts in the pad are made from a carbon compound and they create a considerable amount of bounce when a key is pressed.

Since the computer is a high-speed piece of equipment, it will pick up an incorrect value if the three contacts in the switch are not closed when it is being read.

To overcome this a short delay is introduced between the time when a key is pressed and when it is read.

The program can be modified to suit your own requirements. For example: a random output can be turned ON, or more than one output can be turned ON at the same time. A delay could be introduced to turn OFF and output after a set period of time or you could create a visual effect on a set of LEDs.

It's up to you. Study the program and try making some modifications.

For a very simple test program, try this:

3E FF
D3 04
C7

Eight LEDs will illuminate to show the program and board is working.

Wiring diagram showing the connection of the phone pad to the input/output module, and the module to the DIP header plug. Note: line '80' is not used when connecting the phone pad.

Photo, left: Motor and gearbox with two 100/16v electrolytics placed back-to-back to create a non-polar capacitor to reduce spikes from the motor. (i.e: the positive lead of each electro connects to a motor lead and the join of the negative leads is left 'floating').

Photo, right: The key pad connected to the input/output module via ribbon cable and to the TEC via hook-up flex.

KEY PAD CONTROLS OUTPUT LINES

Instruction	Address	Code	Description
XOR A	0900	AF	Data for port 05 is stored at 0A05 and 0A04 for port 04. These two locations are initially cleared in the first 6 lines of the program. Later, you will see why we have chosen registers B and C for this operation.
LD B,0B	0901	06 0B	
LD C,04	0903	0E 04	
LD (BC),A	0905	02	
INC C	0906	0C	
LD (BC),A	0907	02	
LD HL,09FF	0908	21 00 0A	HL is the pointer for the byte table.
LD D,00	090B	16 00	D is the count register for the key.
IN A,(03)	090D	DB 03	The program inputs via port 03, looking for a key press. Any value other than FF will exit from the loop.
CP FF	090F	FE FF	
JR Z,090D	0911	28 FA	A short delay is created via the D register to give the pressure sensitive keypad switches a short period of time to settle to a value that can be read correctly.
DEC D	0913	15	
JR NZ,0913	0914	20 FD	
IN A,(03)	0916	DB 03	Input this key value via port 03 to the accumulator.
INC D	0918	14	The next 4 lines generate a value for D that will be the same as the key. This is done via a loop and incrementing D until the key compares with the byte in the table will make D equal the key value.
INC HL	0919	23	
CP (HL)	091A	BE	
JR NZ,0918	091B	20 FB	
CP EB	091D	FE EB	The next 8 lines look for the STAR key or HATCH key and if either is pressed, C is loaded with either 05 or 04. This will allow the program to output to the correct port via the instruction **OUT (C),A** Also locations 0A05 and 0A04 use the C register for storage. In this way the C register serves a dual role and some of the powerful instructions such as OUT (C),A can be employed.
JR NZ,0925	091F	20 04	
LD C,05	0921	0E 05	
JR 0945	0923	18 20	
CP AF	0925	FE AF	
JR NZ,092D	0927	20 04	
LD C,04	0929	0E 04	
JR 0945	092B	18 18	
LD A,(BC)	092D	0A	Load A with the byte at location 0A05 or 0A04.
LD E,D	092E	5A	Store the key value for later use.
RRCA	092F	0F	The next 3 lines rotate the accumulator so that the wanted bit is rotated to the end of the register and thus only one TEST will be required.
DEC D	0930	15	
JR NZ,092F	0931	20 FC	
BIT 7,A	0933	CB 7F	Look at the highest bit and jump if it is zero. Otherwise execute next instruction.
JR Z,093B	0935	28 04	
RES 7,A	0937	CB BF	At this line the bit will be '1' and thus the program resets it to '0' and a jump is performed.
JR 093D	0939	18 02	
SET 7,A	093B	CB FF	The highest bit is SET via this instruction.
LD D,E	093D	53	Load D with the key value in readyness for rotating the accumulator back to it previous position. RLCA is a single byte instruction that rotates the accum and sets the carry flag. The bits don't enter the CARRY.
RLCA	093E	07	
DEC D	093F	15	
JR NZ,093E	0940	20 FC	
LD (BC),A	0942	02	Store the resulting byte in memory.
OUT (C),A	0943	ED 79	Output the byte to either port 05 or 04
IN A,(03)	0945	DB 03	Look at the input port and loop the next 3 instructions until the key has been released. This is a debounce routine, essential to produce a clean key action.
CP FF	0947	FE FF	
JR NZ,0945	0949	20 FA	
JP 0908	094B	C3 08 09	Jump to the start of the main part of the program.
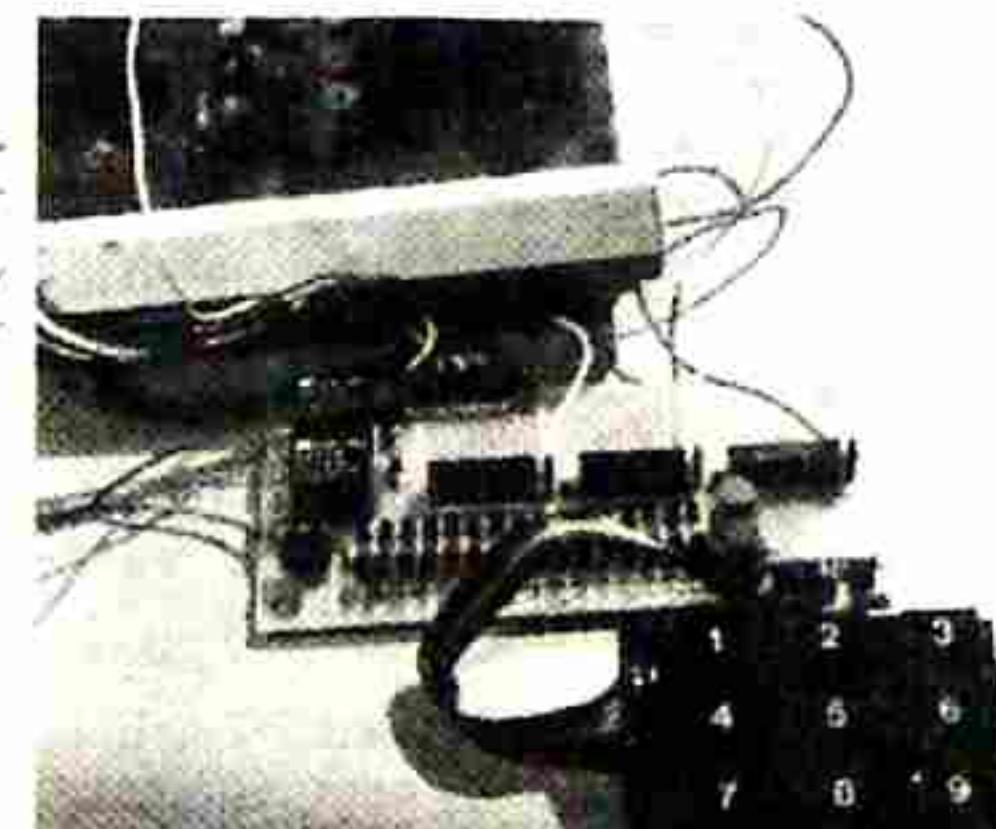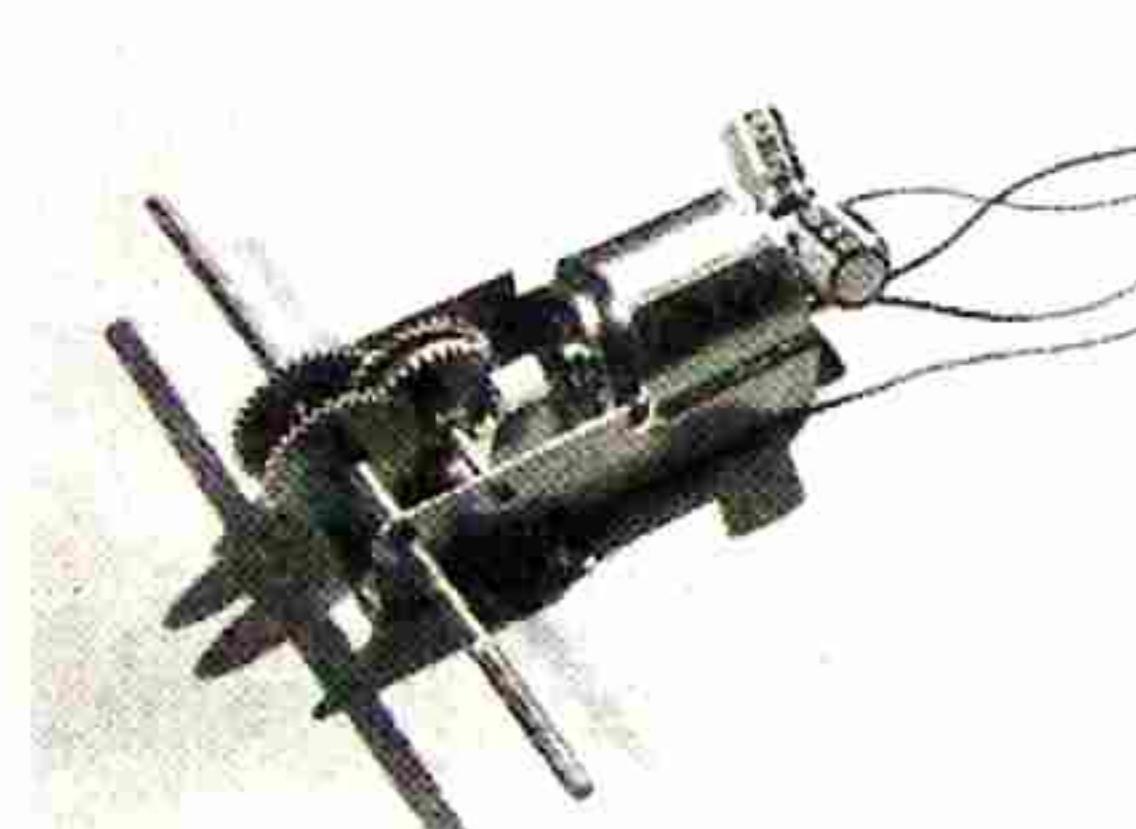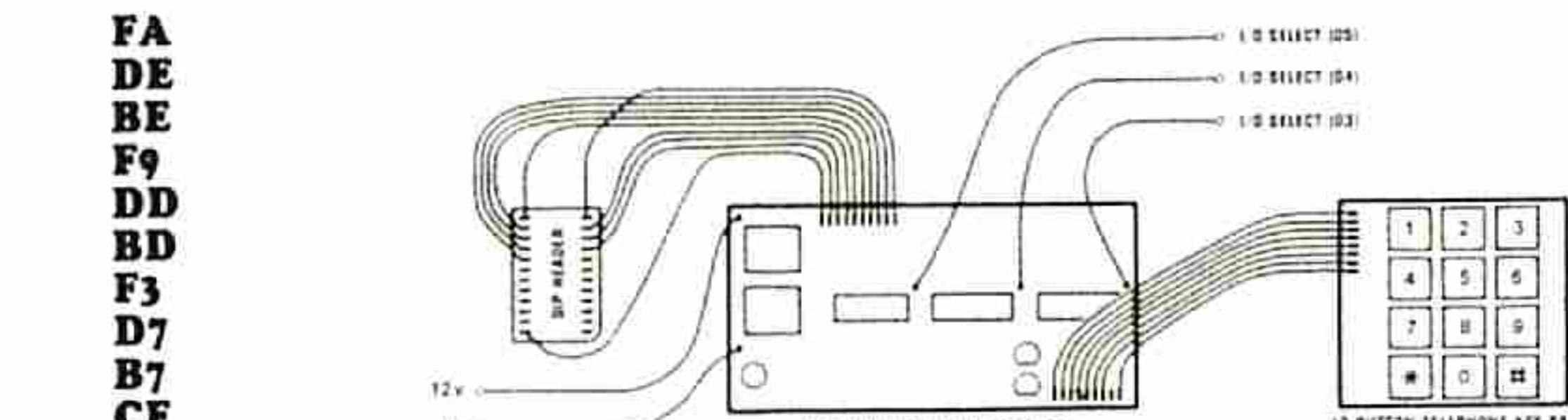
At 0A00:

FA
DE
BE
F9
DD
BD
F3
D7
B7
CF
EB
AF

GUITAR PRACTICE AMPLIFIER

AN 8-WATT AMPLIFIER WITH PRE-AMP AND FUZZ

Kit of parts: $12.30
PC Board: $3.40
Complete: $15.70
Kit of extras: $26.40

GUITAR PRACTICE AMPLIFIER CIRCUIT

POWER SUPPLY (INBUILT)

PARTS LIST

1 - 1R ¼watt		
1 - 2R2	1 - 10k	
1 - 39R	1 - 15k	
1 - 220R	1 - 56k	
1 - 470R	1 - 68k	
2 - 1k	1 - 680k	
1 - 6k8	1 - 4M7	

1 - 10k mini trim pot
1 - 50k mini trim pot

1 - 1n greencap
1 - 39n
4 - 100n

1 - 4u7   25v PC mount electro
1 - 100uF   25v PC mount
1 - 470uF   25v PC mount
3 - 1000uF 25v PC mount

2 - BC 547 transistors
4 - 1N 4002 diodes
1 - 5mm red LED
1 - LM 383 (TDA 2002) IC

1 - DPDT slide switch
1 - Mini U heatsink
1 - 6BA nut and bolt

1 - GUITAR AMP PC BOARD

Following the success of the 8-watt amplifier in issue 9, Paul has designed a project around the 8-watt chip and the result will have an even wider appeal.

It's an amplifier with a built-in pre-amp section, a fuzz section and a power supply which makes the unit completely self-contained.

As an amplifier it can be used as a mini amp for halls or outdoor events while the fuzz effect makes it suitable for guitar practice etc.

The circuit is quite simple as the use of a single chip eliminates the need for the usual array of components. It also makes the project extremely reliable as most of the work associated with biasing and matching of output devices has already been done.

It just takes a few external components to get the chip operating and a few more to create the pre-amplifier/fuzz section. The power supply is merely a set of diodes and a filter electrolytic, this being found adequate to eliminate any hum.

In the past, we have been hesitant to include a transformer in any of the projects as it requires working on components that will carry the 240v supply.

We have now overcome much of the previous apprehension by laying out the wiring in such a way that the live terminations can be completely covered with heatshrink, thus totally eliminating possible contact with the mains voltage.

This means the assembly must be carried out strictly according to instructions and the wiring checked by an authorised supervisor before connecting the project to the mains.

Even though the mains section involves only about 4 or 5 connections, the possibility of a faulty or misplaced lead is always there, and thus someone else must check you work before plugging it into the outlet.

Once again, the LM 383 amplifier chip has been used as it is virtually impossible to damage with overload.

In our circuit, we are supplying a safe operating voltage and include an electrolytic in the output, thus offering no cause for complaint. The chip is designed to deliver its full output wattage into 1.6 ohms and even if the output leads are shorted together, the internal temperature limiting circuit will prevent the chip from getting too hot.

The advantage of delivering into a low impedance such as this means the amplifier can be connected to up to five 8ohm speakers in parallel and each speaker will be supplied with about 2 watts.

It works like this: If you connect one 8ohm speaker to the output, the wattage from the chip will be about 2 to 3 watts. If another 8ohm speaker is added to the output, the power delivered by the chip will be about 5 watts. This is 2.5 watts per speaker.
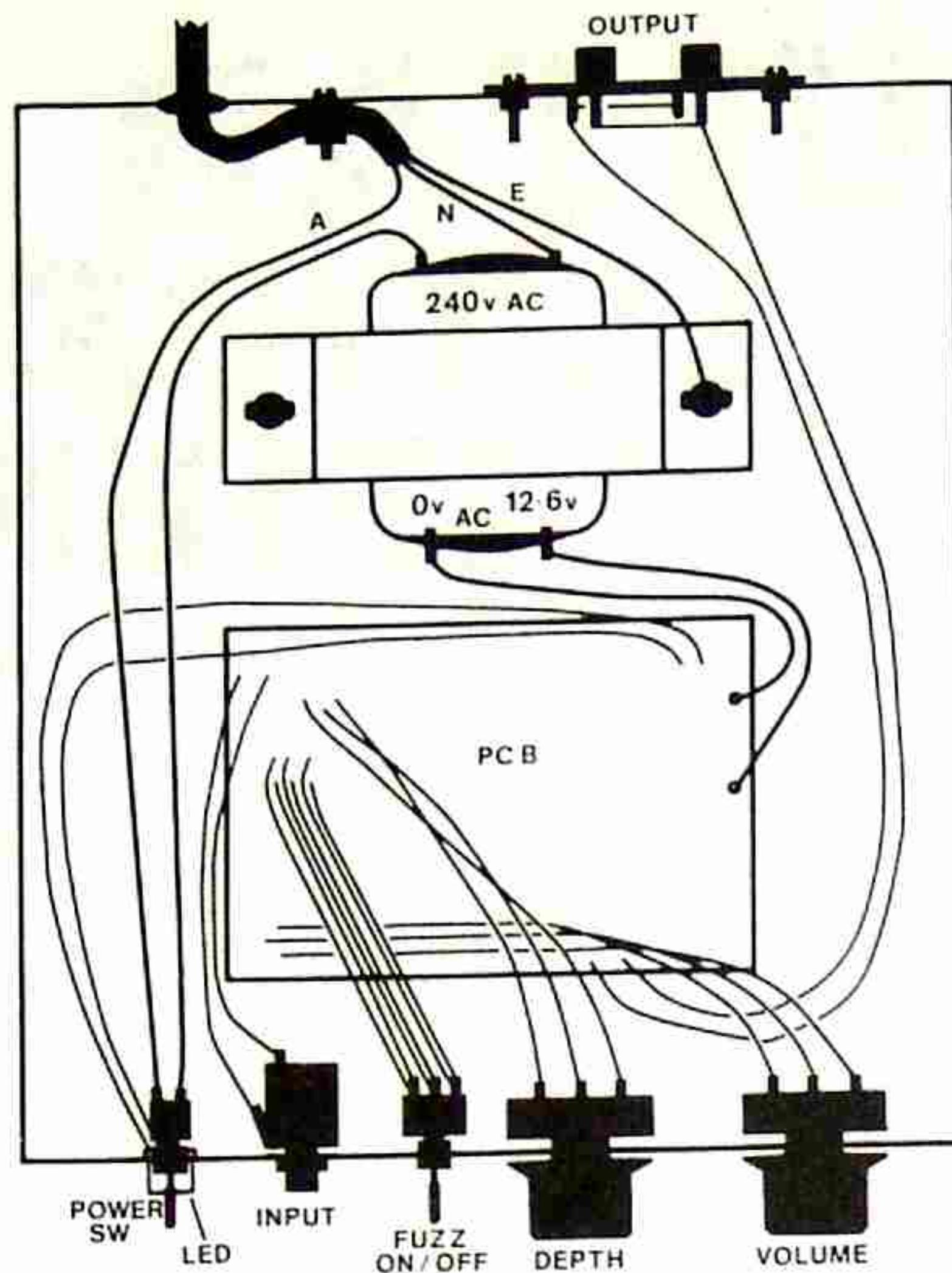
When 3 speakers are connected, the output power rises to 6 watts so that when 4 speakers are connected the power becomes about 8 watts. The data on the chip indicates the output power is about 10 watts when the load is 1.6 ohms but these figures are absolute maximum and would only occur on peak signals.

The main point to realise is the fact that the power delivered to each speaker does not alter appreciably as the number of speakers is increased.

This is handy if you wish to use the amplifier in a hall, or large room, where you may need full power.

The terms: POWER, MUSIC POWER, DECIBELS, and RMS OUTPUT are all very difficult to grasp and far too much discussion has been centred around trying to explain what they mean.

We are not intending to make an amplifier that will damage your eardrums or fill a

concert hall and so the loudness of the amplifier is of no real concern.

The reason why all these terms are hard to grasp is because the energy required to increase the loudness of a sound is not linear but a logarithmic scale and as such we can make the amazing comment that to double the loudness from the speakers would require an amplifier of about 50 watts. See how irrevelant figures are!

The only way to determine the suitability of an amplifier is to build it and use it.

Providing you don't expect too much from 8 watts, you will find this project to be adequate for all types of situations.

In fact we think it is loud enough to call it a Guitar Practice Amplifier. And you can
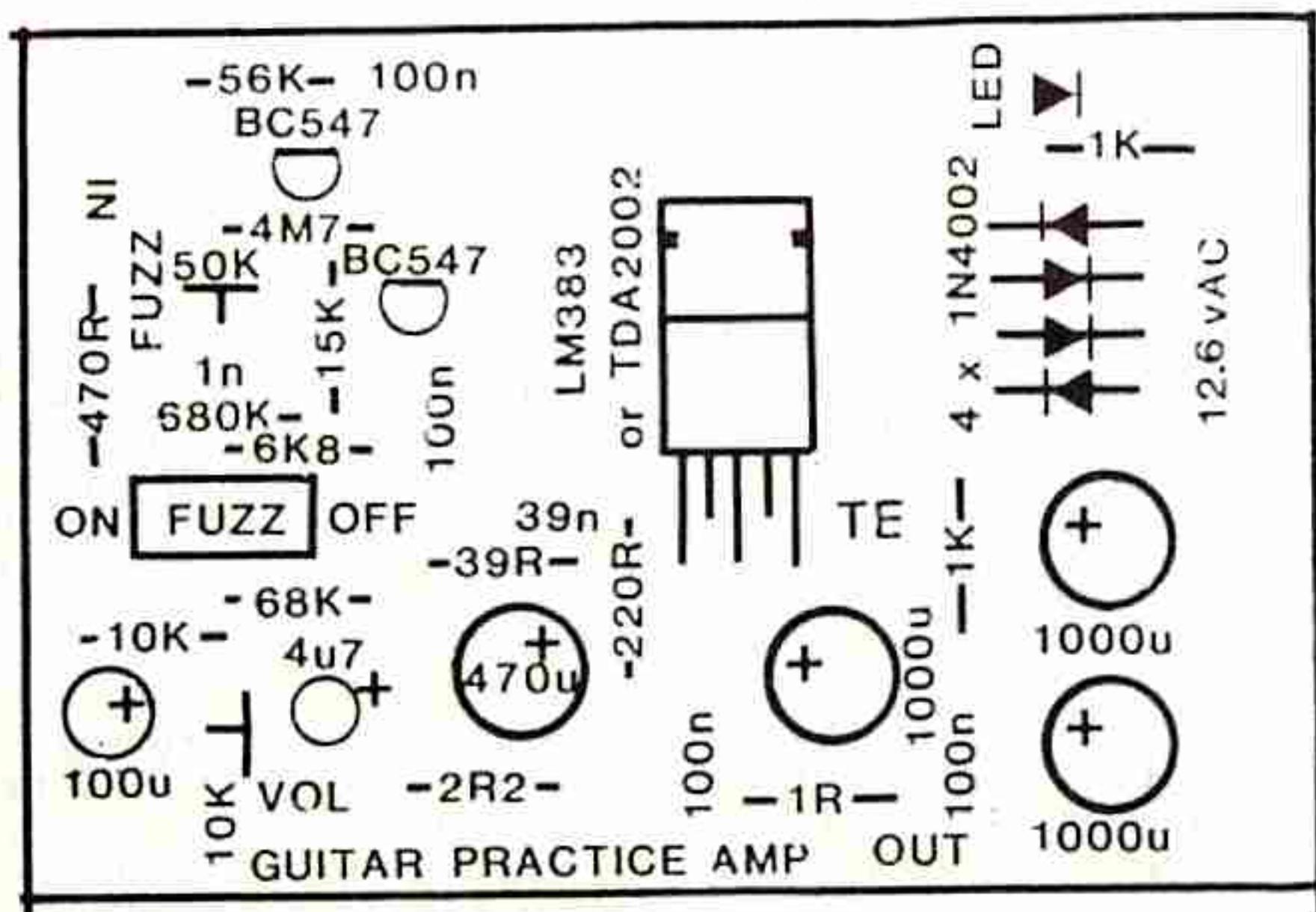
be sure neighbours from two or three doors down will hear you practicing.

Most of the effectiveness of an amplifier is in the speakers and there is one enormous misunderstanding concerning them.

Most people think large speakers require more power to drive them and as such you must use speakers capable of handling say 2 watts. for this project. But this is not so.

The wattage rating on a speaker refers to its maximum power handling capability and you could quite comfortably use four 15 watt speakers for this project.

The reason why large speakers are to be preferred in any amplifier system is due to the large amount of cone area they offer.

GUITAR PRACTICE AMP

As the cone of the speaker moves in and out, it shifts an equally large amount of air. It is this air which forms the sound waves that are picked up by the ear.

Obviously there are certain limits to this analogy but large speakers will certainly produce more room-filling sound than small ones.

To this extent it is not a bad choice to use a speaker capable of handling 15 watts, for a 2-watt amplifier.

HOW THE CIRCUIT WORKS

This project is based on a single amplifier chip LM 383 and as the schematic diagram shows, it can be thought of as an amplifier block with an inverting and a non-inverting input.

Provided the circuit board is correctly designed, the chip is highly reliable and produces a very clean output when supplied by smoothed DC in the range 12v to 18v. (Note: the SGS type MUST NOT be supplied with more than 18v as it has automatic switch-off at 18v!!).

A portion of the signal is taken from the output and fed into the inverting input to provide negative feedback. This reduces the output power slightly but more important it reduces the distortion.

Although we may want a considerable amount of distortion when in the fuzz mode, we require the amplifier section to be relatively distortion free so that it can be used as a PA amplifier etc.

The pre-amplifier/fuzz section is a clever design. It consists of two transistors and by using a switch, the bias on the two can be altered so that they change from amplifying mode to distortion mode.

This distortion mode is called fuzz and as they try to amplify, the high value of collector resistor causes the supply voltage to fall and thus the signal becomes distorted.

In addition, an AC feedback path is created via the 100n, 68k, 10k, 1n and 50k pot and this further upsets the DC bias so that the characteristic fuzz can be adjusted in depth.

When the fuzz switch is turned off, the first transistor provides a small amount of amplification to increase the input wavefom about 2 times.

The power supply is a simple rectified and smoothed supply, suitable for the LM 383 chip, with an indicator LED on the supply line to show when the unit is on.

CONSTRUCTION
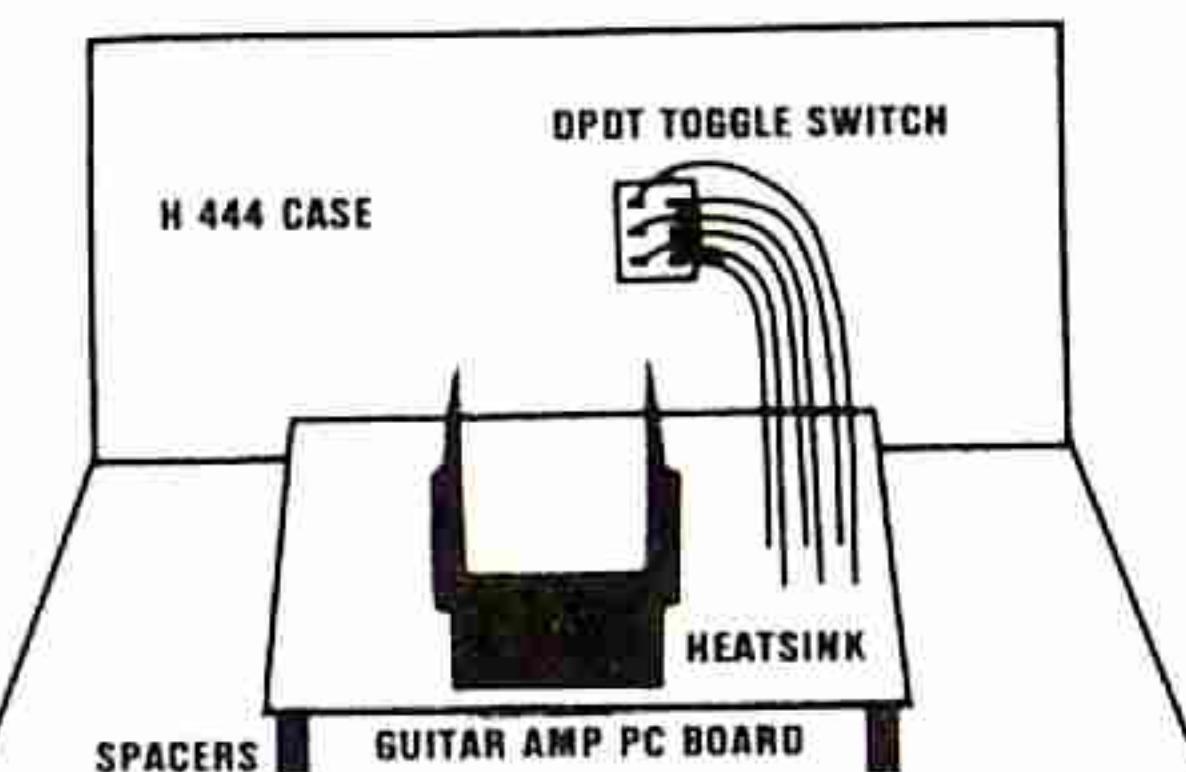
It is essential that this project be built on a PC board as the characteristics of the LM 383 are such that it requires a heavy earth plane to be present.

Without a tight earth plane and the hum reducing kink in the PC track, there is a tendency for the amplifier to generate its own background hum.

Only after designing 3 different PC layouts did we reduce the background hum to near zero. That's why we suggest construction on our PC board pattern.

This still allows you to make your own board from the artwork on the back page, but don't forget, our boards come with an overlay and once you have worked on boards with an overlay, you will never go back to a plain board again.

All the parts fit on the board and the controls are attached via plated leads. The twisting or plating of the leads serves to keep them neat and reduce interference between various parts of the circuit.

FUZZ SWITCH WIRING

Only the input lead needs to be screened to reduce hum pick-up and providing the others are not too long or placed near the power transformer, no induced hum can be detected.

The parts list at the beginning of the article includes all those items required for the PC board. The 'extra' parts list below will complete the project as shown in the photos.

Start by assembling the components on the PC board. Solder the resistors in place first and then the 4 power diodes. The diodes are placed alternately one way then the other with the bar or line on the overlay representing the band on the end of the diode.

GUITAR AMP EXTRAS

Kit: $26.40

1 -	10k lin pot
1 -	50k lin pot
2 -	knobs to suit
1 -	6.4mm mono socket
1 -	2way RCA socket
1 -	SPDT toggle switch
1 -	DPDT toggle switch
10cm heatshrink tubing 2.6mm diam
20 lengths of hook-up wire 15cm long
2 lengths of heavy duty hook-up wire
20cm shielded microphone cable
1 -	2155 transformer
1 -	mains cord and plug
1 -	H 444 case
1 -	rubber grommet
2 -	4BA nuts and bolts
3 -	6BA nuts and bolts
4 -	1/8" whitworth nuts & 25mm bolts
4 -	9mm spacers
1 -	earth tag
1 -	cable clamp

The 1 ohm and 2.2 ohm resistors will be new to many readers and they can be identified as follows: 1R = brown, black, gold, gold. 2R2 = red, red, gold, gold.

Next mount the amplifier chip and fit it onto the mini U heatsink with a 6BA nut and bolt. Push the 5 leads through the holes in the board and screw it into position. Make sure the leads are slightly bent as the heat from the chip will gradually expand and contract the leads and we do not want them to push on the solder lands. Bend the leads slightly and solder them in place.

Next fit the greencaps, making sure each one is identified correctly and solder them in place.

Finally the electrolytics are added to the board, making sure the negative lead as identified on the body of the electro goes down the negative hole. You will notice the positive is identified on the board and make sure this doesn't cause confusion.

This completes the PC assembly and is normally where we finish a project. But this one requires a number of external controls and so we have made an 'extras' kit containing the pots, switches, transformer, case, plugs and sockets so that you can complete the project.

The first thing to do is mount the controls on the front panel in the same relative places as in the photograph.

The switches are mounted using two nuts one on the front and one on the back of the panel, so that the switch does not protrude too far.

The same applies to the pots so that the knobs fit nearly flush with the panel. Use the line diagram on the previous page to assist you with wiring between the controls and the PC board. Each of the connecting wires should be about 15cm long to allow the board to be turned over for soldering. If the connecting wires are too short, you will experience great difficulty in repairing the board, should a fault occur.

Plat the three wires from each pot to keep them together and use different colours for each line so that they can be easily recognised at the other end.

When all the leads have been added, fit the board to the base of the case with 4 long bolts and use spacers to keep the board above the base.

Mount the transformer so that it is near the back of the case to prevent hum pick-up and use the 0v and 12.6v tags for the project.

The output sockets are a pair of RCA sockets and are mounted on the rear of the case with 2 6BA buts and bolts. The idea of this is to allow two 8 ohm speakers to be connected to the amplifier and up to four 8 ohm speakers can be connected to realize the full 8 watts output.

The last item to add is the power lead. Use a grommet in the rear of the case and also fit a cable clamp inside to prevent the cord twisting around inside the case and straining on the lugs of the transformer.

Take the active lead directly to the switch on the front of the case and another lead from the switch to the AC input on the transformer.

The neutral lead is taken directly to the other AC tag on the transformer and the earth lead is connected to one of the legs of the transformer to make contact with the case.

It is wise to use heatshrink tubing over each of the mains voltage connections to prevent accidental touching of an exposed joint.

Check over the completed project and make sure it looks like our version. When you are satisifed all has been done, its ready for testing.

TESTING

Plug an electric guitar or microphone into the input socket and connect speakers to the output.

Switch the unit on and try the amplifier with the fuzz section off. You should hear a clean response, even with the volume control turned fully up. The background hum and noise should be barely audible when an input jack is inserted and will drop considerably when removed.

Switch on the fuzz and adjust the depth control. You will be most impressed with the effect. It's now up to you to put it to good use.

FAULT FINDING

If the unit fails to operate, turn it off immediately and remove the PC board from the mounting screws. Check to see that no short-circuits exist and that all components are placed around the correct way. Also check the underside of the board for solder bridges or connections that may have been missed.
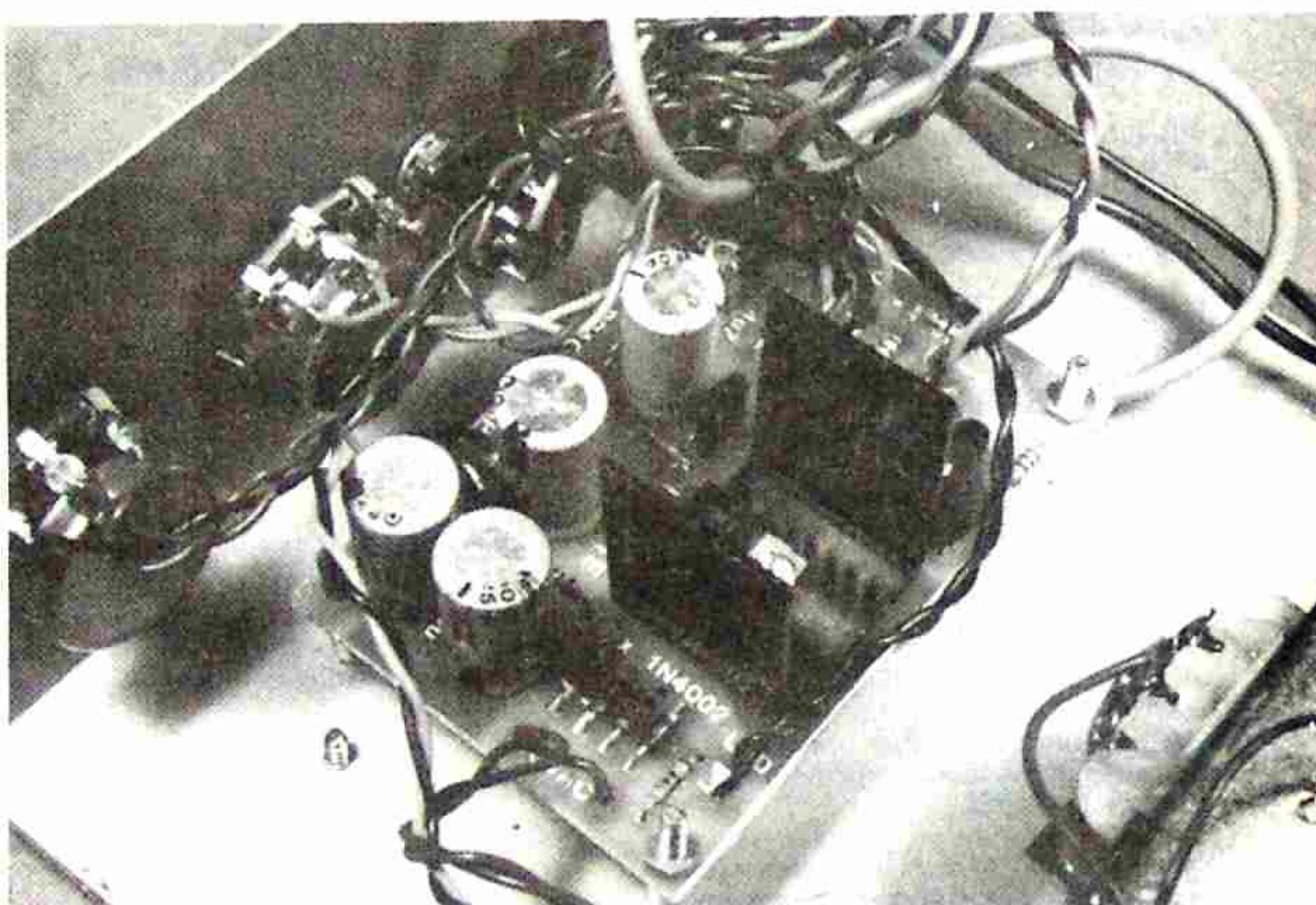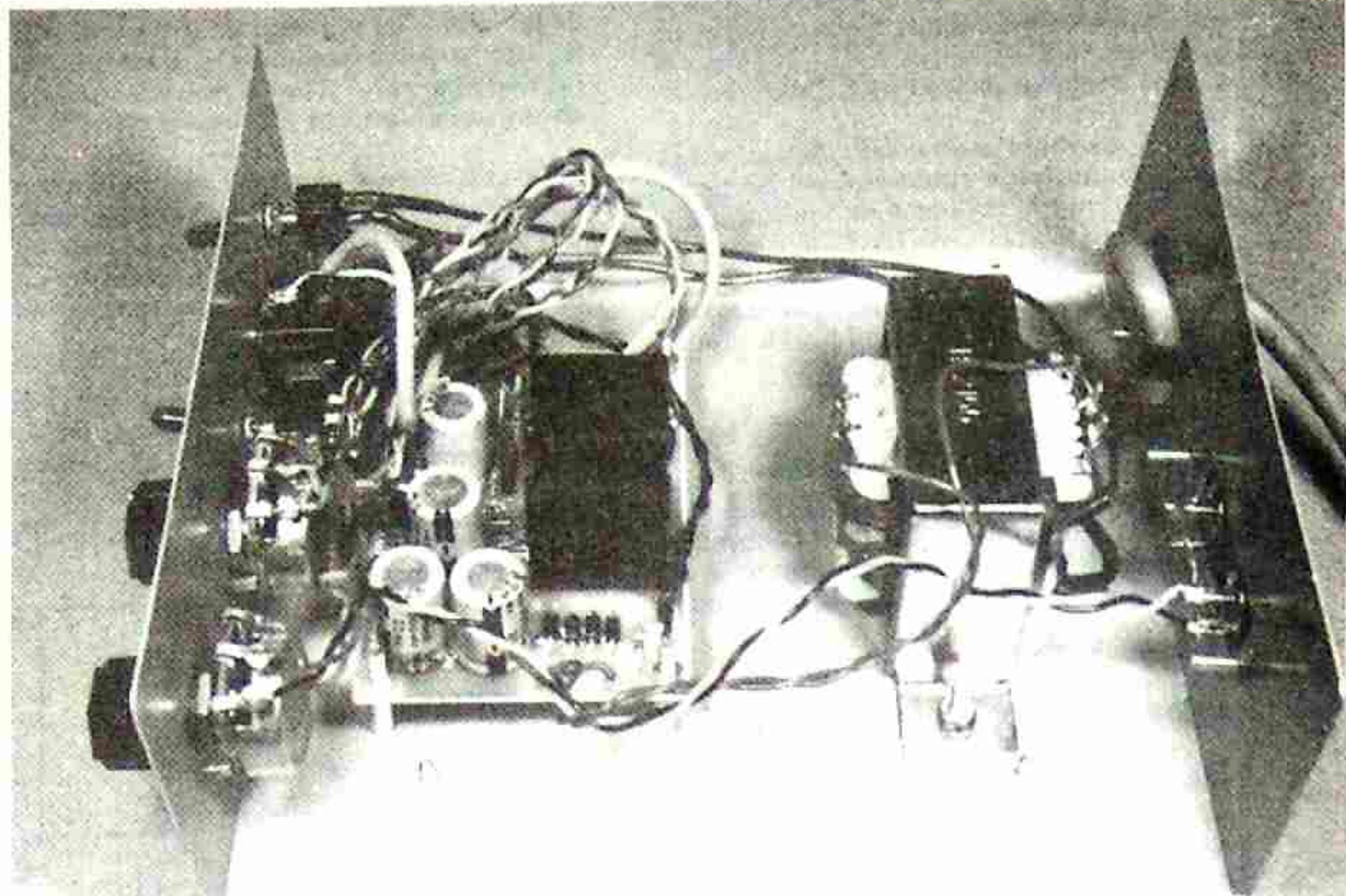
There is very little chance of a fault developing in the electronics of this project as the amplifier chip is fully protected against overload.

The most likely place to look is on the fuzz switch and input socket. If you are not sure how to wire these, use a multimeter set to low ohms and trace through the circuit before soldering.

The only other cause of a short circuit will be due to fine strands from one piece of hook-up wire touching another.

I hope you don't encounter any problems with the project and have the same success as we did.

It's a handy amplifier for those who want to practice without disturbing the rest of the household and can be used for amplifying other devices such as speech chips for the TEC computer!!
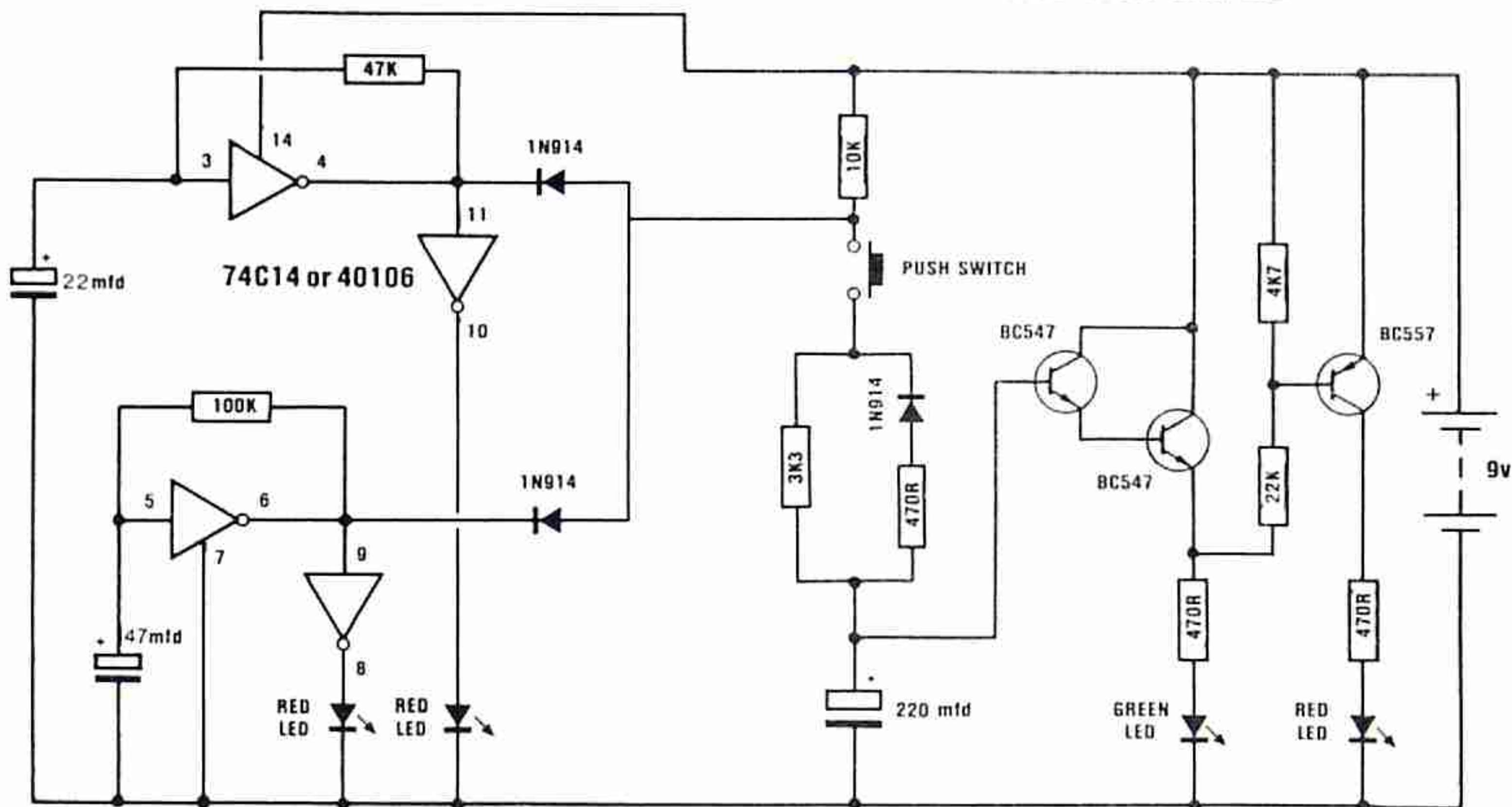
The two photos above show the wiring of the PC board to the controls, transformer and output plugs. Use these in conjunction with the layout diagram on P. 28 and also the fuzz switch diagram on P. 29 to produce a neat layout. Plat the leads to keep them together and choose multi coloured wires to make wiring easier.

The completed Guitar Practice Amplifier.

CO-ORDINATOR
A GAME OF SKILL AND PATIENCE

CO-ORDINATOR CIRCUIT

Co-ordinator is a game of skill. It tests your reaction time and patience.

Patience because a certain condition appears only every few seconds and it corresponds to both indicator LEDs being OFF.

When the two LEDs are off, you must press the button and this supplies a little energy to a reservoir electrolytic. The voltage on this electro is detected by a high impedance amplifier and the result is shown on a "LED-pair". As the voltage rises, one of the LEDs in this pair gradually dims and its brightness is transferred to a LED of opposite colour.

This process is very gradual and may take up to 20 or 30 pushes. But if your timing is out, all the achievement of these presses will be lost to the discharge circuit.

The time when it is safe to press and NOT press, is very abrupt and you have to be very careful not to extend into the "forbidden" zone.

The circuit consists of four sections or 'building blocks' and by describing each of these fully, we cover the complete circuit.

The first building block is the low-frequency oscillator, of which there are two. Each oscillator operates independently and its rate of flash is indicated on a LED. The LED is buffered by an inverter for one very important reason.

The voltage across a LED will not rise above about 1.7v (for a red LED) and if it were connected directly to the output of the oscillator, it would stop it from operating. This is because the output must rise to more than 66% of the rail voltage so that the input can detect a HIGH and change state, due to the function of the feedback resistor. The operation of this type of circuit has been covered in ELECTRONICS Stage-1, pages 77 and 78.

Colin playing with the Co-Ordinator. It's more difficult than you think to get the green LED to turn on fully.

The value of the timing components is not very critical and you will see one set is about half the value of the other. This gives a flash rate of about 4:1 and means the two output LEDs are both OFF for a very small fraction of the time.
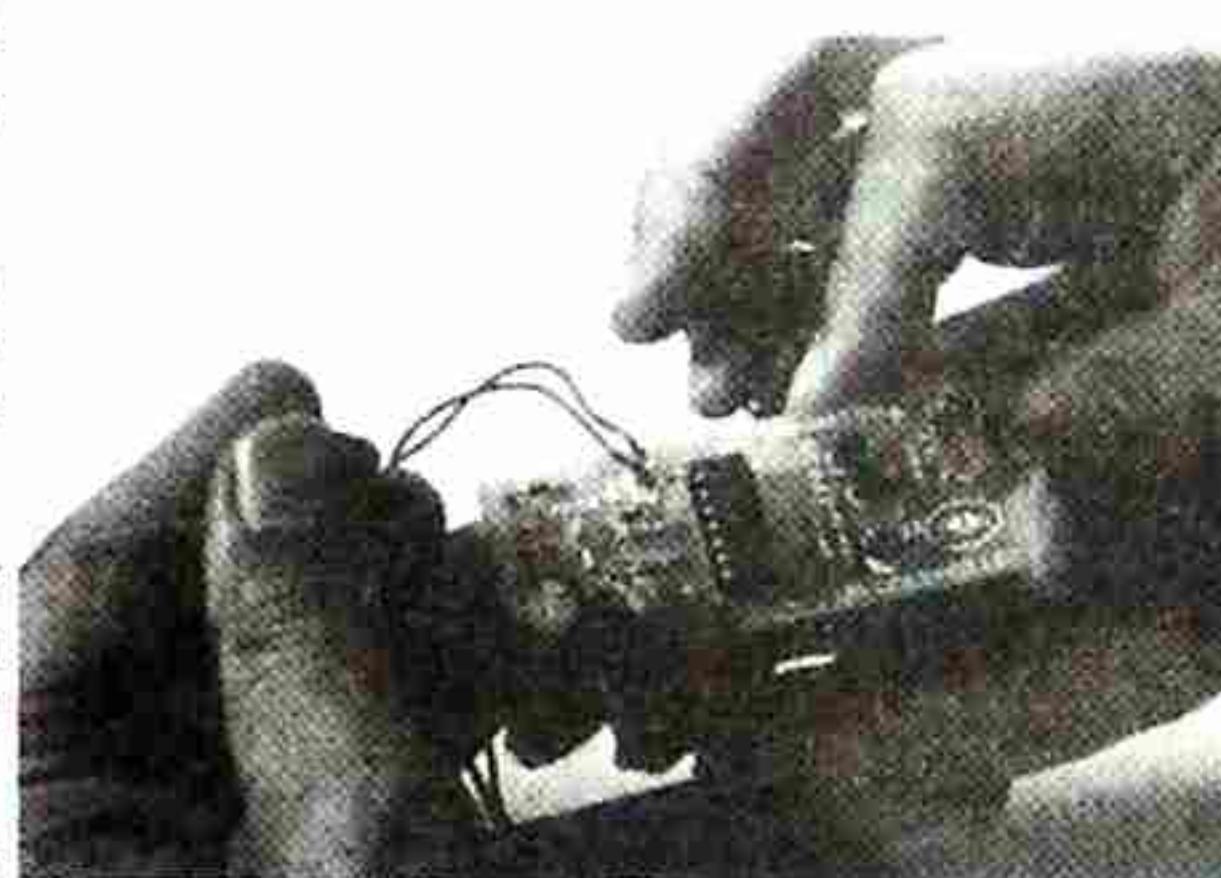
The next block is the GATE, made up of two signal diodes and 10k resistor. This forms an AND gate and operates as follows: The circuit requires a HIGH on the end of the 10k resistor to charge the reservoir electrolytic and this is available ONLY when the cathode end of both diodes is HIGH.

This condition corresponds to both LEDs being OFF and thus the circuit is designed around the OFF state.

When the output of both oscillators is HIGH, the charging path for the electrolytic is via the 10k and 3k3 resistors and during the short period of time when this occurs, the button should be pressed. When either oscillator goes LOW, it brings the discharge path into operation.

The discharge path is made up of a 470R resistor and signal diode. This path has a much lower impedance than the charge path (the 3k3 resistor) and thus most of the good work done during the charging will be removed very quickly by one false move.

During charging, the diode (in series with the 470R) forms a 'blocking' effect and does not play any part in the charging.

Next we come to the high impedance 'detector' circuit. It is actually an emitter follower arrangement, designed to turn ON one of the indicator LEDs.

The question we ask is "Why use two piggy-backed transistors?"

In simple terms, the answer is to keep the LED illuminated between presses of the switch. This block is a good example of how a transistor operates. It shows that a current is required by the base, for it to supply collector-emitter current.

If we remove the top transistor, and connect the base of the lower to the electrolytic, the LED would gradually dim between pushes of the switch. This is because the base requires current for the transistor to operate the LED and it would draw on the electrolytic for this energy. The voltage on the electrolytic would gradually fall and the effect would be lost.

By placing another transistor as shown in the circuit, the current for the lower transistor is obtained from the positive power rail and thus the current drawn from the electrolytic is reduced by as much as 1/200 - 1/300th.

This circuit is called a SUPER ALPHA pair because the the alpha (or gain) of the transistors are multiplied together to produce an arrangement similar to a single transistor with enormous gain.

The output of this stage is also taken to another stage which is called a common-emitter arrangement (for a PNP transistor) in which the load is in the collector. This stage produces an effect which is opposite to the previous so that when one is OFF, the other is ON.

This is the entire circuit. It consists of one chip, 3 transistors, 4 indicating LEDs and a few passive components.
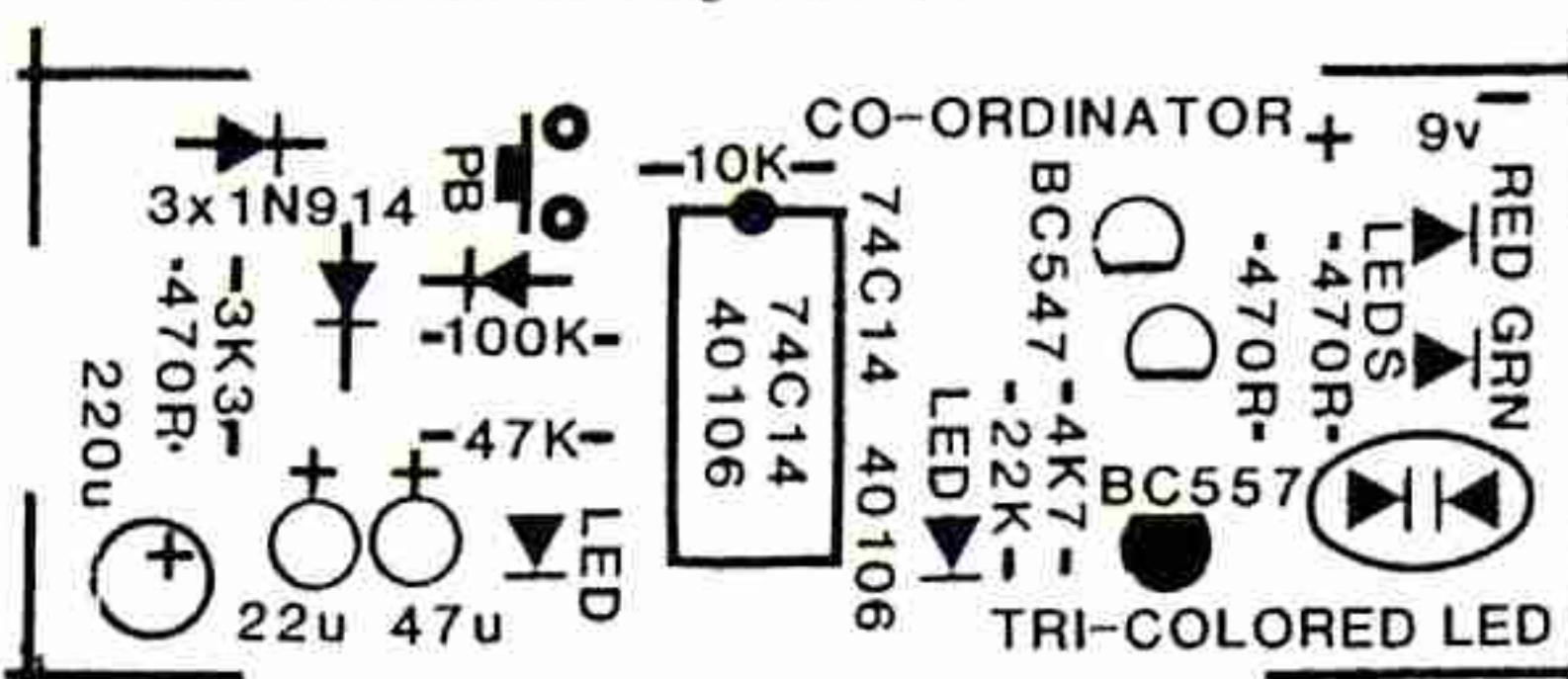
The circuit is supplied by a 9v battery and is designed to accept a tri-coloured LED or a red and green LED in the output.

Building the project will help you understand the circuit a lot more, especially if you come across a small difficulty which you solve yourself.

ASSEMBLY

Begin construction with the smallest components. They are easiest to fit when no other parts are on the board. Begin at one end and fit the 3 diodes and 9 resistors.

The overlay for Co-ordinator shows the positioning for all the components. You can use either a tri-coloured LED or a red LED and green LED. Note: the prototype in the photo below uses 5mm LEDs throughout, however the parts list suggests 3mm LEDs for the flashing indicators.

The 3 diodes must be fitted so that the cathode end (as indicated by a black band around one end) goes over the 'bar' shown on the overlay.

Next fit the 3 transistors. Two are NPN types and will be BC 546, BC 547 or BC 548. The other transistor is PNP and will be either BC 556, BC 557 or BC 558.
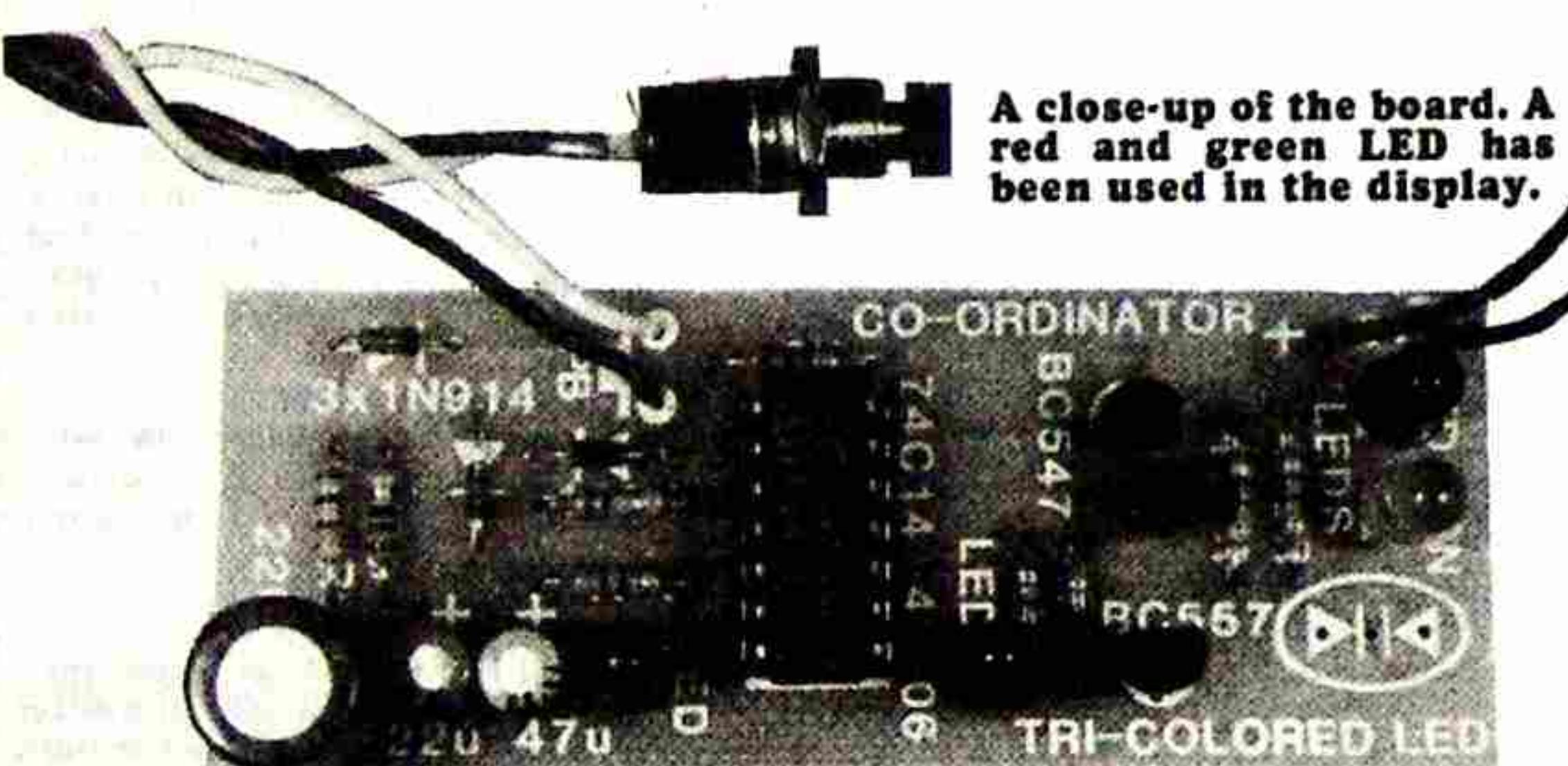
After this we fit the 4 LEDs. The two mini LEDs are 3mm types and the cathode lead must be inserted near the edge of the board. You have a choice of readout. It can be either a 5mm red and green LED or a single tri-coloured LED. Either of these arrangements can be fitted to the board. But NOT both.

Fit the IC socket so that pin 1 identification is towards the top of the board. This makes it easier to insert the IC correctly, after all the other components have been fitted. Solder the 3 electrolytics so that the positive leads go down the holes as marked on the board. The positive leads are generally the longer ones.

Mount the push switch on leads about 10cm long and solder them to the board at locations marked 'PB'.

Finally add the battery snap, push the IC into the socket and the project is ready for testing.

PARTS LIST

3 - 470R ¼watt
1 - 3k3
1 - 4k7
1 - 10k
1 - 22k
1 - 47k
1 - 100k

1 - 22uF 16v electro
1 - 47uF 16v electro
1 - 220uF 16v electro

3 - 1N 4148 diode
2 - BC 547 transistors
1 - BC 557 transistor
2 - 3mm red LEDs
1 - 5mm red LED
1 - 5mm green LED

1 - 74C14 or 40106 IC

1 - Push button
1 - Battery snap

1 - 14 pin IC socket

20cm Hook-up flex

1 - CO-ORDINATOR PC

TESTING

When the power is applied, the two 3mm LEDs should illuminate with random flashing and the red output LED should come on.

The LEDs are controlled by three separate sections of the circuit and if any do not work, you can go directly to the section responsible.

If the left-hand mini LED does not flash, the circuit components to check are: the 47k resistor and 47uF electroytic. If the right-hand mini LED does not flash, the 22k and 22uF electrolytic should be checked. The fault could also lie in the chip.

If the 5mm red LED does not come on, the fault will lie in the BC 557 and/or biasing components, comprising the 4k7, 22k, 470R and the green LED.

Try all your friends with the co-ordinator. Those who play video games will have a very good sense of judgement and will get the LEDs to change illumination without much difficulty.

There is a secret way of getting the circuit to charge the electrolytic without the need for any skill. We call it the test mode but you can call it the 'cheats' mode.

It consists of placing a diode across the switch so that the cathode faces the 3k3 resistor.

In this mode, the electrolytic will charge every time the outputs are HIGH (via the 10k and 3k3 resistors) and will not discharge when the outputs are LOW. And you don't have to push the switch!

I hope you enjoy this project and take advantage of the diode 'trick' to impress your friends!

A close-up of the board. A red and green LED has been used in the display.

SHOP TALK

No matter how long it takes to get an issue out, you can't say it's not worth waiting for. We have been waiting nearly 8 months for this issue. Let's hope the next is quicker.

As we get a little more streamlined and better organised, we have come to the decision that we can bring out a publication every three months.

At least that's the intention. A quarterly publication sounds acceptable and is within our capabilities.

As you can imagine, a magazine which doesn't have any advertising takes much longer to produce. It's very easy to slap together an issue which has 60% advertising and only 40% editorial.

And if the editorial is purely written copy, the task is even easier. Written copy is very easy to produce and although it may sound impressive, a good writer can churn out two or three pages in a day. It's when it comes to diagrams, photos and technical matter that the pace slows down.

Some of the pages, such as the programs for the Microcomp, represent 20 to 30 hours per page and it's nothing to see only 6 pages after a month's hard work. Maybe we aren't very fast but the daily requirements of running the business slows down the progress.

Because we have set our goals very high, we have suffered enormously. It has been our intention to keep the magazine relatively free of advertising and especially avoid breaking up the articles with meaningless ads. By this I mean the ones which tell you nothing about the product. Like lists of products and services, which never vary from one issue to the next. These are not my idea of excitement!

The other requirement is a regular publishing schedule. Up to now the issues have been released only when all the projects are tested and finalised.

Reader's don't want this. They want a monthly or bi-monthly journal or at least a known publishing date.

To this end we have decided upon a quarterly schedule and since we have a lot of projects ready for inclusion, we can see ourselves being very busy for many issues to come.

★★★★★★★

I think every person and every business gets hit with a shoddy purchase or workmanship at some time in their life and if we aired our grievances every time something went wrong, nothing would be bought.

But to know about someone elses bad dealings can be invaluable. It may save you falling into the same trap.

We hear about the extremes, from the man who sued an Australian match company for failing to suppy 50 matches in the match-boxes, to the scrap metal dealer who was duped out of thousands of dollars by a con man offering to sell the Eiffel Tower for scrap!

Thousands of people are being tricked every day: paying for goods and never receiving them.

The prospective swimming pool owner who paid $6,000 for a luxury pool and is still waiting for the hole to be dug, one year after paying the full price! Or the hundreds of Queensland farmers who paid for an electronic scarecrow which has yet to be delivered. And doesn't work anyway!!

With all this prior knowledge, we pride ourselves in knowing not to pay before delivery and thoroughly research a product before signing on the dotted line.

With this in mind, we offer you our latest lesson:

THE COPIER THAT DIDN'T COPY.

Every month we get deluged with advertising literature for photocopiers.

After doing without, for a number of years, we decided to invest in one.

One thing we have learnt in the past is the need to buy a well-known brand. Without the backing of a reputable firm you could be in serious trouble in a few years, if repairs were required.

The sales pitch we were given in the leaflets and from phone calls we made was the enormous reliability of modern photo-copiers and the low cost per copy. Also the copy quality rivalled off-set printing and thus for runs up to 100 or so, a photocopier was the answer.

Obviously one-offs were exclusively the relm of the copier.

After researching the field thoroughly we decided upon a C - - - - model 120. We cannot print the name but if you ring us, we will let you know. We asked every question in the book before signing on the dotted line - except one. The guarantee. Later we found out it was ONE MONTH!

When you consider how many photo-copies an average office makes in one

month, it represents about 1 to 2 hours of operation!

Imagine if you sold a TV and quoted a guarantee of one hour!!

The management's answer to this was, "The copier is only partly electronic and partly mechanical - you cannot compare the two."

You may not be able to compare the two but ONE MONTH is an indication of the faith they have in their product.

Within the guarantee period we were told the machine would be adjusted free of charge and so, within the 30 days, we phoned for a final adjustment.

This was duely done and had I not been present, we would have been charged $45.00 for the call. Only under insistence did the serviceman ring his superior and cancel the invoice.

Throughout the next weeks and months the copy quality gradually deterioriated until one day JET-BLACK copies emerged.

Fortunately the service centre is just down the road and I hopped into my car and called into the 'Serviceman's Club Rooms.'

At first they said point-blank that I needed a service call and would not help me any way at all. When I said I had adjusted some of the controls and made no improvement, they started rambling on with the most amazing load of rubbish about what could be the cause.

After about 10 minutes I could see I wasn't getting anywhere and decided to go back home and call head office.

Getting past the 'front desk' was a feat in itself and to get a feasible explanation to the fault was way beyond the scope of any of the personnel.

In desperation I took the front cover off the copier and was confronted with a series of pots - none of which were identified and it was only by remembering the controls touched by the previous serviceman, that we got any clue at all.

Even full adjustment of each and every control produced little better than an extremely dark copy.

Accepting defeat I rang for a service call.

Quite soon we had the pleasure of a smartly dressed servicememan who saw the name TALKING ELECTRONICS on the door and thought "What a load of twits - I'll fix this!

Faster than a rabbit down a hole, he had the cover glass off the machine and was merrily polishing the plate and lens system. Next he went to the corona wires and made them sing like a single string violin.

The excess toner container had a little in it and he promptly and eagerly dumped it onto a sheet of paper and wrapped the whole thing up like a 3 week old baby.

Then came the master's touch. He made a photocopy. Black as ink. Off came the cover. Fiddle, fiddle. Another copy. Black as ink. Out came the manual. Fiddle, fiddle. Another copy. Black again!

By now his quick, efficient antics had slowed and he made his first utterance: "I think the drum's gone." "Drums are on a pro-rata basis and a new one will cost xxx dollars.

Knowing I wouldn't be contributing a cent, I said "Go ahead." Half an hour later he had a new drum fitted and made a copy. Black as ever.

Now the chips were down. He wasn't as smart as he thought and started to talk to us.

I can't remember what he did next but about half an hour later he had changed the fluorescent tube which scans the page. Instant success. Copies came out perfectly.

It took about 30 minutes to replace the drum and adjust all the controls back to normal.

My point is this. Why did it take one hour and forty five minutes to locate the fault and then be charged nearly $100 for the job?

When you call the manufacturer you expect the get expert service. Not to have someone learning at your expense.

Repairs are one of the hidden costs when operating a piece of sophisticated equipment and especially when a firm has a monopoly on servicing.

There is absolutely nothing you can do. If we refused to pay the bill, the serviceman said we would be black banned.

The story doesn't end here. There's a little more.

At the time when the tube was replaced, we asked about some streaks down the page. The reply was the toner roller was worn and would cost about $120 to replace. He didn't look at the offending part and made no mention that it could be repaired.

At $120 we wern't interested and let the stripes remain.

Gradually these got worse and when it came to the time when an important photocopy was required, I decided to look into the fault.

To my amazement the toner roller assembly came away via a couple of screws and two clips.

Upon inspection I noted some of the powder had hardened and built up on the magnetic roller. A little solvent and some

hard rubbing removed all signs of the build-up and when it went together, the copies were perfect.

Keeping quiet as to the remedy, I phoned the manufacturer and asked about the fault. I could get absolutely no further than 'a service call was necessary'.

Again I ask. Why didn't I get any assistance from either the servicemen or the service department?

If our experience is any indication of the treatment being offered by multi-national companies, it's no wonder they have a team of servicemen in each district.

If you are in the market to buy a copier, remember us. By the time you buy a machine, fill it with paper, toner and cough up for a few service calls, you will be better off going to the local chemist or library and paying 10¢ per copy!

THE POST OFFICE . . .

Some people wonder why I am so critical of the Post Office.

They want me to be "all smiles and jovial" when taking the packets of mail and bags of publications to the counter.

But I have been around longer than most of them and seen an enormous decline in the service and an alarming increase in the ignorance towards the dissemination of the printed word.

I have seen postal rates increase from ½¢ per item to 30¢ per item in a period of time when wages have increased only about 400%. All along, the Post Office has maintained that the distribution of registered publications has been a LOSING proposition and with this they have increased the rates to an extent that they are choking the Australian publisher out of business.

Do you realize it costs more to post a magazine than to have it printed!!Quite a rediculous situation.

But there is nothing more absurd than the reasoning and intelligence of the personnel behind the decisions in the marketing and pricing sections of the Aust Post Monolopy.

For the privilege of getting reduced rates for registered publications, they charge a $36.00 application fee each year! But the final straw broke last week when a parcel of publications cost MORE than a normal postal article. In fact to post large quantities of magazines interstate costs exactly the same as normal rates.

So much for the understanding of the spread of information.

For a face-to-face discussion over some of the anomalies I had the pleasure of seeing the Field Manager for our local area. After a few attempts to explain the why's and wherefore's of how the prices were generated, he came up with a comment that absolutely floored me.

"I don't see why you publishers should get cheaper rates anyway," he said.

Can you imagine how I reacted! With a certain amount of cool I said, "Do you read a lot?" "No."
"Do you get any journals?"   "No."

If this type of reasoning prevails in our glorious Post Office administration, heaven help us in the future.   They have already tried to

squeeze the most out of the long-suffering customer, now they want to cripple him completely.

The Post Office has already lost many large contracts and they are in peril of losing many more.

They answer this with "some you lose, some you win," sort of attitude.

There are two further requirements of the Post Office for registered publications but they are so absurd you will only laugh.

I'll finish now, while I'm still smiling and leave you with some of the inside workings of our largest national monument.

Oh, by the way, I'M not allowed to sign the Editorial. They rang me 2 years ago and wasted half an hour of my time over this blatent disregard of the rules! You will notice I don't sign it any more. Take a look. See. I don't sign it any more. Aren't we obedient.

Last week we got a long letter to say the publication number must appear on the cover of the magazine. I don't want the cover filled up with junk inscriptions like this. But, as you know what bureaucracy can do, we have regretfully added the wording.

When you get this magazine, consider it a great privilege the Aust Post has condescended to handling it!

CAUTION for NZ READERS:

While on the topic of letting you know what and how we think, I must relay the dissatisfaction readers have had with one or more of the New Zealand suppliers as recommended in previous issues.

I have recieved nothing but complaints from our readers. The following excerpts from a letter is typical:

"I rote to the PC board supplier as suggested by you (NZ firm) for a TEC board and waited nearly 6 months! When it finally arrived the board had no overlay and no solder mask, even though it was the same price as if ordered from TE. It was really home-made!'

We have seen similar results from boards made in NSW. It had no overlay, no solder mask and no roll tinning! In fact it wasn't worth buying! If you compare ours to theirs, you will know what we mean.

Back to the NZ reader. After getting the board and sending for parts, it failed to work. So he rang us and asked if we could help. We said the TEC could be sent in for repair and it would be returned the same day. After picking himself up off the floor, he said he would do just that.

A few days later it arrived and was repaired. (I can't remember the fault). He rang to say he got it 3 days later and couldn't believe the service.

I don't like canning any firm but the NZ firms we recommended in previous issues have certainly not lived up to expectations.

I would be pleased to hear from them and get their side of the story.

PHONE DIALLER Part III
EXPERIMENTING FURTHER

Phone dialler part III took about one week of part-time effort for Colin to write (He's not very quick!) and has been tidied and closed up for publication.

However there are a number of improvements that can be made to the program (apart from the auto re-dial extension). For instance, the first byte in the number table is not used and can be deleted, the **CP 0A** instruction at **09C6** is not valid, and a few others.

The six middle locations are used by the SCAN routine for displaying data onto the screen. The 7 arrows under the locations show how the data is shifted from one location to the next via the SHIFT routine. Locations **0A80** to **0A86** are the ones cleared by the CLEAR routine to blank the display.

The diagram below shows how the DISPLAY BUFFER operates.

0A7F	0A80	0A81	0A82	0A83	0A84	0A85	0A86

Diagram showing the DISPLAY area, the SHIFT procedure and address for NEW DATA.

NEW DATA

These will be your challenge and at the same time see how you can simplify the program by using higher level commands. If you can't, don't worry. Programs in the next issue will be at a higher level and will use logic operations to create the same result with fewer instructions.

New data is inserted at **0A85** and this location is cleared via the SHIFT routine prior to a value being inserted (refer to SHIFT on P. 18). This prevents rubbish being shifted into the location from **0A86** as this would appear on the screen as brief flashes of junk.

VIC-20 MAGAZINE

A letter from Cris Groenhout, Editor VIC-20 Magazine.

Despite the commercial demise of the very popular Commodore VIC-20 Colour Computer, there is still a great number of enthusiastic users remaining with absolutely no intention of disposing of their equipment. So, to support this large group of users, we have recently decided to continue publishing the Association's magazine 'VIC'.

'VIC' is now in its third year of publication with 16 bi-monthly issues under its belt. The magazine sells to subscribers and retail customers for $2.00, a price which, compared to similar magazines, is very low. The magazine is also entirely dedicated to the VIC computer with no advertising and very little space tied up by news, letters, etc.

The association also distributes public domain software for a small copying fee and maintains a library of about 900 programs. There are also a number of other services and if you are interested in more information, write to Cris, at 25 Kerferd St., Watson, ACT, 2602. Tel: (062) 412 316, and enclose 2 stamps for return postage.

SCROLL

Instruction	Addr	Code	Comment
LD A,FF	09C0	3E FF	Load A with **FF** and transfer to the I register to detect when a key has been pressed.
LD I,A	09C2	ED 47	
LD A,I	09C4	ED 57	Look to see if a key has been pressed by comparing the accumulator with **0E**. Return if the accumulator is **0E**.
CP 0A	09C6	FE 0A	
NOP	09C8	00	
CP 0E	09C9	FE 0E	
RET Z	09CB	C8	
LD A,(HL)	09CC	7E	Load the value pointed to by HL into the accumulator.
LD D,20	09CD	16 20	Load D with a short delay value (for below.)
INC HL	09CF	23	Increment to the next location.
CP 10	09D0	FE 10	Look to see if end of table reached.
RET Z	09D2	C8	Return if end reached.
LD (0A85),A	09D3	32 85 0A	Load the byte of the table into the display buffer.
CALL SCAN	09D6	CD 80 09	Call SCAN for 32 loops (as determined by the D register).
DEC D	09D9	15	
JR NZ,09D6	09DA	20 FA	
CALL SHIFT	09DC	CD E1 09	Call SHIFT.
JR 09C4	09DF	18 E3	Jump to the start of the sub-routine.

SHIFT

Instruction	Addr	Code	Comment
LD B,07	09E1	06 07	Load B with 7.
LD IX,0A7F	09E3	DD 21 7F 0A	Load IX with location one lower than display buffer.
LD A,(IX + 01)	09E7	DD 7E 01	Load A with the value in the display buffer and transfer it to the next lower location.
LD (IX + 00),A	09EA	DD 77 00	
INC IX	09ED	DD 23	Increment the IX register.
DEC B	09EF	05	Dec B.
JR NZ,09E7	09F0	20 F5	and jump to above for 7 loops
RETURN	09F2	C9	Return.

at 0A00:

DISPLAY TABLE:

0 =	EB
1 =	28
2 =	CD
3 =	AD
4 =	2E
5 =	A7
6 =	E7
7 =	29
8 =	EF
9 =	AF
0 =	EB

The alphabet table on the right is used to produce the letters for the name. Two key presses are required for each letter.

The display table on the left is used by the program to produce the digits of the phone number. These hex vlaues can also be used in conjunction with the alphabet table if you want a digit to appear in the NAME.

A =	6F	N =	6B
B =	E6	O =	EB
C =	C3	P =	4F
D =	EC	Q =	3F
E =	C7	R =	44
F =	47	S =	A7
G =	E3	T =	46
H =	6E	U =	EA
I =	28	V =	E0
J =	E8	W =	E1
K =	67	X =	26
L =	C2	Y =	AE
M =	65	Z =	C9

at 0A0C:

	00
E	C7
N	6B
T	46
E	C7
R	44
	00
I	28
N	6B
D	EC
E	C7
X	26
	00
N	6B
o	E4
	00
0	EB
0	EB
.	04
3	AD
6	E7
	00
P	4F
R	44
E	C7
S	A7
S	A7
	00
E	C7
	00
O	00
N	00
E	00
	0
	00
10	10

at 0A2C:

	00
E	C7
N	6B
T	46
E	C7
R	44
	00
N	6B
A	6F
M	65
E	C7
	00
E	C7
N	6B
T	46
E	C7
R	44
	00
F	47
	00
E	C7
N	6B
T	46
E	C7
R	44
	00
P	4F
H	6E
O	EB
N	6B
E	C7
	00
N	6B
o	E4
	00
T	46
H	6E
E	C7
N	6B
	00
E	C7
	00

S	A7
P	4F
A	6F
C	C3
E	C7
=	84
F	47
	00
C	C3
L	C2
E	C7
A	6F
R	44
=	84
C	C3
	00
R	44
E	C7
T	46
U	EA
R	44
N	6B
=	84
A	6F
	00
D	EC
I	28
A	6F
L	C2
=	84
D	EC
	00
E	C7
N	6B
D	EC
=	84
E	C7
	00
	00
	00
10	10

SUMMARY

THE FLIP FLOP FORMS THE BASIC ELEMENT OF THE SEQUENTIAL LOGIC CIRCUIT. THIS IS AN EXTREMELY VALUABLE BUILDING BLOCK AS IT POSSESSES:

1. MEMORY CAPABILITY
2. A DIVIDE BY TWO FEATURE

FLIP FLOPS CAN ALSO BE CALLED LATCHES, DIVIDERS OR MULTIVIBRATORS, DEPENDING ON HOW THEY ARE WIRED INTO A CIRCUIT.

THE MOST BASIC FLIP FLOP IS CALLED THE R-S FLIP FLOP. IT HAS 2 INPUTS CALLED S FOR SET & R FOR RESET. THE ONLY OTHER LINES ARE THE OUTPUT LINES. THESE ARE LABELED Q FOR THE NORMAL OUTPUT AND Q (Q-BAR) FOR THE COMPLEMENTARY OUTPUT.

WHEN THE NORMAL OUTPUT IS HIGH, THE FLIP FLOP IS SAID TO BE SET. WHEN THE NORMAL OUTPUT IS LOW THE FLIP FLOP IS SAID TO BE RESET.

FLIP FLOPS NORMALLY HOLD DATA FOR A SHORT PERIOD OF TIME & THE R-S FLIP FLOP IS OFTEN CALLED AN R-S LATCH.

BUT THE R-S FLIP FLOP SUFFERS FROM ONE LIMITATION. IF BOTH INPUTS ARE HELD LOW THE OUTPUTS BOTH BECOME HIGH. THIS PRODUCES AN UNDESIRABLE RESULT AND THIS STATE MUST NOT BE ALLOWED TO OCCUR.

THE RESET-SET FLIP FLOP CAN BE USED TO DEBOUNCE A MECHANICAL SWITCH, OR STORE DATA. IT IS AN ASYNCHRONOUS DEVICE AS THE OUTPUT CHANGES IMMEDIATELY THE INPUTS ARE ACTIVATED.

TO ACHIEVE A TIMING CONDITION A CLOCKED R-S FLIP FLOP HAS BEEN PRODUCED. THE INCLUSION OF A CLOCK LINE MEANS THE FLIP FLOP WILL NOT CHANGE UNTIL THE ARRIVAL OF THE CLOCK PULSE. — IT OPERATES SYNCHRONOUSLY.
HOWEVER IT DOES POSSESS AN UNDESIRABLE OR PROHIBITED STATE WHERE BOTH OUTPUTS GO HIGH. TO OVERCOME THIS CONDITION A "D" LATCH HAS BEEN PRODUCED. THE D FLIP FLOP HAS AN INVERTER, FITTED TO THE RESET LINE & FEEDS FROM THE SET LINE SO THAT THE TWO INPUTS ARE ALWAYS OUT-OF-PHASE.

AT THE TOP OF THE RANGE IS THE J-K FLIP FLOP. IT OVER—COMES ALL THE LIMITATIONS OF THE PREVIOUS TYPES HOWEVER IT IS COMPLEX AND EXPENSIVE AND IS GENERALLY USED ONLY WHEN REQUIRED.
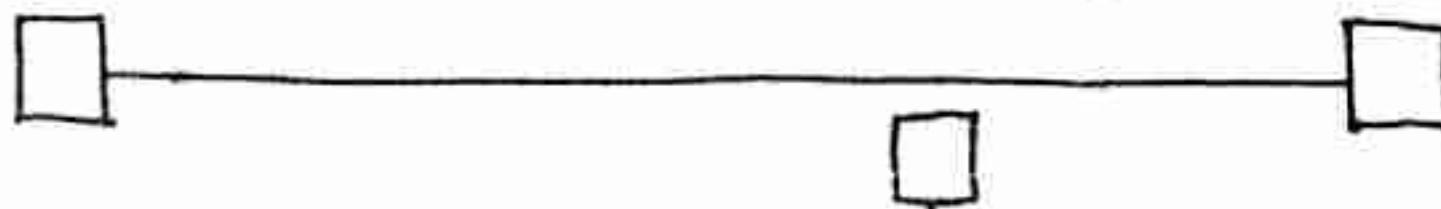
IT IS CAPABLE OF PERFORMING ALL THE OPERATIONS OF THE SIMPLER TYPES, AND MORE.

DEPENDING ON THE STATE OF THE INPUTS, THE OUTPUTS WILL PRODUCE 4 DIFFERENT EFFECTS:
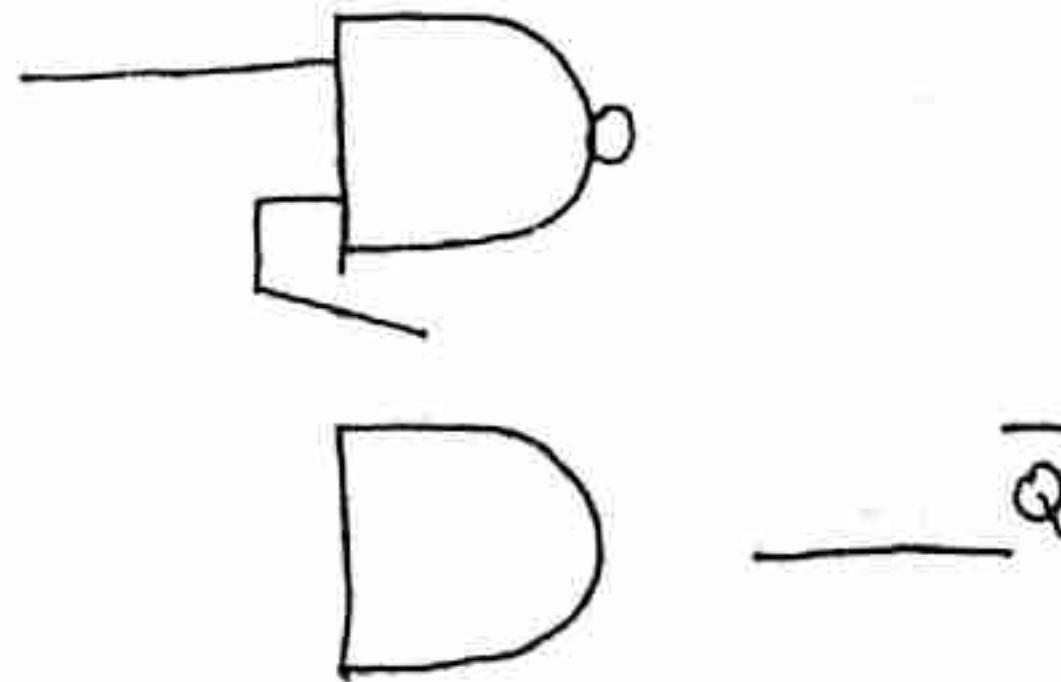
1. FREEZE.
2. REMAIN OR CHANGE TO SET. CONDITION
3. " " " " RESET CONDITION.
4. TOGGLE.

THE TOGGLE FEATURE CAN BE USED TO PRODUCE A DIVIDE-BY-2 STAGE AND THE FLIP FLOP CAN BE CASCADED TO PRODUCE LONG LINES OF DIVISION STAGES. THESE SOON BECOME EFFECTIVE. WITH 7 STAGES THE DIVISION IS 128 AND 12 STAGES PRODUCES A DIVIDE-BY-4096 COUNTER.

THIS IS THE TOPIC FOR THE NEXT SECTION.

QUIZ:

1. WHEN A LATCH IS STORING A BINARY 1 IT IS IN THE ____ (SET, RESET) MODE.

2. DRAW THE BLOCK DIAGRAM FOR AN R-S FLIP FLOP.

3. IF A FLIP FLOP IS RESET THE $\overline{Q}$ OUTPUT IS ____ (HIGH, LOW)

4. THE NORMAL OUTPUT OF A FLIP FLOP IS Q, $\overline{Q}$ ____ .

5. FOR AN ACTIVE LOW R-S FLIP FLOP, SUPPLYING THE R LINE WITH A ____ (HIGH, LOW) WILL ____ (CLEAR, SET) THE Q OUTPUT TO A ____ (0, 1)

6. ASSUME AN R-S LATCH IS SET. A LOW TO THE S INPUT WILL:
   (a) DO NOTHING.
   (b) CHANGE THE FLIP FLOP TO RESET.

7. BOTH INPUTS OF A NOR LATCH ARE HIGH. THE STATE OF THE LATCH IS:
   (a) SET
   (b) RESET
   (c) UNDESIRABLE OR PROHIBITED
   (d) CAN'T TELL, NOT ENOUGH INFORMATION

8. WHEN THE NAME LATCH IS IN THE "LIMBO" CONDITION BOTH OUTPUTS WILL BE: ____ (HIGH, LOW)

9. Q & $\overline{Q}$ OUTPUTS SHOULD ALWAYS BE:
   (a) THE SAME
   (b) LOW
   (c) HIGH
   (d) COMPLEMENTARY.

10. STATE 2 USES FOR A LATCH:

11. COMPLETE THIS DIAGRAM OF A NAND GATE LATCH.

12. A FLIP FLOP OPERATING IN STEP WITH A CLOCK IS SAID TO BE OPERATING:
   (a) SYNCHRONOUSLY.
   (b) ASYNCHRONOUSLY.

13. WHAT DOES "R-S" STAND FOR?

14. DRAW THE BLOCK DIAGRAM FOR A CLOCKED R-S FLIP FLOP:

15. A CLOCKED R-S FLIP FLOP OPERATES:
    (a) ASYNCHRONOUSLY.
    (b) SYNCHRONOUSLY.

16. THE Q OUTPUT IS HIGH WHEN THE CLOCK IS _____ (HIGH, LOW) & THE SET OUTPUT IS _____ (HIGH, LOW) & THE RESET LINE IS _____ (HIGH, LOW)

17. THE $\overline{Q}$ OUTPUT OF A D FLIP FLOP IS LOW. THE FLIP FLOP IS SAID TO BE IN THE _____ (SET RESET) STATE.

18. DATA AT THE D INPUT IS TRANSFERRED TO THE Q OUTPUT ON THE H-TO-L OR L-TO-H _____ TRANSITION OF THE CLOCK PULSE FOR A NAND GATE D FLIP FLOP.

19. FOR A D FLIP FLOP THE S&R LINES ARE ALWAYS:
    (a) HIGH
    (b) LOW
    (c) OUT-OF-PHASE.

20. THE CLOCK LINE DETERMINES THE STATE OF THE FLIP FLOP.
    (a) TRUE
    (b) FALSE.

21. WHY DOES A D FLIP FLOP HAVE AN INVERTER AT THE INPUT OF ONE LINE ?

22. DRAW THE LOGIC SYMBOL FOR A J-K FLIP FLOP:

23. LIST THE 4 SYNCHRONOUS MODES OF OPERATION OF THE J-K FLIP FLOP:
    A.
    B.
    C.
    D.

24. THE TWO INTERNAL SECTIONS OF A J-K FLIP FLOP ARE:
    (i)
    (ii)

25. NAME 2 FLIP FLOPS WHICH HAVE A CLOCK INPUT LINE:
    1.
    2.

26. THE OUTPUT STATE OF A J-K FLIP FLOP IS DETERMINED BY THE MASTER OR SLAVE LATCH ?

27. IF THE $\overline{Q}$ OUTPUT GOES HIGH - LOW - HIGH - LOW; THE FLIP FLOP IS IN WHAT MODE OF OPERATION:
    A:

28. A J-K FLIP FLOP IS CAPABLE OF PROVIDING A DIVISION. HOW MANY ARE REQUIRED TO PRODUCE A DIVIDE-BY-32.

29. IF THE J-K INPUTS ARE HIGH. WHAT IS THE MODE ?

30. CAN THE J-K FLIP FLOP PRODUCE THE PROHIBITED OR UNDESIRABLE STATE ?
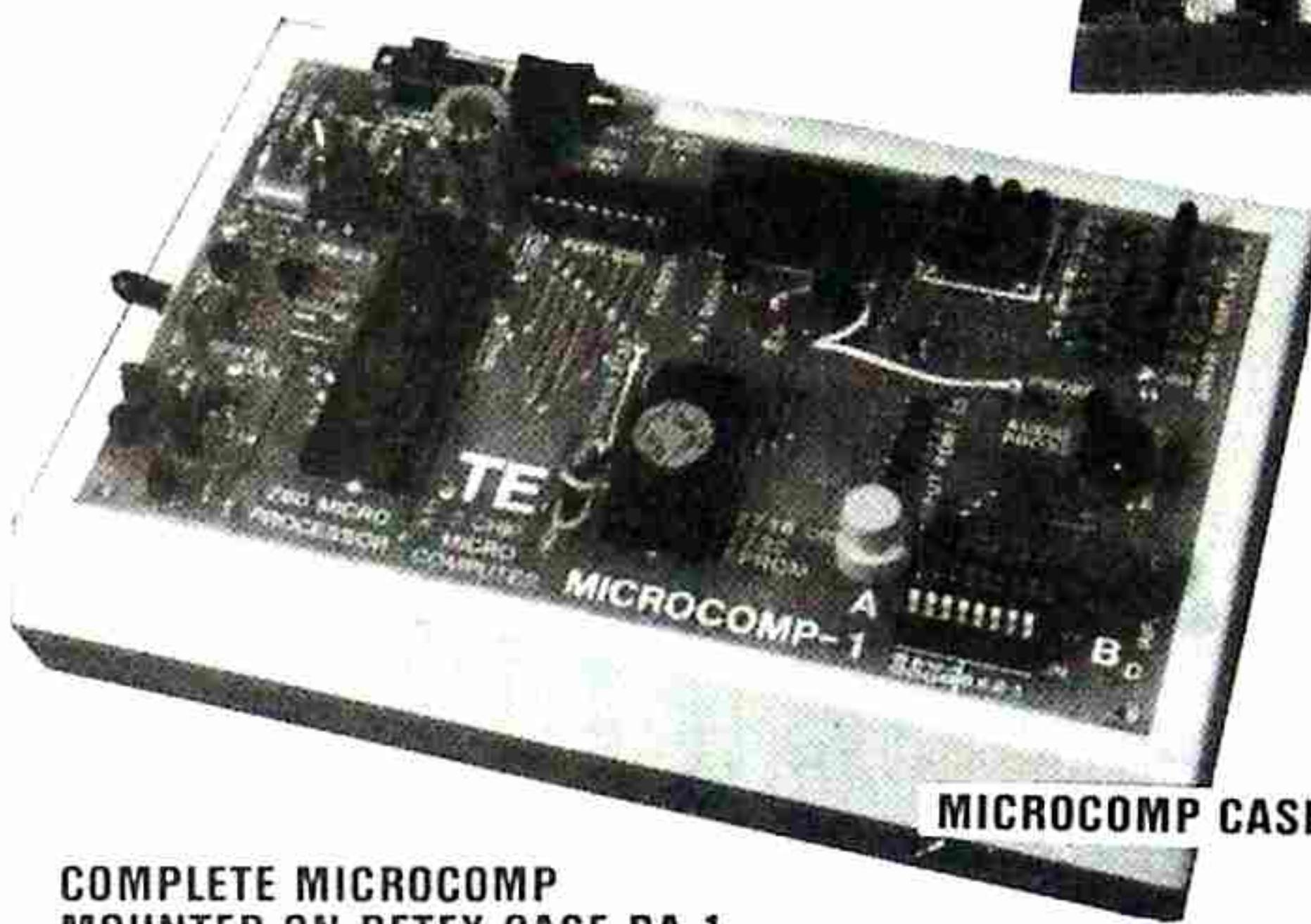
ANSWERS:

1. SET. 3. HIGH. 4. Q 5. LOW, CLEAR, 0.
6.(a) 7.(c) 8. LOW 9.(d) 10. MEMORY, DIVISION, DEBOUNCE 12. (a) 13 RESET-SET. 15 (b)
16. HIGH, HIGH, LOW. 17. SET. 18. L TO H.
19.(c) 20 (b) 21. TO PRODUCE OUT OF PHASE S&R LINES. 23. HOLD, SET, RESET, TOGGLE.
24. MASTER LATCH, SLAVE LATCH. 25 D, J-K, CLOCKER R-S. 26. SLAVE. 27. TOGGLE.
28. 5. 29. TOGGLE 30. NO.

$^{25}/_{30}$ OR HIGHER IS EXCELLENT.

MICRO COMP

A 3-CHIP Z-80 COMPUTER

$59.95 COMPLETE
COMES WITH **FREE** STORAGE BOX!!

Part 'I

TO BE RELEASED

MORSE TRAINER
$13.30 complete

COMPLETE MICROCOMP MOUNTED ON RETEX CASE RA-1.

MICROCOMP CASE $15.00

Kit of parts: $50.70
PC Board: $10.20
Complete: $59.95

This is the second article on the Microcomp and by now we have whet a lot of appetites.

Some constructors have gone way beyond that covered in the first article and investigated many of the remaining programs in the EPROM.

One constructor even listed the entire contents by using the LOOKING AT DATA routine at 0200. There were a couple of mistakes in his listing where he forgot to change from PROGRAM to DATA. This is one of the problems when trying to disect a listing.

By now you will have some idea of how the bytes appear in EPROM. They come in a continuous string - without spaces or identification as to the beginning or end of a sequence. If you jump into the middle of a program and look at a byte, you will not know if it is an instruction, part of an instruction or a piece of data. That's why you must start at the beginning of a listing.
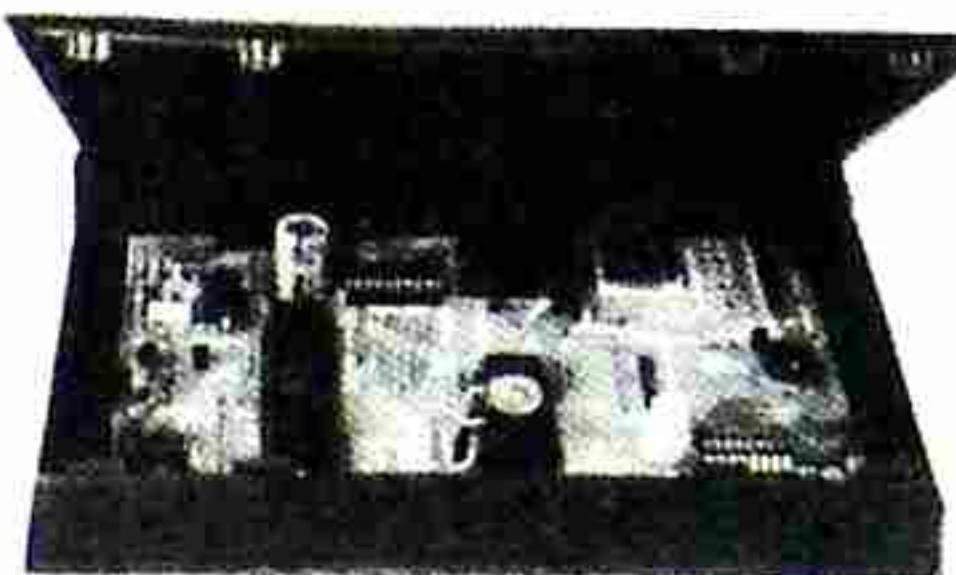
When trying to disect a program, write down the values, byte by byte and you will soon see groups which you recognise. From there you can place the others in groups and start to see a program emerging.

These values are called MACHINE CODE values and are used by the micro directly. It doesn't need spaces or stops and starts as it is pre-programmed within and knows exactly what to do.

The difficulty you would experience in disecting a program is understandable. You are not a micro and cannot keep track of the flow of the program. This is a very difficult direction to work in. The way we will be working is from IDEA-to-machine-code-listing. This is the forward direction and is much easier.
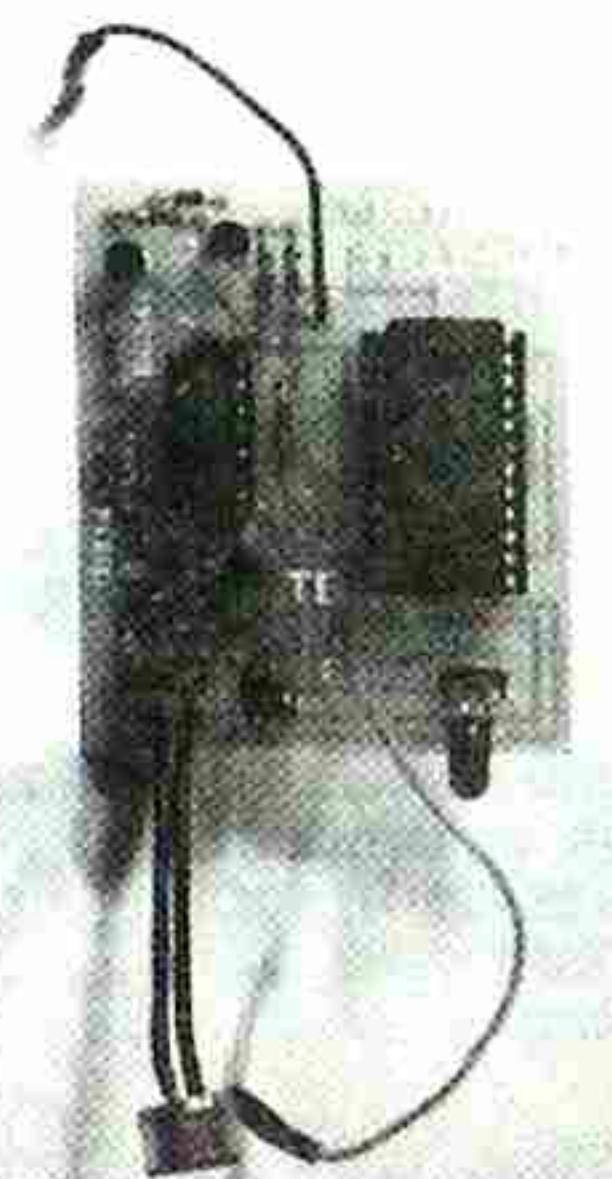
Most programs are made up of lots of small building blocks and the quickest way to learn about these is to study a few programs.

In this article we will be continuing with a close study of each of the programs in the EPROM but before we do this we have designed a couple of games for those who want to do a little programming themselves.

If you have a TEC and either the non-volatile RAM or EPROM burner, these programs can be typed into memory and transferred to the microcomp for execution.

As designed, the programs are run at page ZERO however only a few changes are required and they can be run at any other location. The details of this are included with the programs.

The two games are titled: **TUG O' WAR** and **BLACK JACK.** Alongside each is a flow diagram showing what each part of the program does. Also we have explained each instruction with a simple sentence to show how we converted each idea into a computer instruction.

Getting back to the Microcomp, we have described a few more of the 'ins' and 'outs' of computer design and especially the tricks we used to simplify the circuit.

Notebook No. 3 has just been released and it contains a number of pages on the Microcomp design as well as Z-80 Machine Code values for assembly and Disassembly. It also includes the interpretation of each instruction and a listing of computer terms. This will help you with programming and the circuit design pages will help you with input and output decoding and how the Z-80 communicates with all the other chips.

TUG O' WAR & BLACK JACK

TWO programs for the MICROCOMP.

These two programs bring together the TEC computer, Non-volatile RAM and Microcomp. They show some of the techniques of displaying, inputting and running a program at a speed suitable for human involvement.

These games were developed on the above equipment and you can create similar programs or adapt them to suit your own requirements.

TUG O' WAR
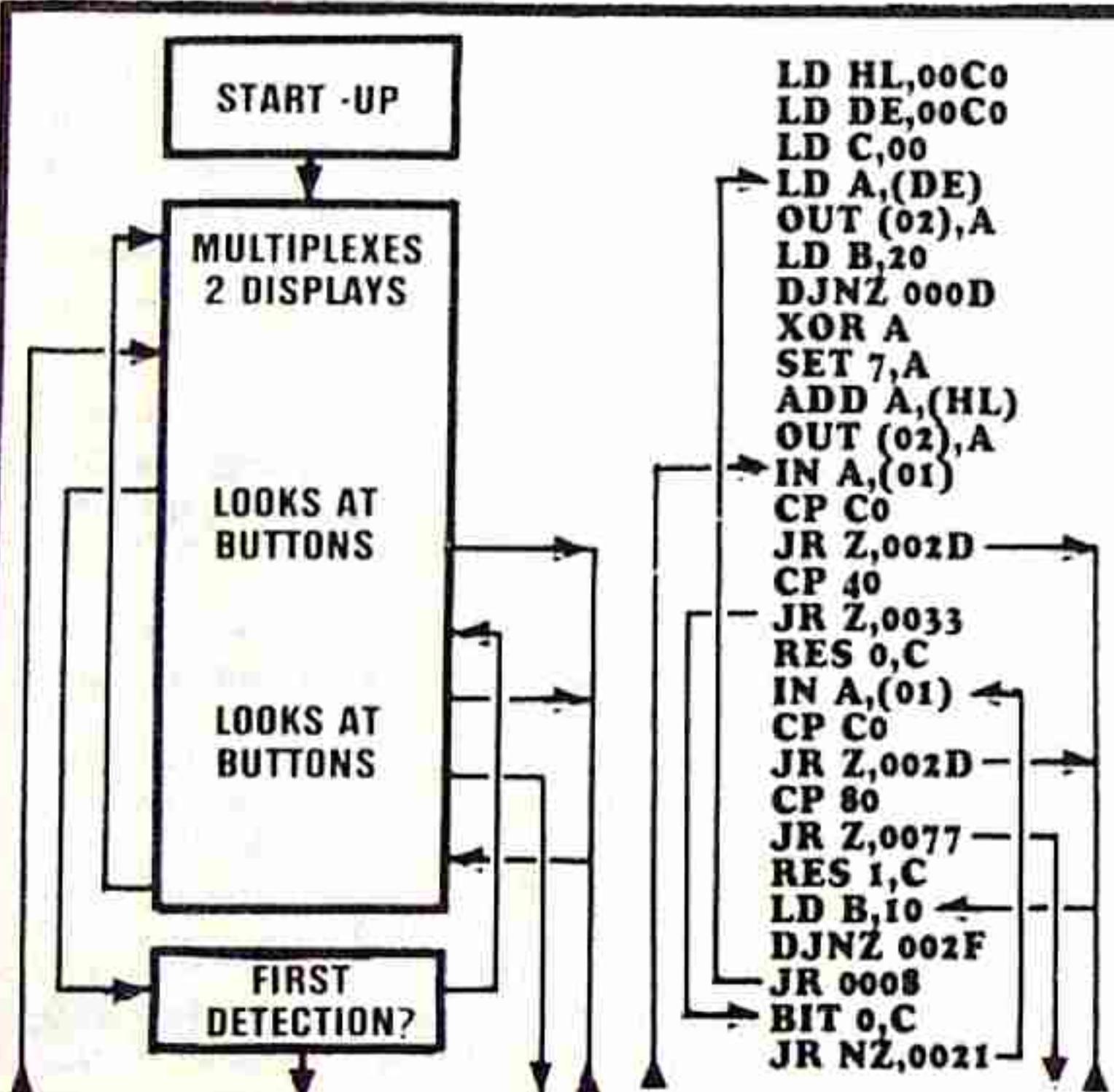
Instead of making a TUG O' WAR game from a kit, you can create an improved version by producing a program and running it on a computer.

Initially we saw this game in a popular electronics magazine and liked the way it worked.

It used a row of 15 LEDs and by pressing one of two buttons, a single illuminated LED would move towards you. Seven LEDs were available for each player and your opponent had the same opportunity to make the LED travel towards himself.

The difficulty of play could NOT be adjusted and a player would win whenever he pressed his button seven times more than his opponent.

TUG O' WAR PROGRAM:

In our version, we have made it increasingly more difficult to reach the end by weighing the table of increments.

The lowest value has only one corresponding value in the table whereas the highest value requires nine steps before it will advance to WIN!

This can be seen by referring to the byte table and counting the number of bytes for each output value.

Not only does this program show you some new techniques in programming but will also save you a few dollars, if you already have the items mentioned above.

In a similar way, lots of other ideas and games can be produced and this will save you the expense of buying special PC boards and unusual chips.

Our version has nine steps and requires a total of 45 pushes for one player to win over his opponent.

This makes the game quite difficult and you have to introduce quite a lot of strategy to win.

DESIGNING THE PROGRAM

When designing a program, the first thing you have to consider is the hardware available. In our case this means the program has to be designed around two push buttons and two 7-segment displays. The row of 8 LEDs does not give us sufficient scope.

The two displays can be used to display numbers, letters, or individual segments. We opted to display the numbers 0-9.

The rest of the effect lies in the program.

This is how we went about designing it:

When the game starts, the two displays are illuminated with zeros. This requires a
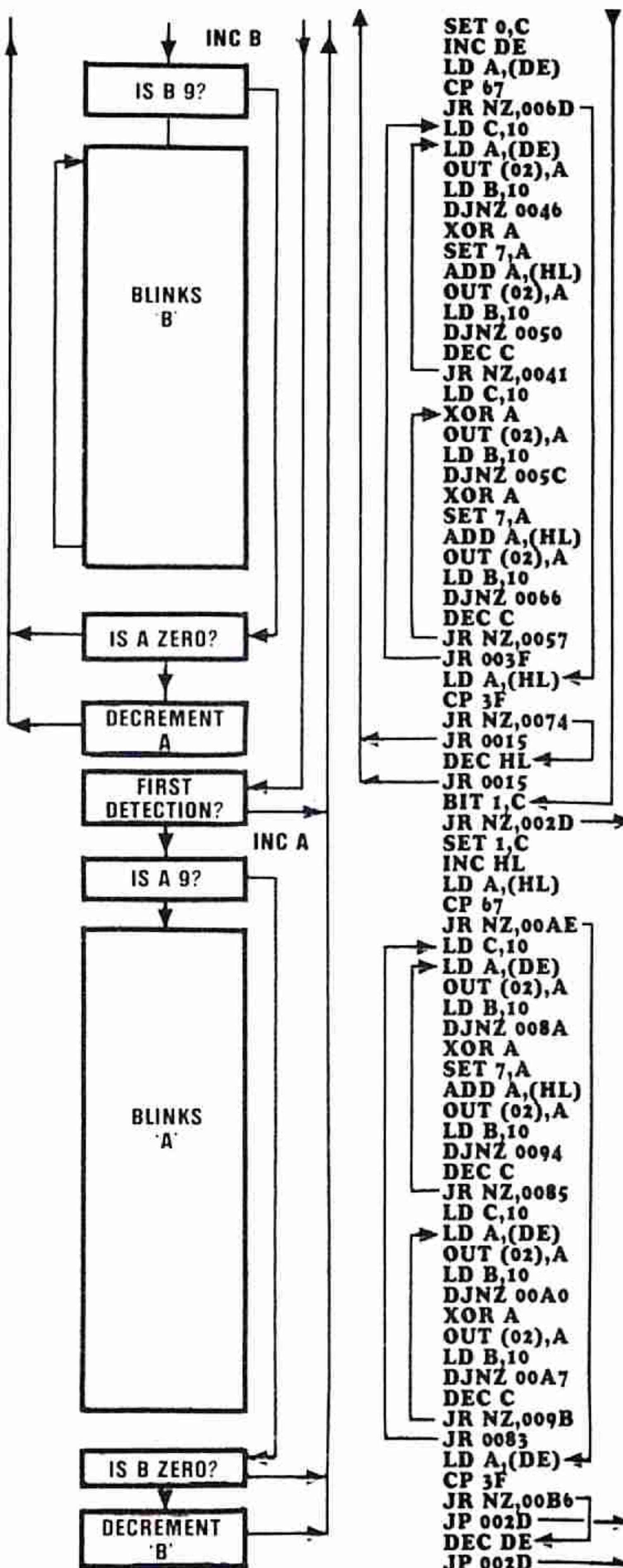
The TUG O WAR program starts below and continues on the next page. It requires a table of 46 bytes for the display and this is placed at **00C0**:

AT C0:

3F	7D
06	7D
06	7D
5B	7D
5B	7D
5B	07
4F	07
4F	07
4F	07
4F	07
66	07
66	07
66	07
66	7F
66	7F
6D	7F
6D	7F
6D	7F
6D	7F
6D	7F
6D	7F
7D	7F
7D	67

START -UP	LD HL,00C0	0000	21 C0 00	Load HL with start of table for Left Hand display.

```
LD HL,00C0      0000   21 C0 00   Load HL with start of table for Left Hand display.
LD DE,00C0      0003   11 C0 00   Load DE with start of table for Right Hand display.
LD C,00         0006   0E 00      Load the BIT TESTING register with zero.
LD A,(DE)       0008   1A         Load the accumulator with the first byte in the table.
OUT (02),A      0009   D3 02      Output this value to the latch.
LD B,20         000B   06 20      Load B with a value for a delay routine.
DJNZ 000D       000D   10 FE      Create 32 loops of decrementing register B.
XOR A           000F   AF         Zero the accumulator.
SET 7,A         0010   CB FF      Set the highest BIT so that the LH display will illuminate.
ADD A,(HL)      0012   86         ADD the byte looked at by the HL register, to the accumulator.
OUT (02),A      0013   D3 02      Output to the latch.
IN A,(01)       0015   DB 01      Look at the switches.
CP C0           0017   FE C0      Compare C0 with the accumulator to see if both switches are pressed.
JR Z,002D       0019   28 12      Jump if both switches are pressed.
CP 40           001B   FE 40      Compare the accumulator with 40 to see if B is pressed.
JR Z,0033       001D   28 14      Jump if B is pressed.
RES 0,C         001F   CB 81      Reset bit 0 of the C register.
IN A,(01)       0021   DB 01      Look at the input port.
CP C0           0023   FE C0      Compare the accumulator with C0 to see if both switches are pressed.
JR Z,002D       0025   28 06      Jump if both are pressed.
CP 80           0027   FE 80      Compare the accumulator with 80 to see if A is pressed.
JR Z,0077       0029   28 4C      Jump if A is pressed.
RES 1,C         002B   CB 89      Reset bit 1 of the C register.
LD B,10         002D   06 10      Load B with 10 for a short delay.
DJNZ 002F       002F   10 FE      Create 16 loops of decrementing register B.
JR 0008         0031   18 D5      Jump to start of multiplexing routine.
BIT 0,C         0033   CB 41      Test bit 0 to see if it is the first time B is detected.
JR NZ,0021      0035   20 EA      Jump if not the first time.
```

MULTIPLEXES 2 DISPLAYS

LOOKS AT BUTTONS

LOOKS AT BUTTONS

FIRST DETECTION?

Flowchart (left column):

- INC B
- IS B 9?
- BLINKS 'B'
- IS A ZERO?
- DECREMENT A
- FIRST DETECTION?
- INC A
- IS A 9?
- BLINKS 'A'
- IS B ZERO?
- DECREMENT 'B'

Mnemonic	Address	Hex	Explanation
SET 0,C	0037	CB C1	Set bit 0 of C before processing button B.
INC DE	0039	13	Increment pointer for RH display.
LD A,(DE)	003A	1A	Load A with second byte in table.
CP 67	003B	FE 67	Compare the accum. with 67 to see if end of table has been reached.
JR NZ,006D	003D	20 2E	Jump if end of table NOT reached. Increment if reached.
LD C,10	003F	0E 10	Load C with 10 for multiplexing time-length.
LD A,(DE)	0041	1A	Load the accumulator with data pointed to by DE.
OUT (02),A	0042	D3 02	Output to the latch.
LD B,10	0044	06 10	Load B with 10 for short delay.
DJNZ 0046	0046	10 FE	Decrement B 16 times.
XOR A	0048	AF	Zero the accumulator.
SET 7,A	0049	CB FF	Set the highest bit to turn on the LH display.
ADD A,(HL)	004B	86	ADD the byte pointed to by HL, to the accumulator.
OUT (02),A	004C	D3 02	Output to the latch.
LD B,10	004E	06 10	Load B with 10 for short delay.
DJNZ 0050	0050	10 FE	Decrement B 16 times.
DEC C	0052	0D	Decrement C.
JR NZ,0041	0053	20 EC	Jump if C not zero. Increment if C is zero.
LD C,10	0055	0E 10	Load C with 10.
XOR A	0057	AF	Zero A to turn off RH display to create BLINK.
OUT (02),A	0058	D3 02	Output to the latch.
LD B,10	005A	06 10	Load B with 10 to create a short delay.
DJNZ 005C	005C	10 FE	Decrement B 16 times.
XOR A	005E	AF	Zero A.
SET 7,A	005F	CB FF	SET the highest bit of the accumulator to turn on the LH display.
ADD A,(HL)	0061	86	ADD the byte pointed to by the HL pair, to the accumulator.
OUT (02),A	0062	D3 02	Out put to the latch.
LD B,10	0064	06 10	Load B with 10.
DJNZ 0066	0066	10 FE	Decrement B 16 times.
DEC C	0068	0D	Decrement C.
JR NZ,0057	0069	20 EC	Jump if C not zero.
JR 003F	006B	18 D2	Jump to start of BLINKING ROUTINE.
LD A,(HL)	006D	7E	Load A with data byte pointed to by HL.
CP 3F	006E	FE 3F	Compare with 3F to see if LH display is zero.
JR NZ,0074	0070	20 02	Jump if not zero.
JR 0015	0072	18 A1	Jump to start of program if zero.
DEC HL	0074	2B	Decrement player A pointer.
JR 0015	0075	18 9E	Jump to start of program.
BIT 1,C	0077	CB 49	TEST bit 1 of the C register.
JR NZ,002D	0079	20 B2	Jump if bit 1 is SET. Increment to next instruction if not set.
SET 1,C	007B	CB C9	SET bit 1 of the C register.
INC HL	007D	23	Increment player A pointer.
LD A,(HL)	007E	7E	Load the data byte into the accumulator.
CP 67	007F	FE 67	Compare the accumulator with 67.
JR NZ,00AE	0081	20 2B	Jump if the two are not the same. Go to next instruction if the same.
LD C,10	0083	0E 10	The next 25 instructions produce a multiplexing effect on the two displays so that the LH display turns on and off in a BLINKING pattern.
LD A,(DE)	0085	1A	
OUT (02),A	0086	D3 02	
LD B,10	0088	06 10	
DJNZ 008A	008A	10 FE	
XOR A	008C	AF	
SET 7,A	008D	CB FF	
ADD A,(HL)	008F	86	
OUT (02),A	0090	D3 02	
LD B,10	0092	06 10	
DJNZ 0094	0094	10 FE	
DEC C	0096	0D	
JR NZ,0085	0097	20 EC	
LD C,10	0099	0E 10	
LD A,(DE)	009B	1A	
OUT (02),A	009C	D3 02	
LD B,10	009E	06 10	
DJNZ 00A0	00A0	10 FE	
XOR A	00A2	AF	
OUT (02),A	00A3	D3 02	
LD B,10	00A5	06 10	
DJNZ 00A7	00A7	10 FE	
DEC C	00A9	0D	
JR NZ,009B	00AA	20 EF	
JR 0083	00AC	18 D5	
LD A,(DE)	00AE	1A	Load the accumulator with the value pointed to by DE.
CP 3F	00AF	FE 3F	Compare the accumulator with 3F to see if the RH display is zero.
JR NZ,00B6	00B1	20 03	Jump if player B is zero, increment to next instruction if not zero.
JP 002D	00B3	C3 2D 00	Jump to start of program.
DEC DE	00B6	1B	Decrement player B pointer.
JP 002D	00B7	C3 2D 00	Jump to start of program.

This section is very nearly identical to the instructions between **003F** to **006B**. Refer to the above for the explanations.

loop in which the value for each display is looked after by a separate register pair. The left hand display is looked after by the HL register pair and the right hand display by the DE register pair.

This choice is goverened by the fact that the HL pair has a larger number of op-codes available to us and thus it is more versatile.

You will see the need for this later.

Numbers produced on the right hand display can be created on the left hand display simply by turning on the highest line at the same time. This is done by adding '80' to the value of data. The same effect can be created by SETTING bit 7 of the accumulator and then ADDing the value of the right hand display. This is what we have done. The data required to produce a number in the right hand display has been added to the accumulator after the highest bit has been SET, with the result that the number appears on the left hand display.

Before this can be done, there is one point which must be remembered.

The accumulator must firstly be cleared so that all bits are zero. SETTING a bit and ADDing to the accumulator does not clear out any initial junk.

Using these facts, and a short DJNZ delay, will produce a loop program which will illuminate both displays.

Also in this loop we must include an instruction to look at the input port and detect 3 things:

We must detect if button A is pressed, button B and also if both buttons are pressed at the same time.

Detecting button A will cause the program to branch to a sub-routine, button B to another sub-routine and both buttons will cause the program to jump over the other branch-instructions.

When the micro jumps to either sub-routine, there are 4 instructions which must be taken into account.

Firstly it looks to see if it is the first time the sub-routine has been jumped to (during this press of the button). It does this by checking the debounce BIT in the C register. We must create a debounce condition so that the displays will increment only one byte in the table for each press of the button. This is achieved by resetting the BIT(s) in the C register while executing the main program. When a button is pressed, the micro goes to the sub-routine and looks at the particular bit in question.

If it is in a RESET state, the micro runs through the sub-routine and SETs the bit. It then increments the pointer register to look at the next byte in the table. It then compares the value with 67 to see if the end of the table has been reached. If it has, it goes to a loop program which flashes the winning display.

If the end of the table has not been reached, the program looks at the opposition value to see if it is zero. If it is zero, the micro returns to the main program. If the opposition is not zero, it decrements the pointer register and jumps to the main program.

The effect on the screen may or may not be an increment or decrement, depending on the position of the pointer registers, however you can be assured the byte table has been decremented and/or incremented correctly.

All you have to do now is put these facts into a machine code program.

When doing this, it is very helpful to use arrows to incdicate where the program jumps to. You can also put labels and notes at various locations to indicate what the program is doing. This will assist you when debugging and tidying up.

Study the program on the previous 2 pages and see how it's done.

●●●●●●●●●●●●●

BLACK JACK

This program is designed around Paul's Black Jack in issue 11.

The concept of the program is to deal a hand of random values exactly like playing cards.

It then keeps a tally of your hand and adjusts the total to your advantage when one or more ACES are dealt.

It is the feature of the Ace being equal to 1 or 11 which adds interest to the game and brings a little strategy into the program.

Apart from the normal requirements, the program must keep track of an ace. When one is included, BIT 7 of the C register is SET. The C register is our TEST REGISTER.

The computer keeps dealing cards until a value over 21 is reached. It then looks to see if an ace is included by testing BIT 7. If this bit is SET, it subtracts ten from the total, making the ace equal to one.

Further cards are dealt and once again a score is kept, in an attempt to reach 21.

When exactly 21 is reached, the program jumps to a routine which flashes '21' and at the same time looks at the input port for button B being pressed. If it is pressed, the program returns to the start.

The other important feature to remember when producing a program is TIMING. By this we mean the length of time for the things to be done, such as the numbers appearing on the screen.

If they appear for too short a duration, it will be annoying. A long duration will slow down the game.

These periods are controlled by a delay routine which is inserted into the program to 'waste computer time'.

The length of these delays depends on the clock speed and since we have a very slow clock frequency, we have delay routines to match.

Our maximum clock speed is 35,000 cycles per second so that if we waste 35,000 clock cycles, we produce a delay of 1 second.

The simplest way of producing a delay is to use **DJNZ** The maximum DJNZ delay is produced by loading B with FF and this wastes 13 x 255 cycles (3315 cycles) or about 1/10th sec. Longer delays can be obtained by using 2 DJNZ's and shorter delays by decreasing the value of B.

The other way to create a delay is to run through a loop which gradually decrements a delay value. This type of program is necessary when multiplexing is required.

The only way of obtaining a suitable value for the delay is to study some of the examples.

If you are unsure, insert '80' and trim the value during final testing. '80' represents a mid-value and you can increase or decrease it later.

INDEXED ADDRESSING

Black Jack uses a table (located at the end of the program) which does three things. Firstly it determines the character to appear on the right hand display, then the character for the left hand display and finally the equivalent hex value.

This requires 3 bytes which we have grouped together to form a 'block'.

Even when the left hand display is not showing a value, it is being accessed with a zero output so that uniform illumination is produced when a value such as '10' is displayed.

To pick up the 2nd and 3rd byte in each group, we have used INDEXED ADDRESSING.

This is a handy way of jumping down a table without incrementing the register.

If you were to increment it, you would have to decrement it before the start of the next loop and this would involve extra instructions.
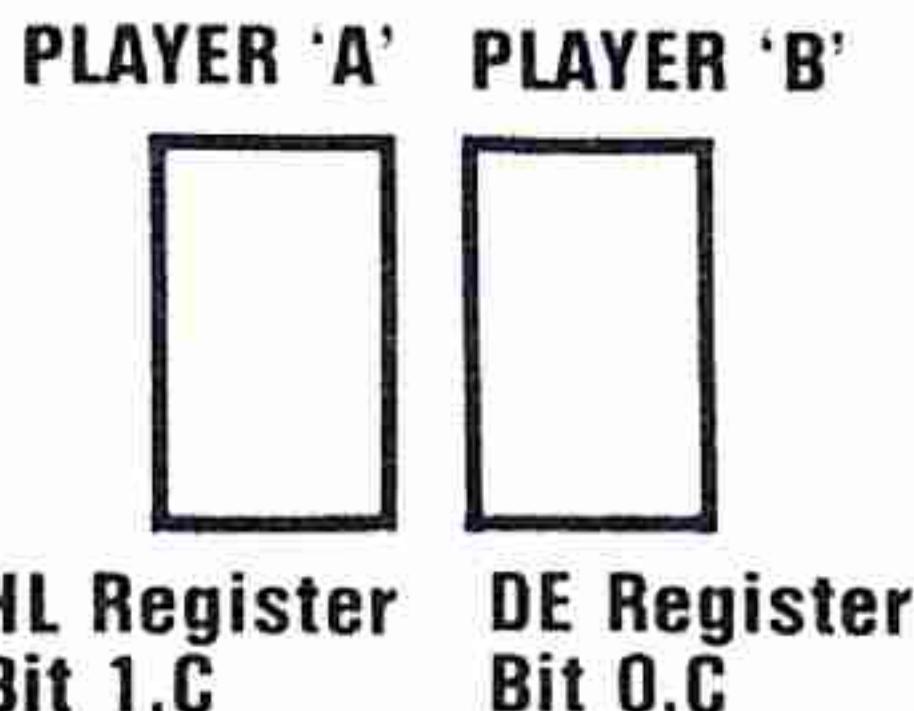
In our program, the register in charge of the table is incremented only after a multiplexing operation (which may involve a number of passes of a loop).

When the register is incremented, it is incremented 3 times so that it looks at the first byte of the next group. That is the 1st, 4th, 7th 10th byte etc.

The 2nd and 3rd bytes of each group are looked at via the indexing feature which uses a displacement value. For instance (IX + 01) looks at the second byte and (IX + 02) looks at the 3rd byte.

RELOCATING THE PROGRAM

Although the program is designed for the Microcomp and to be run at page zero, it can be shifted to any other location by simply changing all the absolute address values.

PLAYER 'A'   PLAYER 'B'

**HL Register   DE Register
Bit 1,C   Bit 0,C**

The diagram shows the two displays and their associated register pair. The Debounce is done in register 'C'.

There are two main types of addressing. ABSOLUTE and RELATIVE. Relative values refer to locations by using a displacement value in the program and whenever the program is shifted, these values remain unchanged.

However absolute address values must be changed whenever a program is shifted as the values refer to specific locations.

In our program, the absolute values include the address of the tables and jumps which are over 80 hex bytes away. (Relative jumps can only cope with jumps less than 80 hex bytes away, in either direction).

The '5 CARD HAND' which wins if 21 is not obtained. Our program does not take this into account but it would be a simple matter to make it do so.

Here's the program: Type it on the TEC, hold it in the non-volatile RAM and play it on the Microcomp.
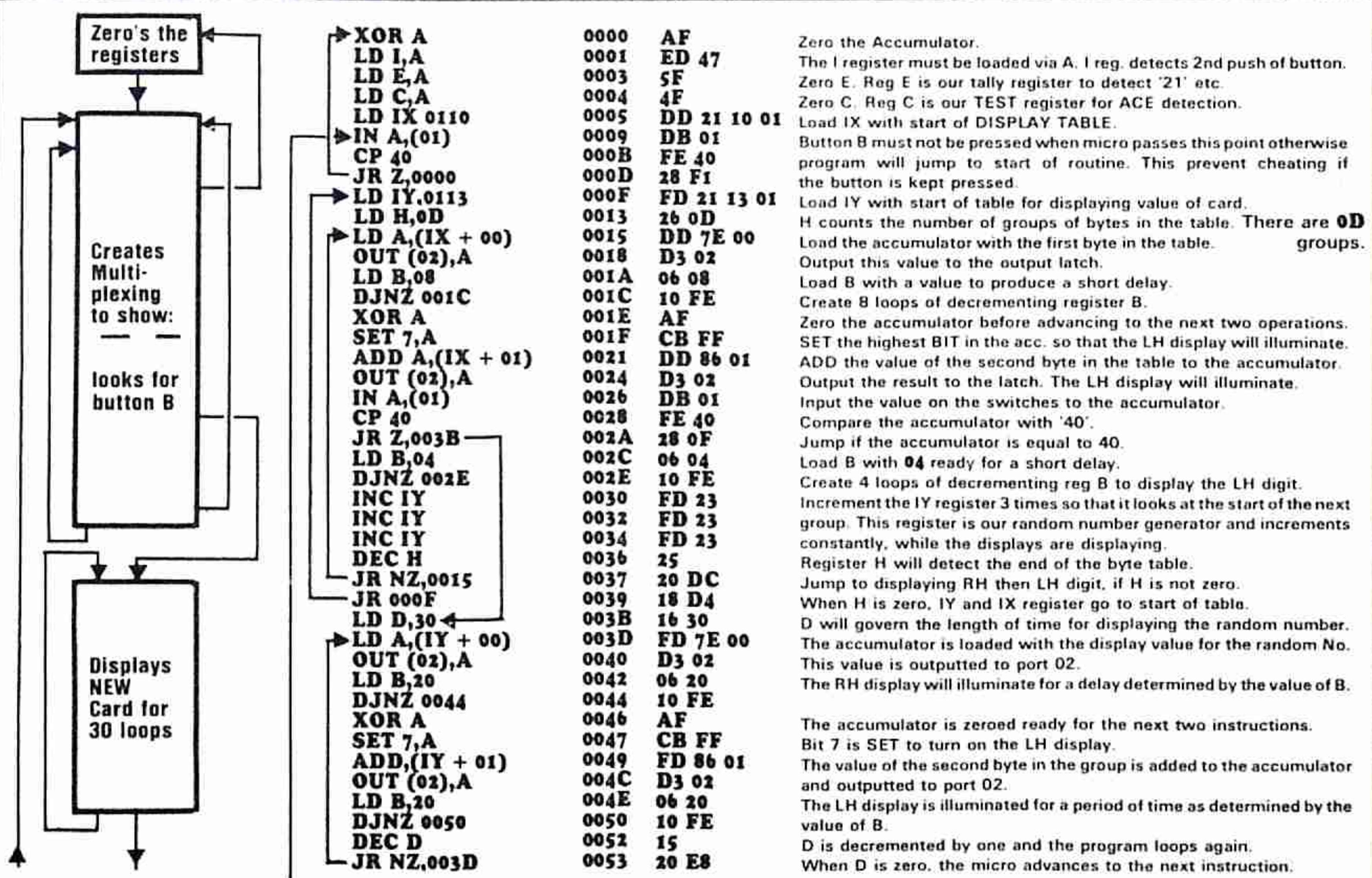
At 0100:
Each hex value produces a number from 0 to 9:
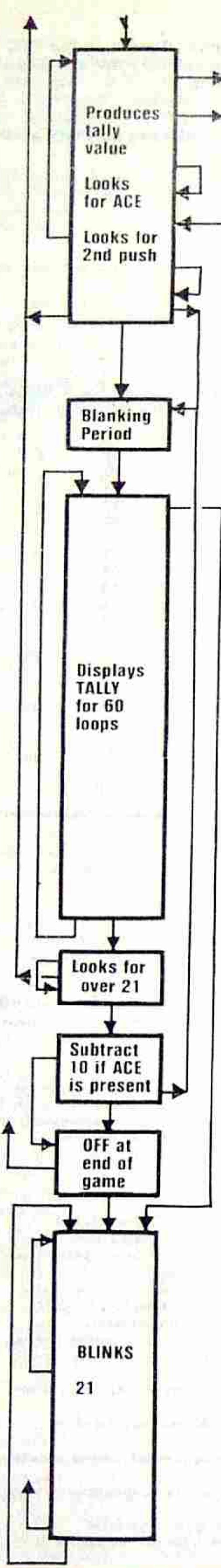
3F	0
06	1
5B	2
4F	3
66	4
6D	5
7D	6
07	7
7F	8
67	9

At 0110:
The first two bytes produce the 'CARDS' and the third byte holds the value of the card.

40	-	7F	8
40	-	00	
00		08	
5B	2	67	9
00		00	
02		09	
4F	3	77	A
00		00	
03		0B	
66	4	1E	J
00		00	
04		0A	
6D	5	3F	10
00		06	
05		0A	
7D	6	3F	10
00		06	
06		0A	
07	7	3F	10
00		06	
07		0A	

		Address	Code	
Zero's the registers	XOR A	0000	AF	Zero the Accumulator.
	LD I,A	0001	ED 47	The I register must be loaded via A. I reg. detects 2nd push of button.
	LD E,A	0003	5F	Zero E. Reg E is our tally register to detect '21' etc.
	LD C,A	0004	4F	Zero C. Reg C is our TEST register for ACE detection.
	LD IX,0110	0005	DD 21 10 01	Load IX with start of DISPLAY TABLE.
	IN A,(01)	0009	DB 01	Button B must not be pressed when micro passes this point otherwise
	CP 40	000B	FE 40	program will jump to start of routine. This prevent cheating if
	JR Z,0000	000D	28 F1	the button is kept pressed.
Creates Multiplexing to show:	LD IY,0113	000F	FD 21 13 01	Load IY with start of table for displaying value of card.
	LD H,0D	0013	26 0D	H counts the number of groups of bytes in the table. There are **0D** groups.
	LD A,(IX + 00)	0015	DD 7E 00	Load the accumulator with the first byte in the table.
	OUT (02),A	0018	D3 02	Output this value to the output latch.
	LD B,08	001A	06 08	Load B with a value to produce a short delay.
	DJNZ 001C	001C	10 FE	Create 8 loops of decrementing register B.
—	XOR A	001E	AF	Zero the accumulator before advancing to the next two operations.
looks for button B	SET 7,A	001F	CB FF	SET the highest BIT in the acc. so that the LH display will illuminate.
	ADD A,(IX + 01)	0021	DD 86 01	ADD the value of the second byte in the table to the accumulator.
	OUT (02),A	0024	D3 02	Output the result to the latch. The LH display will illuminate.
	IN A,(01)	0026	DB 01	Input the value on the switches to the accumulator.
	CP 40	0028	FE 40	Compare the accumulator with '40'.
	JR Z,003B	002A	28 0F	Jump if the accumulator is equal to 40.
	LD B,04	002C	06 04	Load B with **04** ready for a short delay.
	DJNZ 002E	002E	10 FE	Create 4 loops of decrementing reg B to display the LH digit.
	INC IY	0030	FD 23	Increment the IY register 3 times so that it looks at the start of the next
	INC IY	0032	FD 23	group. This register is our random number generator and increments
	INC IY	0034	FD 23	constantly, while the displays are displaying.
	DEC H	0036	25	Register H will detect the end of the byte table.
	JR NZ,0015	0037	20 DC	Jump to displaying RH then LH digit, if H is not zero.
	JR 000F	0039	18 D4	When H is zero, IY and IX register go to start of table.
	LD D,30	003B	16 30	D will govern the length of time for displaying the random number.
	LD A,(IY + 00)	003D	FD 7E 00	The accumulator is loaded with the display value for the random No.
Displays NEW Card for 30 loops	OUT (02),A	0040	D3 02	This value is outputted to port 02.
	LD B,20	0042	06 20	The RH display will illuminate for a delay determined by the value of B.
	DJNZ 0044	0044	10 FE	
	XOR A	0046	AF	The accumulator is zeroed ready for the next two instructions.
	SET 7,A	0047	CB FF	Bit 7 is SET to turn on the LH display.
	ADD,(IY + 01)	0049	FD 86 01	The value of the second byte in the group is added to the accumulator
	OUT (02),A	004C	D3 02	and outputted to port 02.
	LD B,20	004E	06 20	The LH display is illuminated for a period of time as determined by the
	DJNZ 0050	0050	10 FE	value of B.
	DEC D	0052	15	D is decremented by one and the program loops again.
	JR NZ,003D	0053	20 E8	When D is zero, the micro advances to the next instruction.

Flowchart boxes (left column, top to bottom):

- Produces tally value
- Looks for ACE
- Looks for 2nd push
- Blanking Period
- Displays TALLY for 60 loops
- Looks for over 21
- Subtract 10 if ACE is present
- OFF at end of game
- BLINKS 21

Instruction	Address	Machine Code	Comment
LD A,(IY + 02)	0055	FD 7E 02	Load A with the 3rd byte in the group. We know the byte must have a
INC E	0058	1C	value of one or greater and so we can safely INCrement E
BIT 1,E	0059	CB 4B	Register E is our TALLY register. We require it to add the values of the
JR Z,0066	005B	28 09	cards and hold the result in decimal form. The problem comes when
BIT 3,E	005D	CB 5B	you add one to 9. The register will show 0A. We must convert 0A to ten.
JR Z,0066	005F	28 05	This can be done by a DAA instruction or by software. We have opted
LD B,06	0061	06 06	for software. We detect 0A via bit 1 and 3 being HIGH and then
INC E	0063	1C	increment the E register 6 times. Each time the tally register is
DJNZ 0063	0064	10 FD	incremented (apart from the decimal adjusting loop), the accumulator
DEC A	0066	3D	is decremented and when the accumulator is zero, the program
JR NZ,0058	0067	20 EF	advances
LD A,(IY + 00)	0069	FD 7E 00	Load the accumulator with the first byte of the group
CP 77	006C	FE 77	Compare 77 with the accumulator. We are looking for an ACE.
JR NZ,0072	006E	20 02	If the accumulator is not 77, the micro will jump to LD A,I If the
SET 7,C	0070	CB F9	accumulator is 77, the program will advance to the next instruction
LD A,I	0072	ED 57	and SET bit 7 of the C register.
INC A	0074	3C	The I register counts the number of presses of the B button. We are
LD I,A	0075	ED 47	looking for 2 or more presses so that the tally can be displayed. This is
CP 02	0077	FE 02	the advantage of using the CARRY command
JR NC,007D	0079	30 02	The micro jumps when I is 2 or MORE
JR 0009	007B	18 8C	Jump to start of program (button B has been pressed once.)
LD D,60	007D	16 60	Register D produces the time for the tally to appear
XOR A	007F	AF	Blank the display
OUT (02),A	0080	D3 02	Output to latch
LD B,FF	0082	06 FF	Load B with maximum delay value
DJNZ 0084	0084	10 FE	Perform FF loops of decrementing register B
LD HL,0100	0086	21 00 01	Load HL with start of display values.
LD A,E	0089	7B	Load the tally register into the accumulator
CP 21	008A	FE 21	Compare 21 with the accumulator
JR Z,00D4	008C	28 46	If accumulator is 21, the micro jumps to 'BLINKING 21'
AND 0F	008E	E6 0F	If not 21, remove high nibble by ANDing with 0F
ADD A,L	0090	85	L contains '00' from address above. ADD 00 to accumulator
LD L,A	0091	6F	Load the result back into L so that the micro looks at one of the
LD A,(HL)	0092	7E	addresses of the table. Load the value it finds into A
OUT (02),A	0093	D3 02	Output the byte to the latch
LD B,10	0095	06 10	Load B with a low value.
DJNZ 0097	0097	10 FE	Create 16 loops of decrementing register B
LD A,E	0099	7B	Load the tally register into the accumulator
RRA	009A	1F	Rotate the accumulator right, effectively bringing the 4 bits of the
RRA	009B	1F	HIGH nibble to occupy the 4 lower places.
RRA	009C	1F	
RRA	009D	1F	
AND 0F	009E	E6 0F	Remove the 4 high bits by ANDing with 0F
LD HL,0100	00A0	21 00 01	Load the HL register with start of display table
ADD A,L	00A3	85	ADD L to accumulator to create a new value for L so that we look at one
LD L,A	00A4	6F	of the addresses in the table.
LD A,(HL)	00A5	7E	Load the byte from the table into the accumulator
SET 7,A	00A6	CB FF	Set bit 7 of the accumulator so that the LH display turns on
OUT (02),A	00A8	D3 02	Output this value to the latch
LD B,08	00AA	06 08	Load B with a short delay value.
DJNZ 00AC	00AC	10 FE	Create 8 loops of decrementing register B
DEC D	00AE	15	Decrement D and go to start of multiplexing loop. When D is zero,
JR NZ,0086	00AF	20 D5	increment to next instruction in program.
LD A,E	00B1	7B	Load the tally register into the accumulator
CP 22	00B2	FE 22	Compare with 22.
JR NC,00B9	00B4	30 03	If tally is 22 or MORE, increment to next instruction.
JP 0009	00B6	C3 09 00	Jump to start of program. If tally is less than 22, jump to BIT 7,C.
BIT 7,C	00B9	CB 79	Test bit 7 of the C register to see if an ACE has been dealt
JR Z,00C4	00BB	28 07	Jump if no ACE. Increment if an ACE is present
SUB 10	00BD	D6 10	Subtract ten from tally, making ACE equal to ONE
LD E,A	00BF	5F	Load the new tally into the tally register
RES 7,C	00C0	CB B9	Reset bit 7 to show ACE has been turned into ONE
JR 007D	00C2	18 B9	Jump to displaying new tally
XOR A	00C4	AF	Zero the accumulator
OUT (02),A	00C5	D3 02	Output to latch for a delay period equal to 4 DJNZ s (with B =FF) to
LD B,FF	00C7	06 FF	indicate END OF GAME
DJNZ 00C9	00C9	10 FE	
DJNZ 00CB	00CB	10 FE	
DJNZ 00CD	00CD	10 FE	
DJNZ 00CF	00CF	10 FE	
JP 0000	00D1	C3 00 00	Jump to start and re-load all registers
LD C,10	00D4	0E 10	Load C with 10 for 16 loops of multiplexing '1' and '2'
LD A,06	00D6	3E 06	Load A with 06 to create '1' on RH display.
OUT (02),A	00D8	D3 02	Output this to latch.
LD B,10	00DA	06 10	Create short delay
DJNZ 00DC	00DC	10 FE	Decrement B to zero.
LD A,DB	00DE	3E DB	Load the accumulator with DB to create '2' in LH display
OUT (02),A	00E0	D3 02	Output to latch.
LD B,10	00E2	06 10	Create short delay.
DJNZ 00E4	00E4	10 FE	Decrement B to zero.
DEC C	00E6	0D	Decrement C and if not zero, jump to start of multiplexing the displays.
JR NZ,00D6	00E7	20 ED	
XOR A	00E9	AF	Zero A
OUR (02),A	00EA	D3 02	Output to latch to turn off both displays.
LD B,FF	00EC	06 FF	Load B with FF to produce a short delay for the OFF time.
DJNZ 00EE	00EE	10 FE	The only way of jumping out of 'BLINKING 21' is to push button B or
IN A,(01)	00F0	DB 01	reset the computer. The program inputs from the set of buttons and if
CP 40	00F2	FE 40	B is pressed, the program jumps to 0000. Otherwise the program
JP Z,0000	00F4	CA 00 00	keeps looping.
JR 00D4	00F7	18 DB	

Before we continue our disection of the program for the Microcomp, let us pause for a discussion on a number of related topics. These will help you to understand how a micro system goes together and how it functions.

PROGRAMMING THE 2732.

The 2732 in the Microcomp kit comes ready programmed with a set of experimental programs and only the lower half of the ROM has been filled.

This leaves the upper half vacant, for use in any way you wish.

There are two ways in which the upper half can be filled. One is by using an EPROM programmer and burning the locations yourself. The other is to write the program and have someone else burn the ROM.
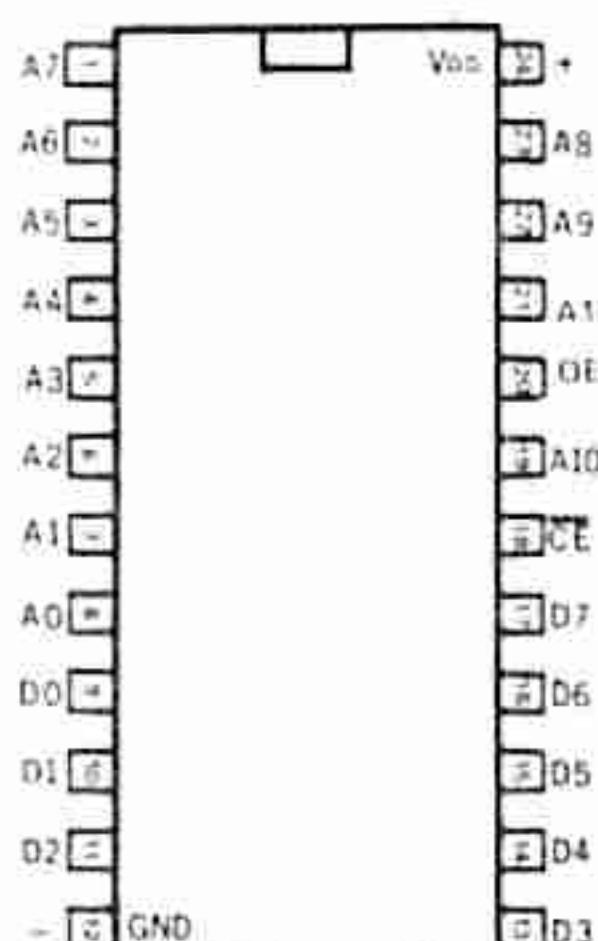
Burning a program is only done after you are thoroughly satisfied with its performance, as it is very difficult (if not impossible) to change the program, once it is burnt. For this reason it is best to get the program up and running via a medium which can be easily altered, as a program quite often has to go through lots of changes and modifications before you are completely satisfied.

The most logical way is to use some form of RAM memory, in which the locations can be altered as many times as you like. The only difficulty with RAM memory is it will lose its contents when the power is switched off. If the RAM is backed up with a battery, the contents will be retained.

This arrangement can then be used to generate programs without the fear of loss, should the computer be turned off.

The program can then be transferred from the programming computer to the Microcomp.

The Microcomp sees each half of a 2732 as a separate 2k block of memory.

2732 PIN-OUT

The program-accessing routine at 0000 must be written for both the lower half and upper half and this will enable you to start at any address, providing it is an even hex value.

Burning can be carried out on the TEC EPROM BURNER and full details of this project can be found in issue 13.

Memory is divided into PAGES and each page consists of 256 bytes. When programming, all address values are written in hexadecimal form and one page contains FF bytes. See P. 16 of issue 11 for the hex table and details on understanding hex notation. A 1k block of memory has 4 pages and a 2k memory chip such as 2716 has 8 pages. A 4k memory chip such as 2732 will hold 16 pages of bytes.

A program can range from only a few bytes to many pages and to give you an idea of the compactness of machine code, the two previous games, TUG O' WAR and BLACK JACK, occipied about 1 page each. Obviously a more complex game with a more complex display (such as a video screen) would require more instructions but one page has the capacity to hold about 100 instructions.

This means a 2k ROM will hold about 8 simple programs.

Programs are not fast to be produced and it may take 10 to 50 hours to create a one-page program. A 2k ROM may take weeks or even months to fill!

Once you are satisfied with the performance of a program, you are ready to burn it into an EPROM.

Before this can be done there are two things you should do.

Firstly you should determine where you are going to place the program. This is importart as it will be in a different location to where it was being created and the absolute address values will not apply.

Often the program is created at address 0000 and all jump instructions relate to this. Any address values which have been defined are called absolute and must be changed when the program is shifted to a new location.

When you have determined the new location, you should BLOCK TRANSFER the program to the same address in the non-volatile RAM, using the following program:

at **0C00:**

```
11 ___ ___   TO: address + 1000H
21 ___ ___   From: address + 1000H
01 ___ ___   No of bytes.
ED B0
C7
```

For example, if you have produced a 148 byte program at 0000 in the non-volatile RAM and need to shift it to 0280, here is the Block Transfer program:

at **0C00:**
```
11 80 12
21 00 10
01 48 01
ED B0
C7
```

At the beginning of the RAM you need a jump routine:

```
06 00
DB 01
21 00 00
6F
29
29
29
29
E9
```

This is entered at 0000 in the non-volatile RAM, which is ADdress 1000 on the TEC (to access the start of the expansion port socket)

Now you must change all the absolute address values (such as the start of a table, a jump instruction etc.)

Change the switch on the non-volatile RAM card to 'ROM' and switch the TEC off. Transfer the non-volatile RAM to the Microcomp and load '28' on the input switches. Turn the comp on and push reset. The program will run.

You should now remove all traces of the lower program so that you are sure the new one is the only one being run. This is done on the TEC by loading **FF** into each location of the old program

The program is now ready for transfer to EPROM. You have confirmed its operation and run it at its new location - nothing more need be done.

Refer to the EPROM BURNER project in issue 13 for the actual transfer procedure.

When you have completed a program and burnt it into EPROM, it should be fully documented by writing it out as shown in our examples.

It is important to use arrows to indicate the jumps and even a block diagram explaining what is happening at various locations.

A description of the program including which buttons are doing what, will also help as it's very easy to forget how the game is played, after a few months.

Give the program a name and fill out the log below to assist in identification.

If you follow these rules you will be able to use parts of the program when creating new ideas and save generating everything afresh.

Sw. Positions:	Address:	Name of Program:

RAM and ROM

RAM is the abbreviation for RANDOM ACCESS MEMORY.

It is tempory storage memory in which data is only retained while the power is applied.

When the power is removed, the contents are lost. This is because data is stored via a flip flop or single MOS transistor and these require power (although very little) for the data to be retained.

There are two forms of Random Access Memory. **STATIC and DYNAMIC.**

Static Memory uses a flip flop for each bit of information and this will hold the HIGH or LOW as long as the power is connected to the chip.

Dynamic Memory uses only a single MOS transistor in which a charge on a substrate indicates the presence of data. Since this charge has the tendency to leak away, it must be replenished every 2 milliseconds. This requires additional circuitry and is inconvenient in a small system; although it is the cheapest way to purchase blocks of memory.

RAM is also called Read/Write memory as it can be written into and read during the process of executing a program.

A micro system which does not have any RAM is called a dedicated system and is limited to running a program contained in ROM memory.

The need for RAM varies enormously with the task. Sometimes you only need a few bytes of RAM to store tempory values and the same locations can be written into again and again.

Othertimes you need a large amount of RAM to store a whole screen of information.

With as little as one page (256 bytes) a system can be designed to perform quite complex tasks as the data can be updated and written-over constantly.

The Z-80 requires only two very small sections of RAM for it to become a 'thinking' computer. These two areas are called SCRATCHPAD and STACK.

The scratchpad or BUFFER zone needs only a few bytes where such data as displays values are kept. This frees registers for carrying out program commands.

The other area is STACK and this is where bytes are loaded (in pairs) so that the contents of a particular register can be saved. The stack is unusual in that it grows downwards as more bytes are added and it is essential to keep removing bytes at the same rate as they are added so that the stack does not grow too large.

The other peculiar feature about the stack is the access you have to its contents. It is a LAST-ON FIRST-OFF arrangement and only the top byte (and the next) is

accessible and this is another reason for keeping the stack manageable.

The main purpose of the STACK is to free registers for other operations and then be able to re-load them with the value that had been saved.

Our Microcomp does not have RAM memory and thus the stack and scratch-pad features are not available.

The alternative to scratch-pad is to use a register pair to hold 2 bytes of data and this has been done in many of the programs. This severly limits programming as the working registers are held-up as memory cells.

Without a stack, programs have to be designed differently and may take more programming steps, but they work just as well.

IX, IY, HL and DE register pairs and also the alternate A, BC, DE and HL registers can be used to get around the storage problem.

Some of the programs for the Microcomp show how the registers have been used in this way.

ROM

ROM is Read Only Memory.

This memory is used to store instructions which do not have to be altered. Data in ROM remains fixed and stable, even when power is removed. It is permanent.

There are different types of ROM memory. One is programmed by the manufacturer and cannot be changed, the other is erasable memory and can be programmed by the client. It can also be erased if the contents are not required, by exposing to ultra violet light for about 15 minutes.

In the Microcomp project, a 2732 EPROM has been used. This is the most economical size for the job and is capable of holding 4k of information. 4k is equivalent to 4096 bytes and would be a very long program if it contained a single program!

If we assume an instruction takes an average of 2 bytes, the program will extend for 2048 lines! A program of this length would take many weeks to produce and the number of things it could do would be quite impressive.

In the Microcomp, the 2732 is accessed in two halves. This is done via a jumper. The lower half contains a range of programs which we are currently investigating and by taking the jumper lead to the lower pin on the PC board, the upper half of the EPROM is accessed.

The upper half is blank and you can fill it with programs of your own. The first 10H bytes must contain a jump routine identical with the lower half to allow you to jump to the start of each program.

In the near future you will be able to send in your EPROM for filling with additional routines. The programs for the 'add-ons'

will be loaded into the upper half and many of these are already finalized. But firstly we want to fully explain the lower half and get you aquainted with the concepts.

One question we have been asked is why the Microcomp has only 11 address lines whereas the 2732 requires 12!

The answer is we are creating the 12th address line via the jumper lead. When the 12th line is LOW, the lower 2k is accessed. When the jumper is HIGH, the upper 2k is accessed. Since this is a manual operation, a program cannot cross the 2k border and routines in the lower half cannot be accessed by those in the upper section. (If you wish to cross the 2k boundary, place the jumper on A11).

Because of our arrangement, the 2732 can be considered as two separate 2k blocks, each of which is equivalent to a 2716 EPROM. In fact you can use 2716's without the need for any modifications.

Each 2k block is addressed in hexadecimal notation. It starts at **0000** and goes to **07FF**. The next 2k starts at **0800** and finishes at **0FFF**. There are 8 pages in 2k and these are: Page 0, 1, 2, 3, 4, 5, 6 and 7. Each page contains **FF** bytes as explained previously.

All address values, data values and Jump Relative values are Hex values and you need to think in HEX notation when writing Machine Code programs.

Using the Microcomp will familiarize you with hex and encourage you to think in this notation.

BASIC vs MACHINE CODE

Everyone has heard much about BASIC. It introduced many of us into the world of microcomputers and it deserves its reputation for being the best language for teaching computers to beginners.

And true enough, Basic has enabled beginners to perform tasks which would have been absolutely impossible otherwise.

But basic isn't the solution to all programming. When you need a simple program for sequencing or timing, you don't need basic. When you need high-speed graphics, you don't use Basic. And when you want to design your own system, you can't include Basic.

In fact you don't use any high level language at all. You use only the codes which the microprocessor understands: and these are called MACHINE CODES.

That's the language or instruction set we are teaching: MACHINE CODE or MACHINE CODE PROGRAMMING.

With Machine Code you can perform all the operations and effects available to the Basic programmer except you have to create them all yourself.

Remember that all the work and skill put into compiling the set of Basic instructions would represent years of effort and we would never be able to attain this level of development via a simple model.

For us, we will have to be satisfied with starting at the beginning and learning some of the simplest forms of programming. Even these will achieve an amazing variety of effects and you will be quite impressed with the results.

We are not rubbishing Basic but let's say it is completely removed from the field we are covering. Machine code is is up to 10,000 times fast and takes up to 500 times less memory. But Most impressive is a Machine code system can be created without any external assistance. You become the master - designing your own system and only requiring a list of Machine code instructions for you to be able to complete anything from a sequencer to a robot.

HOW TO START PROGRAMMING

All programs start with an idea. The idea may be vague at first or you may be lucky enough to know exactly what you want to achieve.

Vague or concrete, the way to start programming is by getting a sheet of paper and jotting down notes.

Start with sketches, scribbles and bits of data.

Put a date on the sheet and think up a name for the project. Names and labels help identify and strengthen your ideas.

These jottings will look feeble when you look back on them, but at the beginning they form the groundwork on which to build. It's the only positive way of getting the facts together.

Put down all you know and all you want to do, then go away and sort it over in your mind.

Your brain can actually put things together much better after you have cleared it first by writing down all the preliminaries.

Don't be afraid to use paper. It will take about 6-10 pages to produce one page of finished work.

At first the best idea is to use parts of existing programs and modify them to suit. Later you can think about creating complete programs of your own.

Lastly, don't be disappointed if the program doesn't work first go. We have trouble with all of ours. They rarely work first time.

But that's the wonderful part about programming. The micro picks up your mistakes and fails to operate.

When this happens, you can spend hours trouble-shooting the fault.

The best advice in this situation is to give the program to a friend aquainted with programming and ask him to check it. A fresh mind is more able to spot a silly mistake.

If you don't have anyone in this category, you will have to work through it yourself.

If the displays fail to light up, you will not know how far through the program the processor has gone.

Start at the beginning and look for the first OUT command. Immediately after this instruction place a HALT command. This will let you know if the micro has travelled this far through the program.

If the display still fails to light up, you will have to investigate each of the steps and instructions very carefully. Work backwards through the program using the DISASSEMBLY codes on the back of issue 12 (and also in Notebook No 3) and make sure you get the same instructions as in the original production of the program.

Next check the JUMP and JUMP RELATIVE values to confirm that the microprocessor is actually landing on the address intended. Read the section on Jump Relative in issue 12 of TE, because these are the trickiest bytes to add to a program. Remember, they are the LAST bytes to be inserted as you need to count the number of bytes between the present address and the address to be jumped to.

★★★★★★★★★★★★
Machine Code programming allows you to create your own system - with pen, paper and op-codes.
★★★★★★★★★★★★

When creating a program, you will not know the value of a displacement byte initially and it is important to put a line in place of the byte thus: _____ so that it can be inserted later. This line lets you know that one byte must be counted when working out the displacement values.

If the display still fails to illuminate, you can create your own display value by loading the accumulator and outputting it to the display and then adding a HALT instruction. This is a last resort! and lets you know how far the program is progressing.

I hope you don't have troubles of this complexity but if so, this will get you out.

Start with simple programs and get your ideas flowing. It's not as difficult as you think to convert ideas into visual effects and its very rewarding to see them running.

When writing a program for the Microcomp, you start at address 0000. This is where the processor naturally starts when the reset button is pressed.

It can then be shifted to a higher location and a jump routine used to access it.

Creating a program which RUNS takes a certain amount of skill. By 'runs' we mean it completes one pass of the program and displays the appropriate information on the displays. After you get it to run you can concentrate on adjusting the values of timing to achieve the most pleasing effects.

But the main problem is getting the program to run and we have already mentioned how to get into the program and force it to display. There are a couple of other points which we forgot to mention and they involve the placing of tables.

Tables should be placed well away from the program so that you don't run out of room. When everything works perfectly, they can be moved up and the pointers changed accordingly.
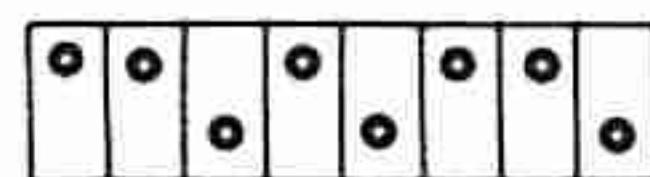
The idea is to get everything into a compact block and relative addressing uses less bytes than absolute addressing. so use it whenever possible. . Also remove any **NOPs** and any holes or spaces. Closing up a program and neatening it up takes time but it makes it much more presentable in the end.

●●●●●●●●●●●●

We will now continue with the programs in the monitor, explaining each and every instruction and how the program is intended to work.

FROM INPUT PORT TO 8 LEDS

This routine is located at 0290 and is addressed by switching the switches ON thus:

This program is very handy for checking the operation of the computer in the early stages. This may be too late for some constructors, but for those with a problem in the displays, it will help locate the fault.

The program checks each line of the input port and outputs it to the displays.

Each time you turn an input switch ON, the corresponding LED, in the row of 8 LEDs, will be illuminated.

If this does not happen, you can trace through the particular line and locate the fault.

The program at **0290** contains 6 bytes. That's all, just 6 bytes! It inputs the data on the input port and loads it into the accumulator. It then outputs it to port 2 to turn on the appropriate LEDs and then jumps back to the start of the program.

This means it is rapidly looping around the program and will update the displays as soon as the input values are changed.

The program can also be used to compare between the row of 8 LEDs, the 7-segment display(s) and the 4x4 matrix.

Experiment by inputting a hex value and see the effect you get on each of the displays.

In this way you can create any effect you want on the 4x4 (within limits).

FROM INPUT PORT TO 8 LEDS

IN A,(01)	0290	DB 01	Looks at input switches and places the value in the accumulator.
OUT (02),A	0292	D3 02	Outputs accumulator to the latch.
JR 0290	0294	18 FA	Jumps to start of program.

From this program you will see:

1. The value of each LED in the row of LEDs corresponds to a switch. The lowest value is 01, then 02, 04, 08, 10, 20, 40, 80, and this can be confirmed by the values written on the PC board.

2. The value of each switch also corresponds to a segment in the 7-segment display. Turn on various switches and see the effect(s).

Prove the following:

Adding '80' to a value will make the display jump to the 10's display. Note that 80 by itself does not turn on ANY display.

Button 'A' is connected to **80** and will make the figures jump from one display to the other.

3. The 4x4 matrix has been wired so that each column is turned on by a LOW value. These values are: 01, 02, 04, and 08. This will cause all the LEDs to come on. Each of the rows can be turned OFF and this is done via the values 10, 20, 40 and 80.

There are some limitations as to what combinations of LEDs can be turned on and this is something you must be aware of.

Experiments:
Create these effects by using the input switches:

(a) (b) (c) (d)

Create these effects on the 4x4 matrix:

★ = 'ON'

Create these on the 7-segments displays:

INCREMENT via BUTTON A

This program at 02A0 increments the display each time button A is pressed.

LD A,00	02A0	3E 00	Load the accumulator with zero.
LD C,A	02A2	4F	Load zero into C.
IN A,(01)	02A3	DB 01	Input the value on the switches to the accumulator.
BIT 7,A	02A5	CB 7F	Test BIT 7 of the accumulator to see if button A is pushed.
JR Z 02A3	02A7	28 FA	Jump to 2A3 if NOT pressed. Go to 2A9 when pressed
LD A,C	02A9	79	Load C into the accumulator.
INC A	02AA	3C	Increment the accumulator.
LD C,A	02AB	4F	Load the answer into the TALLY register 'C'.
OUT (02),A	02AC	D3 02	Output the accumulator to the displays.
IN A,(01)	02AE	DB 01	Input the switches to the accumulator.
BIT 7,A	02B0	CB 7F	Test BIT 7.
JR NZ 02AE	02B2	20 FA	Jump to 2AE if A is pressed. Go to 2B4 when released.
JR 02A3	02B4	18 ED	Jump to 2A3.

This will enable you to see the effects on the display without having to manually input values via the switches.

The accumulator is required for two functions. It outputs the value of the count and then looks to see if a switch is pressed. That's why we need another register to hold the value of the count, so that the accumulator can be loaded with other information. Thus the C register has been used for temporary storage.

The program contains two small loops and the micro is constantly executing the top one when button A is not pressed and the lower one when the button is pressed. The micro jumps from one loop to the other during the time when the button is travelling from one state to the other.

```
3E 00
4F
DB 01 ◄────┐
CB 7F      │
28 FA ►────┤
79
3C
4F
D3 02
DB 01 ◄────┐
CB 7F      │
20 FA ─────┘
18 ED ►
```

This is a very simple way of creating a debounce condition and prevents more than one count being registered on each press of the button.

AUTO INCREMENT (fast)

This program is located at **02C0** and lets you sit back and watch the displays

increment automatically. You will be interested to know that the program takes 256 steps before it repeats!

Compare the effect on the row of 8 LEDs with the 4x4 and seven segment displays.

Notice that they produce entirely different effects due to the placement of the LEDs and this can be remembered when designing displays for advertising etc.

LD A,00	02C0	3E 00
INC A	02C2	3C
OUT (02),A	02C3	D3 02
DJNZ 02C5	02C5	10 FE
DJNZ 02C7	02C7	10 FE
DJNZ 02C9	02C9	10 FE
JR 02C2	02CB	18 F5

The first instruction loads the accumulator with zero. You will notice this address is not used again by the program. Thus we call it a START-UP value. The accumulator is then incremented on each pass of the program and the value outputted to the latch. The next three instructions are **DJNZ's** in which the B register is decremented to zero during each instruction. After the 3 DJNZ's the program jumps to **02C2** and outputs the next higher value.

AUTO INCREMENT (variable)

This program is located at 02D0 and the speed with which the computer

 02D0

completes one cycle depends on the setting of the input switches.

LD D,01	02D0	16 01	Load the TALLY register with 01.
IN A,(01)	02D2	DB 01	Input the switch value to the accumulator.
LD C,A	02D4	4F	Load the accumulator into 'C' for the delay value.
LD A,D	02D5	7A	Load the TALLY into the accumulator.
OUT (02),A	02D6	D3 02	Output the tally value to the displays.
DEC C	02D8	0D	Decrement register C.
JR NZ 02D8	02D9	20 FD	Jump to 02D8 if register C is not zero.
INC D	02DB	14	Increment the tally register.
JR 02D2	02DC	18 F4	Jump to the start of the program.

'D' is the tally register and holds the value to be displayed on the screen, so that the accumulator can be used for other things.

'C' is the delay register and it is decremented very similar to a **DJNZ** statement, where FF produces the longest delay and 01 the shortest delay.

This is not quite correct, however, as you will find out for yourself.
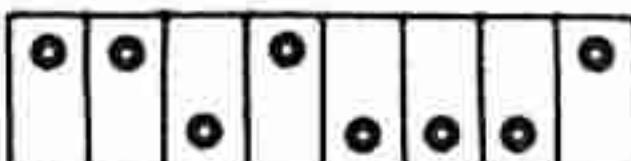
Load the value **01** and compare it with **00**. **00** is a much longer delay and it appears to be as long as **FF**! In fact this is the case! The longest delay is produced when a register is loaded with **00** since the first operation to be performed on the register is to decrement it. The result is **FF** and that's why it takes **FF** loops to bring it to zero.

The program is designed to start with an output value of 01 and increment automatically to FF. The ON time (the delay time) is adjustable via the setting on the input switches.

Note: We don't have any control over the values appearing on the screen, just the speed of the increment.

AUTO DECREMENT

By changing one byte of the program at 02C0, we produce a decrementing
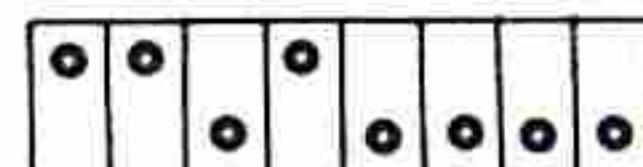
 02E0

counter. The best effect of decrementing can be seen on the 8 LEDs. Adjust speed control to view the effect in slow motion.

AUTO DECREMENT

LD A,00	02E0	3E 00	Load the accumulator with zero.
DEC A	02E2	3D	Decrement the accumulator.
OUT (02),A	02E3	D3 02	Output the accumulator to the latch.
DJNZ 02E5	02E5	10 FE	Decrement register 'B', FF loops.
DJNZ 02E7	02E7	10 FE	" " " "
DJNZ 02E9	02E9	10 FE	" " " "
JR 02E2	02EB	18 F5	Jump to start of program.

AUTO DECREMENT (variable)
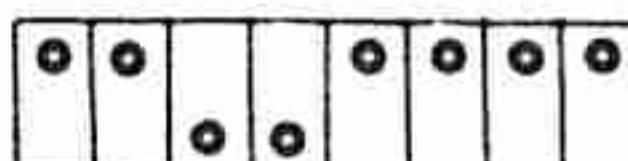
This routine is located at 02F0 and decrementes the display when button A is pressed. It has a fixed rate of decrementing and is not variable.

 02F0

LD E,FF	02F0	1E FF	Load the COUNT HOLD register with **FF**.
LD A,E	02F2	7B	Load the Count Hold register into the accumulator.
OUT (02),A	02F3	D3 02	Output the accumulator to the latch.
DJNZ 02F5	02F5	10 FE	Create a short delay with the B register.
IN A,(01)	02F7	DB 01	Input the bank of switches to the accumulator.
Bit 7,A	02F9	CB 7F	Test bit 7 of the accumulator to see if A is pressed.
JR Z,02F2	02FB	28 F5	Jump to 02F2 if it is not pressed. Go to next line if pressed.
DEC E	02FD	1D	Decrement register E.
JR 02F2	02FE	18 F2	Jump to 02F2.

4x4 DISPLAY

As the name suggests, the program at **0300** is designed for the 4x4 DISPLAY. It

 0300
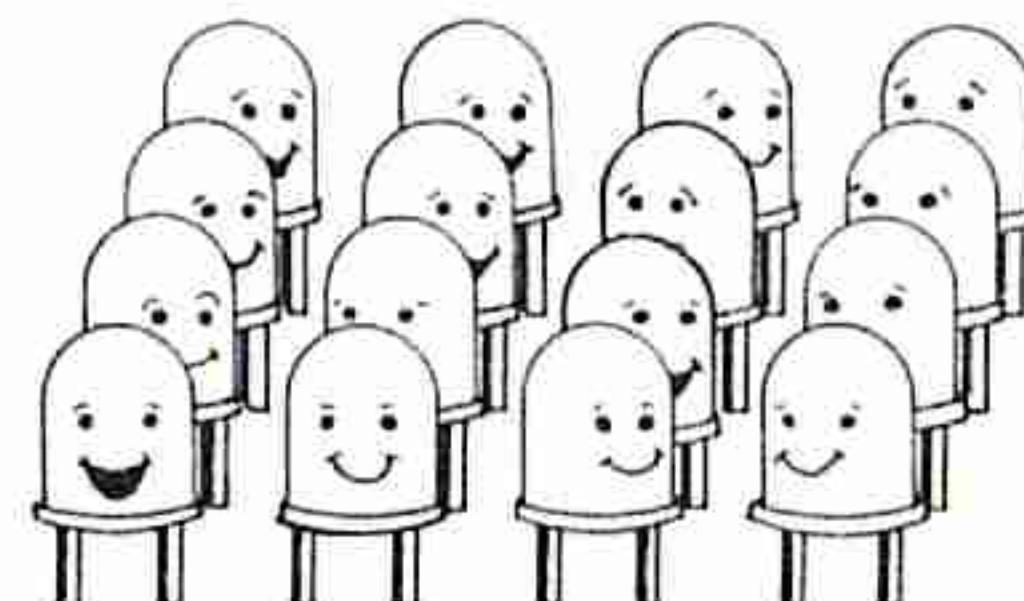
will produce almost no interpretable effects on either of the other displays.

The routine we have presented is only just the start of what you can do with a set of LEDs in an array. Our 4x4 can be multiplied-up many times to produce an enormous array of LEDs or globes and obviously the ultimate is to produce a video screen with coloured globes to duplicate a TV. But the cost of this kind of venture is enormous as the parts alone would cost a fortune and the time taken to wire it up would be too much for an individual constructor.

That's why we have concentrated on a manageable module.

One of the decisions you have to make when outputting to LEDs, is the method of turning them ON. One is to connect each output of a latch directly to a LED. The other is to multiplex the display and scan it. The multiplex method uses the least number of chips and is obviously the cheaper.

The relative merits of each will be covered in future articles and for the moment we will study the effects which can be produced with a display connected in MULTIPLEX mode.

The program at **0300** is an OUTPUT ROUTINE in which a value is loaded from a table into the accumulator and outputted to the display. The display remains illuminated for a delay period and then the next byte is picked up from the table. This is done until all the bytes have been used.

When the end of the table is reached, the program starts again. This is repeated for 8 loops and then the micro advances to the second part. This is identical to the first except for the byte table. It has entirely different values and the effect is completely different. At the conclusion of the second byte-table, the micro jumps back to the start of the program and the first pattern is outputted.

The speed of presenting a pattern is controlled by the clock and the inbuilt delay value. The delay is fixed but the clock can be adjusted to slow-down or speed-up the effect.

Mnemonic	Address	Hex	Comment
LD B,08	0300	06 08	B is the COUNT REGISTER for the number of loops in the first program.
LD HL,0338	0302	21 38 03	Load HL with the address of the start of the BYTE TABLE.
LD C,18	0305	0E 18	Load C with the number of bytes for the program (There are 24 bytes.)
DEC C	0307	0D	Decrement the number of bytes remaining in the table to detect the end of table.
JR Z,0318	0308	28 0E	If no bytes remain, decrement the number of loops and start program again.
LD A,(HL)	030A	7E	Load the accumulator with the byte pointed to b the HL register pair.
OUT (02),A	030B	D3 02	Output this value to port 2.
INC HL	030D	23	Increment HL to point to the next byte in the table.
LD DE,0080	030E	11 80 00	Load DE with a short delay value.
DEC DE	0311	1B	Decrement DE.
LD A,D	0312	7A	Load D into A.
OR E	0313	B3	Logically OR the accumulator with E to see when BOTH D and E are zero.
JR NZ,0311	0314	20 FB	Jump to 0311 if the answer is NOT ZERO.
JR 0307	0316	18 EF	Jump to DEC C and repeat for the second byte in the table.
DJNZ 0302	0318	10 E8	Decrement the number of loops and start the byte table again.
LD B,08	031A	06 08	Load B with 8 for the second part of the program.
LD HL,0350	031C	21 50 03	
LD C,20	031F	0E 20	
DEC C	0321	0D	
JR Z,0332	0322	28 0E	
LD A,(HL)	0324	7E	
OUT (02),A	0325	D3 02	
INC HL	0327	23	
LD DE,0080	0328	11 80 00	
DEC DE	032B	1B	
LD A,D	032C	7A	
OR E	032D	B3	
JR NZ, 032B	032E	20 FB	
JR 0321	0330	18 EF	
DJNZ 031C	0332	10 E8	
JR 0300	0334	18 CA	

This part is identical with that above except the byte table is longer and located at a different address. When 8 loops of this part have been executed the program jumps to the top program and the cycle repeats.

At 0338:		At 0350:		
01	CF	0F	B8	D4
02	3F	FF	D8	D2
04	CF	0F	E8	B2
08	3F	FF	E4	B4
EF	96	0F	E2	D4
DF	FF	FF	E1	D2
BF	96	0F	D1	B2
7F	FF	FF	B1	B4
03	33	71	71	
0C	CC	72	72	
03	C3	74	74	
0C	3C	78	B4	

An almost unlimited number of patterns and effects can be produced on the 4x4. However not every combination can be displayed due to the limitations of how the LEDs are accessed.

This means you will have to learn how to access the LEDs and get the patterns you want.
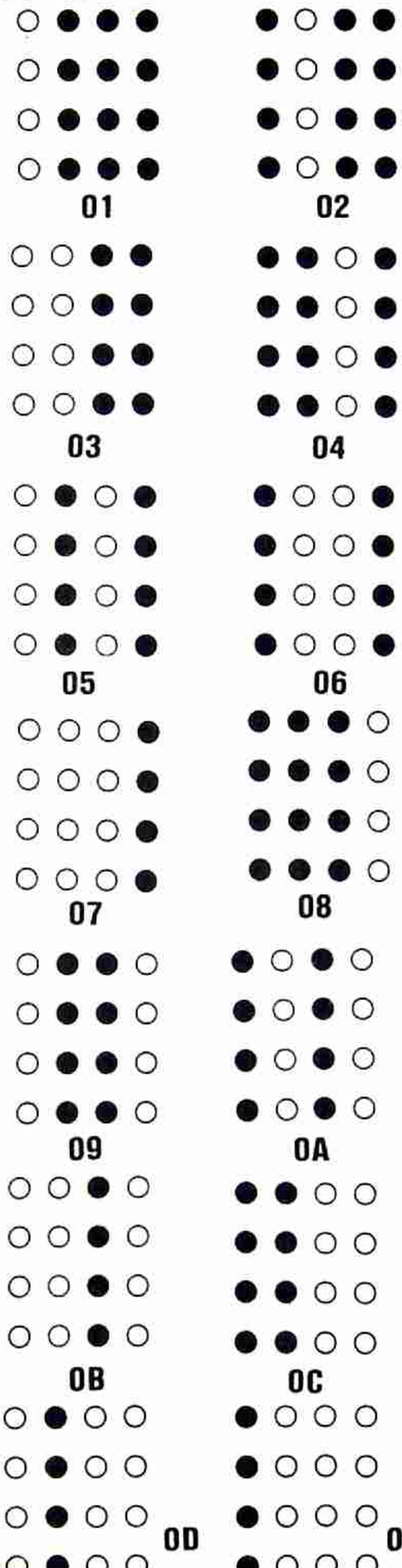
To turn on a LED, the cathode end must be taken to earth and the anode to positive.

This is the hex value required to illuminate an individual LED:

(71) (72) (74) (78)

(B1) (B2) (B4) (B8)

(D1) (D2) (D4) (D8)

(E1) (E2) (E4) (E8)

To access the LEDs we have separated the output latch into two halves with the 4 lower bits connected to the anodes and the 4 upper bits to the cathodes.

The following diagrams give you the values required to turn ON one or more LEDs:

ALL ON "0F"
ALL OFF "FF" or "00".

LEGEND:

○ = ON.

● = OFF.

0F

1F

2F

3F

4F

5F

6F

7F

8F

9F

AF

BF

CF

DF

EF

FF

If you don't want all the LEDs in a row to be illuminated, refer to the diagrams on this page for the hex value needed to illuminate an individual column or column(s).

To use these values select from the first 16 diagrams to give the row(s) and from the following 16 diagrams for the column(s).

01 **02**
03 **04**
05 **06**
07 **08**
09 **0A**
0B **0C**
0D **0E**

OF
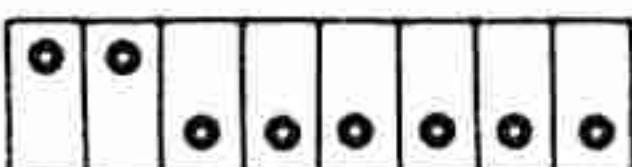
When the two diagrams are placed on top of each other, the LEDs that are common to both, will be illuminated. Due to the sinking and sourcing limitations of the output latch, all the LEDs in the 4x4 can not be illuminated at the same time.

Brightness can be improved by turning off the 7-segment display by shorting the base and emitter leads of the driver transistor together with a jumper lead. This transistor is directly below the second display and is the middle transistor.

VERY LONG DELAY

This routine, at **03F0**, is particularly unusual. Not only is it a very long duration delay but is shows that a program can be split up and placed in two different parts of memory, and still run.

And this is what we have done.

Half the program is located at **03F0** and the other half at **045A**. This makes the Micro jump up and down in ROM as it executes the program.

The jumping back and forth does not occupy many clock cycles but it does increase the overall time by about 5%.

We calculated the time delay to be so long that you may never see the display increment! This is due to the low clock speed. At 70kHz, the Z-80 is operating far below its normal rate and a delay like this introduces many millions upon millions of clock cycles.

WHY DO WE NEED DELAYS?

Delays are very important in micro programs. Due to the high speed of the execution of machine code instructions, some parts of the program must be slowed down so that humans can be involved. This may be for the video aspect, so that the eye can see what is being outputted on a display or for the audio side, so that we can detect tones and beeps.

Delays are also needed to give a SUSPENSE EFFECT for games of chance or strategy to give the impression that the computer is taking time to think.

Or for a video game, to create rates-of-movement for objects moving across the screen.

The delays we are talking about are PROGRAM DELAYS or SOFTWARE DELAYS. They are produced when the micro 'wastes time'. The simplest way of wasting time is to fill a register pair with a large number and gradually decrement it to zero.

By decrementing a single register, the maximum number of loops which can be executed is 256. Each loop may take 20 clock cycles and at the normal running frequency of a system (about 1MHz), the delay time will be very short. By using a REGISTER PAIR, the time can be increased 256 times. The delay becomes more noticeable and will be about 2 seconds.

If we require longer delays we can add another register-pair and increase the delay to more than 131,000 seconds!

When the system is operating at only 70kHz, the delay time turns into hours, days and months!

There is one point to note here: When a micro is performing a very long delay, the entire computer time is being taken up with a COUNT DOWN sequence and this means the micro will not be updating information on the displays or looking at the input port.

If you require other operations to be attended to, they must be included in the loop, as can be seen in the clock program at **0630**.

Instruction	Address	Code	Comment
LD A,01	03F0	3E 01	Segment 'A' will illuminate after a delay period.
LD I,A	03F2	ED 47	Save the 'TALLY' in the I register (Not part of IX).
LD DE,FFFF	03F4	11 FF FF	Load DE with the maximum value.
LD HL,FFFF	03F7	21 FF FF	Load HL with the maximum value.
DEC HL	03FA	2B	Decrement HL.
LD A,H	03FB	7C	Load register H into the accumulator.
OR L	03FC	B5	Logically OR the accumulator with L.
JP 045A	03FD	C3 5A 04	Jump to address **045A**.
JP NZ,03FA	045A	C2 FA 03	If register H and L are not zero, jump to **03FA**.
DEC DE	045D	1B	When HL (the inner loop) is zero, decrement DE.
LD A,D	045E	7A	Load register D into the accumulator.
OR E	045F	B3	Logically OR the accumulator with register E.
JP NZ,03F7	0460	C2 F7 03	If result is not zero, JUMP to **03F7** and DEC HL!
LD A,I	0463	ED 57	When both HL and DE are zero, time is UP!
OUT (02),A	0465	D3 02	Load the TALLY register into A and output it.
INC A	0467	3C	Increment A.
LD I,A	0468	ED 47	Load the new tally into the TALLY register.
JP 03F4	046A	C3 F4 03	Load the register pairs and start again!

When we use two register pairs to create a very long time delay, we do not place one pair after the other as this would only double the time delay. We place them ON TOP of each other so that the effect is MULTIPLICATION. This means one pair is INSIDE the other and we say it is HIDDEN or NESTED. This arrangement gives rise to the term NESTED LOOP. This is what we are creating in this section.

The simplest method of increasing the delay is to add the instruction: 10 FE. This will have the effect of adding 256 cycles to the delay time. This is a **DJNZ** instruction and operates with the B register. The advantage of a **DJNZ** is it does not affect the accumulator. In the Microcomp we do not have any RAM and we cannot save the accumulator via a PUSH operation since we do not have any STACK. Thus it's an advantage not to alter the contents of the accumulator.
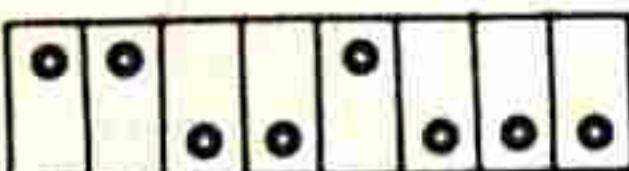
DJNZ loops are not nested loops but are additive and require the **B** register to be zero at the start of the delay routine to create the longest delay. At the end of a **DJNZ** the B register is zero and this is ideal for the next **DJNZ**.

DJNZ's can be grouped thus:

DJNZ FE	10 FE
DJNZ FE	10 FE
DJNZ FE	10 FE
DJNZ FE	10 FE
DJNZ FE	10 FE

0 - 9 COUNTER

The first counter we are going to study is a 0-9 UP COUNTER. This is located at address **0370** and will show us how to

output numbers onto the display and how the INCrement operation is performed.

The main fact to remember with the program is the computer is NOT adding numbers. It is simply going through a table of values and it is the values it fetches that create the increments on the screen.

The table could be designed to produce letters or symbols and we would lose the effect of incrementing.

The requirements of a counter are these:

The computer must detect when a button is pressed and distinguish it from other buttons. In our design button A corresponds to BIT 7 and button B to BIT 6, of the accumulator.

The program must be running or LOOPING at all times ready to instantly pick up an input value.
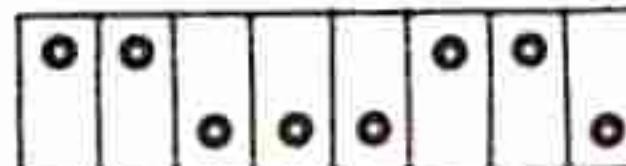
Because the program is running at high speed, we must include a DEBOUNCE feature to prevent more than ONE COUNT being registered when a button is pressed.

With these facts in mind, we have produced the 0-9 COUNTER.

The program contains 2 loops. One is executed when button 'A' is NOT pressed and the other when the button is PRESSED. We also have to detect when the end of the BYTE TABLE is reached.

0 - F COUNTER

This routine, at **0390**, increments the display each time button A is pressed.
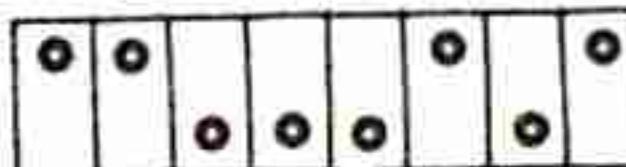
The main program for producing the letters on the display is located at **03A8** and the micro jumps to this address via the instruction JR 03A8. The main program is also used by the A-Z, 0-F counter and shows how the same table and output program can be accessed by two different START-UP ROUTINES.

LD C,10	0390	0E 10
LD DE,03DF	0392	11 DF 03
LD HL,0390	0395	21 90 03
JR 03A8	0398	18 0E

A - Z, 0 - F COUNTER

This counter is located at **03A0** and

produces the letters **A - F** and hex values **0 - F** on the display via button 'A'.

LD C,2A	03A0	0E 2A
LD DE,03C5	03A2	11 C5 03
LD HL,03A0	03A5	21 A0 03
IN A,(01)	03A8	DB 01
BIT 7,A	03AA	CB 7F
JR Z,03A8	03AC	28 FA
INC DE	03AE	13
LD A,(DE)	03AF	1A
OUT (02),A	03B0	D3 02
IN A,(01)	03B2	DB 01
BIT 7,A	03B4	CB 7F
JR NZ,03B2	03B6	20 FA
DEC C	03B8	0D
JR Z,03BD	03B9	28 02
JR 03A8	03BB	18 EB
JP (HL)	03BD	E9

The 3 counters in this section use the table at **03C6**. The 0-9 counter uses only those bytes corresponding to 0-9. The 0-F counter uses bytes from 0 to the end of the table.

The A-Z, 0-F counter uses all the table.
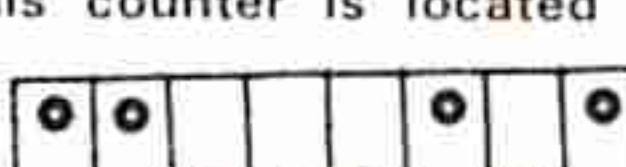
In addition, the 0-F counter uses most of the A-Z, 0-F program and that's why it has only 4 instructions.

At 03C6:

A	77	V	1C
B	7C	W	4E
C	39	X	4C
D	5E	Y	6E
E	79	Z	1B
F	71	0	3F
G	3D	1	06
H	76	2	5B
I	06	3	4F
J	1E	4	66
K	72	5	6D
L	38	6	7D
M	47	7	07
N	37	8	7F
O	3F	9	67
P	73	A	77
Q	67	B	7C
R	33	C	39
S	6D	D	5E
T	78	E	79
U	3E	F	71

By now you will be aware that certain combinations of hex values produce letters and numbers on the display.

Use the program at **0290** to produce the numbers 0 - 9 and letters A - F, by switching ON the correct switches. Use the output display values on P68 to assist you in this. Add the value on the switches and compare with the table at **03C6**.

---

0 - 9 COUNTER

LD C,0A	0370	0E 0A	Register C is the counter for the BYTE TABLE. There are ten bytes.
LD DE,03DF	0372	11 DF 03	The DE register pair is loaded with the start-address of the byte table.
IN A,(01)	0375	DB 01	The input latch is looked at and the value it holds is placed into the accumulator.
BIT 7,A	0377	CB 7F	The only line (or BIT) which is tested is bit 7. This is the 8th line and is button A.
JR Z,0375	0379	28 FA	If it is HIGH (or SET) the program advances. If it is LOW (or RESET), it goes to: IN A,(01)
INC DE	037B	13	INCrement the DE register pair to look at address **03E0**.
LD A,(DE)	037C	1A	The byte at **03E0** is placed in the accumulator.
OUT (02),A	037D	D3 02	Output this byte to the display.
IN A,(01)	037F	DB 01	Look at the input port.
BIT 7,A	0381	CB 7F	Test bit 7 of the accumulator.
JR NZ,037F	0383	20 FA	Jump to address **037F** if button A is pressed. When button is released, advance to next line.
DEC C	0385	0D	Decrement the BYTE COUNT register.
JR Z,0370	0386	28 E8	If end of table is reached, JUMP to start of program. If not reached, go to **0375**.
JR 0375	0388	18 EB	

00 - 99 COUNTER

Counters and counting are a very important part of electronics. Business and industry needs counting. Whether it be to keep track of money or components, it needs to know the answers.

The counter program at **0400** shows the basics of how a counter operates and how the COUNT VALUE can be held in a single register pair.

Functions such as INCREMENT, DECREMENT and RESET can also be included. The most involved part of the program is debouncing the switches, to prevent the count automatically incrementing if the button is kept pressed.

0400

at 03E0:

```
3F
06
5B
4F
66
6D
7D
07
7F
67
```

The program basically consists of two loops. The top loop is executed when the buttons are NOT pressed and the lower when either of the buttons is pressed.
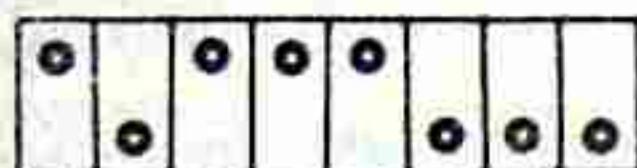
This is necessary to keep the displays illuminated while at the same time preventing the program from incrementing the displays if a button is kept pressed.

Instruction	Address	Code	Description
LD E,00	0400	1E 00	Register E holds the present COUNT VALUE in decimal form.
LD A,E	0402	7B	Load E into the accumulator so that it can be operated upon.
AND 0F	0403	E6 0F	Mask off the 4 HIGH ORDER bits. In other words, remove them.
LD HL,03E0	0405	21 E0 03	Load HL with the start of the BYTE TABLE that produces the display numbers.
ADD A,L	0408	85	Add the start of the byte table to the accumulator.
LD L,A	0409	6F	Load the accumulator into L to produce a new pointer value.
LD A,(HL)	040A	7E	Load the accumulator with the byte pointed to by the HL register pair.
OUT (02),A	040B	D3 02	Output this value to port 2.
LD A,E	040D	7B	Load E into the accumulator again, this time to produce the 10's value.
RRA	040E	1F	Shift the bits in the accumulator one place to the right.
RRA	040F	1F	Shift the bits in the accumulator another place to the right.
RRA	0410	1F	" " " " " " " "
RRA	0411	1F	" " " " " " " "
AND 0F	0412	E6 0F	Mask the 4 HIGH ORDER bits so that they are effectively removed.
LD HL,03E0	0414	21 E0 03	Load HL with the start of the byte table.
ADD A,L	0417	85	Add the value of L to the value in the accumulator.
LD L,A	0418	6F	A new pointer value is created.
LD A,(HL)	0419	7E	Load the accumulator with the byte pointed to by the HL register pair.
SET 7,A	041A	CB FF	SET bit 7 of the accumulator to '1' to turn on the 10's display.
OUT (02),A	041C	D3 02	Output the value of the accumulator to the latch.
IN A,(01)	041E	DB 01	Input the value on the switches to the accumulator.
BIT 7,A	0420	CB 7F	TEST bit 7 to see if button A is pressed.
JR Z,042A	0422	28 06	If it is zero, jump to **024A**. If it pressed, increment to next instruction.
LD A,E	0424	7B	Load E into the accumulator, ready for an INCrement operation.
INC A	0425	3C	Increment the accumulator.
DAA	0426	27	Decimal adjust the accumulator. This means an A will be changed into 10.
LD E,A	0427	5F	Save the new count value by loading it into the E register.
JR 0432	0428	18 08	Jump to **0431**.
BIT 6,A	042A	CB 77	From **0422**, the program jumps to this address and tests for button B.
JR Z,0402	042C	28 D4	If not pressed, the program jumps to **0402**. If pressed, the program increments.
LD A,E	042E	7B	Load the COUNT REGISTER into the accumulator.
DEC A	042F	3D	Decrement the accumulator.
DAA	0430	27	Decimal adjust the accumulator. This will change a zero into a 9.
LD E,A	0431	5F	Save the count value by loading it into the E register.
LD A,E	0432	7B	
AND 0F	0433	E6 0F	
LD HL,03E0	0435	21 E0 03	
ADD A,L	0438	85	
LD L,A	0439	6F	
LD A,(HL)	043A	7E	
OUT (02),A	043B	D3 02	
LD A,E	043D	7B	
RRA	043E	1F	
RRA	043F	1F	
RRA	0440	1F	
RRA	0441	1F	
AND 0F	0442	E6 0F	The remainder of the program keeps both displays illuminated by looping from **0432** to **0456** while either of the buttons remains pressed. As soon as the button is released, the program jumps back to **0402** and executes the top loop.
LD HL,03E0	0444	21 E0 03	
ADD A,L	0447	85	
LD L,A	0448	6F	
LD A,(HL)	0449	7E	
SET 7,A	044A	CB FF	
OUT (02),A	044C	D3 02	
IN A,(01)	044E	DB 01	
BIT 7,A	0450	CB 7F	
JR NZ,0432	0452	20 DE	
BIT 6,A	0454	CB 77	
JR NZ,0432	0456	20 DA	
JR 0402	0458	18 A8	

DICE

The DICE Program at **0470** introduces a few more programming skills.

The first of these is a RANDOM NUMBER GENERATOR. Random numbers are almost impossible to generate via a computer due to it being a very predictable machine. The only reliable way to get a random number is to introduce the human element.

This is what we have done in this program.

At the start of the program a running LED routine moves a single LED around the 4x4 matrix. The ON time for each LED is created by a delay routine that uses the B abd C registers. The C register is loaded with 6 and decrements to zero. Each time this is done, the B register is decremented and when it reaches zero, the LED jumps to the next location.

The random number is generated in the C register and we can exit from the program with a value remaining in C. Since C is the inside loop of the delay it is decrementing very fast and it is not possible to predict what value C will contain.

If it were the outside loop it would be a different matter. Players would gradually get to understand that pressing at the beginning of cycle would generate a low number and at the end of a cycle, a high number.

Owing to the unpredictability of the human reaction, an even spread of numbers from 1 to 6 is created with our routine.

The second feature of the program is the COMPARE and BRANCH.

After the random number has been obtained, a number of flashes are created on the screen and then the accumulator is compared with the random number before jumping to the display routine.

This routine is a very simple multiplexing routine in which three bytes are outputted for a period of 80 cycles.

The program then detects that the input button has been released and jumps to the start of the program.

If a button-check was not made, the same number would appear on the displays due to a constant number of cycles occuring in the program for each game.

At 04D3:

71	E1
72	E4
74	E2
78	E1
B8	D1
D8	B1

Mnemonic	Address	Opcode	Comment
LD D,0C	0470	16 0C	C is the byte table counter for the 4x4
LD HL,04D3	0472	21 D3 04	HL will point to the byte table address
LD A,(HL)	0475	7E	A is loaded with the value of the first byte in the table.
OUT (02),A	0476	D3 02	The accumulator is outputted to port 02.
INC HL	0478	23	The byte table pointer is incremented.
LD B,15	0479	06 15	Load B with 15, for a delay value of 21 loops.
LD C,06	047B	0E 06	Load C with 6, for the dice values: 0-6.
IN A,(01)	047D	DB 01	Input from the input port to the accumulator.
BIT 7,A	047F	CB 7F	Check to see if button A has been pressed.
JR NZ,048D	0481	20 0A	If pressed, jump out of the delay routine.
DEC C	0483	0D	Decrement register C.
JR NZ,047D	0484	20 F7	If C is not zero, jump up. If C zero, advance.
DJNZ 047B	0486	10 F3	Decrement B and if not zero, jump up.
DEC D	0488	15	Decrement the byte table register D.
JR Z,0470	0489	28 E5	When D is zero, jump to start of program.
JR 0475	048B	18 E8	If not zero, continue DELAY ROUTINE.
LD D,06	048D	16 06	Load D with 6 for six flashes of the display.
LD A,0F	048F	3E 0F	Load A to turn on the whole 4x4 display.
OUT (02),A	0491	D3 02	Output to port 02.
DJNZ 0493	0493	10 FE	Register B is decremented to create a delay.
LD A,FF	0495	3E FF	Load A with a value to turn 4x4 OFF.
OUT (02),A	0497	D3 02	Output to port 02.
DJNZ 0499	0499	10 FE	Create a short delay with register B.
DEC D	049B	15	Decrement the flash-count register
JR NZ,048F	049C	20 F1	Loop for 6 flashes.
LD D,80	049E	16 80	Load D for 80 loops for multiplexing routine.
LD A,C	04A0	79	Load our random number into the accumulator.
LD HL,04E0	04A1	21 E0 04	Load HL with address of table for multiplex routine.
CP 01	04A4	FE 01	Compare the accumulator with 1.
JP Z,045F	04A6	CA F5 04	If the accumulator is 1, jump to multiplex routine.
LD HL,04E3	04A9	21 E3 04	Load HL with start address for displaying '2'.
CP 02	04AC	FE 02	Compare the accumulator with 2.
JP Z,045F	04AE	CA F5 04	If accumulator is 2, jump to multiplex routine.
LD HL,04E6	04B1	21 E6 04	Load HL with start-address for displaying '3'.
CP 03	04B4	FE 03	Compare accumulator with 3.
JP Z, 045F	04B6	CA F5 04	If accumulator is 3, jump to multiplex routine.
LD HL,04E9	04B9	21 E9 04	Load HL with start-address for displaying '4'.
CP 04	04BC	FE 04	Compare the accumulator with 4.
JP Z,045F	04BE	CA F5 04	If accumulator is 4, jump to multiplex routine.
LD HL,04EC	04C1	21 EC 04	Load HL with start-address for displaying '5'.
CP 05	04C4	FE 05	Compare accumulator with 5.
JP Z,04F5	04C6	CA F5 04	If accumulator is 5, jump to multiplex routine.
LD HL,04EF	04C9	21 EF 04	Load HL with start-address for displaying 6.
CP 06	04CC	FE 06	Compare accumulator with 6.
JP Z,04F5	04CE	CA F5 04	Jump to multiplex routine is accum is 6.

A jump value must be found and the micro jumps to the multiplexing routine below and produces a display on the 4x4 that is similar to the spots on the face of a dice The routine runs for 80 loops , makes sure button A is not pressed, then jumps to the start of the DICE program.

At 04E0:

B4	D2	72	52	52	52
00	00	B4	00	B4	54
00	78	D8	58	58	58

Mnemonic	Address	Opcode	Comment
LD A,(HL)	04F5	7E	Load A with the value pointed to by HL.
OUT (02),A	04F6	D3 02	Output the value to port 02.
LD B,0A	04F8	06 0A	Load B with a short delay value.
DJNZ 04FA	04FA	10 FE	Create a short delay with register B.
INC HL	04FC	23	Point to next display address
LD A,(HL)	04FD	7E	Load the value pointed to by HL into A.
OUT (02),A	04FE	D3 02	Output to port 02.
LD B,0A	0500	06 0A	Load B with a short delay value.
DJNZ 0502	0502	10 FE	Create a short delay with register B.
INC HL	0504	23	Inc HL to look at next address.
LD A,(HL)	0505	7E	Load value pointed to by HL in the accumulator.
OUT (02),A	0506	D3 02	Output to port 02.
LD B,0A	0508	06 0A	Load register B with a short delay value.
DJNZ 050A	050A	10 FF	Decrement register B to zero.
DEC HL	050C	2B	Dec HL to look at start of display table.
DEC HL	050D	2B	
DEC D	050E	15	Decrement multiplex routine loop counter.
JR NZ, 04F5	050F	20 E4	Loop again if D is not zero.
XOR A	0511	AF	Zero the accumulator and output to port 02 to
OUT (02),A	0512	D3 02	blank the display.
IN A,(01)	0514	DB 01	Look at the output port to see if button A is NOT
BIT 7,A	0516	CB 7F	pressed before re-starting the DICE program.
JR NZ,0514	0518	20 FA	Loop is A is pressed.
JP 0470	051A	C3 70 04	Jump to start of DICE program.

NEG . . . . . Each bit in the accumulator is reversed sign. One's go to zero and zero's go to one. Then one is added to the result.

NOP . . . . . The NO OPERATION instruction. Only the Program Counter advances.

OR ( ) . . . The logic OR operation is performed between the accumulator and the contents of the memory location pointed to by the address in ( ).

OR A,B,C etc A logic OR operation is performed between the accumulator and a specified register.

OTDR . . . . Data from the memory location specified by the contents of the HL register is outputted to a port as specified by the contents of register C. The HL pointer has its value decremented after each transfer operation. The value of register B is decremented and if the result is zero, the Program Counter register is set back 2 units so that the instruction is re-executed.

OTIR . . . . Same OTDR except that the HL pointer is incremented after each execution.

OUT (C),A etc The contents of A, B, C, D, etc are outputted to the port specified by the contents of register C.

OUT port,A . .The contents of the accumulator is outputted to the port specified.

OUTD . . . . Data is outputted from memory location specified by the contents of the HL register pair to the port specified by the contents of register C. (contents of register B will be decremented but no repeat operation will be performed.) HL register pair has its contents decremented after the conclusion of the operation.

OUTI . . . . .Same as OUTD except HL has its contents incremented at the conclusion of the operation.

POP AF . . . Two bytes are removed from the stack. The first byte is loaded into F and the second into A.

PUSH HL . . .Two bytes are placed onto the stack. The contents of the HIGH ORDER register are stored in the stack at the address of the stack pointer less one. The content of the LOW ORDER register are stored at the address of the stack pointer less two.

RES 0,( ) . . RESET BIT 0, 1, 2, 3, 4, 5, 6, 7 to the logic ZERO conditon of the specified register.

RET . . . . . The unconditional RETURN instruction

RET C . . . . Return from the sub-routine if the carry flag in the F register is true (1).

RET M . . . . The instruction will only be performed if the S flag (sign flag) is negative.

RET NC . . . The instruction will only be performed if the NON-CARRY condition is present. i.e. the CARRY FLAG is '0'.

RET NZ . . . The instruction will only be performed if a NON-ZERO condition is satisfied. i.e. the ZERO FLAG is '0'.

RET P . . . . The instruction will only be performed if the sign flag in the F register is positive. i.e. S = 1.

RET PE . . . The instruction will only be performed if the PARITY is EVEN. This means the P/V flag is SET (1).

RET PO . . . The instruction will only performed if the PARITY is ODD. This means the P/V flag is reset (0).

RET Z . . . . The instruction will only be performed if the ZERO flag is SET (1).

RETI . . . . .Return from INTERRUPT.

RETN . . . . .Return from non-maskable INTERRUPT.

RL ( ) . . . The content of the memory location contained in ( ) is rotated to the left, through the carry bit.

RL A,B,C,etc . The contents of the register is rotated one bit position to the left, through the carry bit.

RLA . . . . . This is a one-byte instruction of **RL A** and rotates the contents of the accumulator one bit position to the left, through the carry bit.

RLC ( ) . . . The contents of a memory location pointed to by the contents of the location in ( ) is shifted one bit to the left but not through the carry. The C flag is set to the original status of the register's least significant bit.

RLC A,B,etc . The contents of the indicated register is shifted one bit position to the left. It does not shift through the carry bit but does set the C flag to the original status of the register's most significant bit.

RLCA . . . . . This is a one byte instruction of **RLC A** and operates as above.

RLD . . . . . The 4 low-order bits of a memory location (pointed to by the contents of register pair in brackets) are transferred to the 4 high-order bits of the same memory location. The 4 high-order bits are transferred into the 4 low-order bits of the accumulator. The previous 4 low-order bits of the accumulator are transferred to the 4 low-order bits of the memory location specified above.

RR ( ) . . . Rotate the contents of a memory location pointed to by the contents of the register in ( ) to the right. through the carry bit.

RR A,B,C etc Rotate the indicated register to the right through the carry bit.

RRA . . . . Rotate the accumulator right, through the carry bit.

RRC ( ) . . Rotate the contents of a memory location pointed to by the contents of the register in ( ) to the right but not through the carry bit. The C flag is set to the status of the register's least significant bit.

RRC A,B,etc Rotate register to the right but not through the carry bit.

RRCA . . . .A one-byte instruction for RRC A.

RRD . . . . . The 4 high-order bits of a memory location (pointed to by the contents of register pair HL) are transferred to the 4 low-order bits of the same location. The 4 low-order bits are transferred to the 4 low-order bits of the accumulator. The previous 4 low-order bits of the accumulator are transferred to the 4 high-order bits of the memory location specified above. A special one-

RST 00 . . . A special one-byte subroutine call directive called RESTART. RST 00 will restart at page zero, location 00. i.e. 00 00 RST 08 will restart at location 08. i.e. 00 08 etc.

SBC A,( ) . Subtract the contents of memory pointed to by the contents of the register pair in ( ) and the carry flag from the accumulator. Store the result in the accumulator.

SBC A,B . . The contents of the B register and the carry flag (the C flag in the F register) are subtracted from the contents of the accumulator. The result is stored in the accumulator.

SCF . . . . The carry flag (the C flag in the F register) is set to 1.

SET 0,( ) . Bit 0, etc of the memory location pointed to by the contents of the register in ( ) is set to 1.

SET 0,B . . The indicated bit in the selected register is set to 1.

SLA ( ) . . SHIFT LEFT ARITHMETIC. Shift the contents of the memory location pointed to by the contents of the register in ( ) one bit to the left. resetting the least significant bit.

SLA A,B,etc Shift the contents of the specified register left one bit, resetting the least significant bit to 0.

SRA ( ) . . SHIFT RIGHT ARITHMETIC. Shift the contents of a memory location pointed to by the contents of the register in ( ) to the right. The high-order bit is not altered. Bit 0 is shifted into the carry bit.

SRA A,B,etc Shift the contents of a register one bit to the right. High-order bit is unchanged. Bit 0 is shifted into the carry bit.

SRL ( ) . . SHIFT RIGHT LOGICAL. The contents of the memory location pointed to by the contents of the register in ( ) are shifted to the right. Bit 7 is reset to 0. Bit 0 is shifted into the carry bit.

SRL A,B,etc The contents of the indicated register is shifted one bit position to the right. Bit 7 is reset to 0. Bit 0 is shifted into the carry bit.

SUB ( ) . . Subtract the contents of the memory location pointed to by the contents of the register in ( ) from the accumulator.

SUB A,B,etc Subtract the contents of the specified register from the accumulator.

SUB dd . . Subtract immediate data from the accumulator.

XOR ( ) . . Perform the Exclusive-OR operation on data pointed to by the contents of the register in ( ) with the accumulator.

XOR A,B,etc Exclusive-OR the contents of the specified register with the accumulator.

XOR dd . . .Exclusive-OR the immediate data with the accumulator.