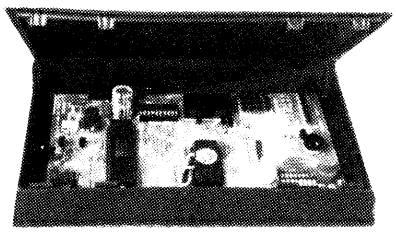
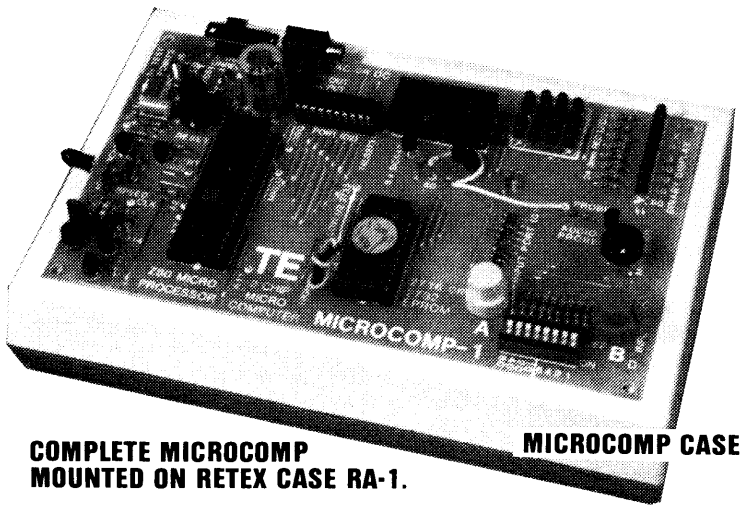
MICRO COMP

A 3-CHIP Z-80 COMPUTER

$55.75 COMPLETE
COMES WITH **FREE** STORAGE BOX!!



Parts: $47.25
Board: $8.50



COMPLETE MICROCOMP MOUNTED ON RETEX CASE RA-1.

MICROCOMP CASE $12.50

Don't think this project displaces the TEC. It goes hand in hand with it and uses some of the 'add-ons' and capabilities to assist in preparing programs. In reality you need BOTH. The TEC produces the programs and the MICRO-COMP runs them.

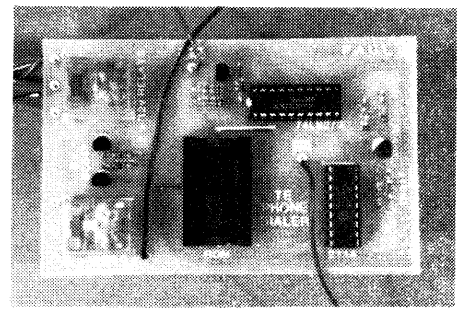
But if you are only starting in this field and want a low-cost introduction into micro-computer programming - THIS IS IT!

There are some pre-requisites however. Although the project is simple (according to computer standards), it needs a degree of competence in assembly and you should have constructed at least 6 other TE projects before attempting it. Digital projects have an inherently high degree of success however construction requires a fine tipped soldering iron and a lot of attention to detail.

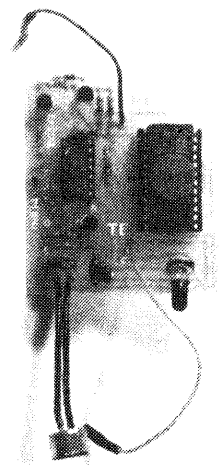
If you are not up to it, don't do it. But if you have made a lot of projects and want to graduate to the next level - this is how to go. You will be amazed with the capabilities of the unit and it will involve you in hundreds of hours of programming.

At the conclusion of this project you will:
- Know a little (a lot) about the Z 80 microprocessor.
- Learn about Read Only Memories and Random Access Memories.
- Learn about Input and Output ports and devices.
- Learn how a Micro system works
- Learn how to produce MACHINE CODE programs.

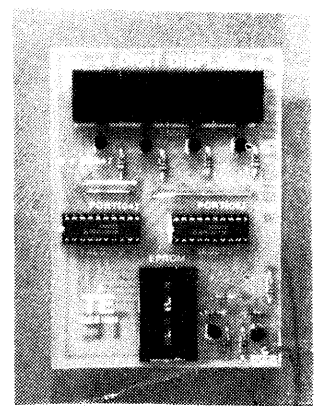
SOME OF OUR 'ADD-ONS':



PHONE DIALLER



MORSE TRAINER



4 DIGIT DISPLAY

This project contains everything to get you started in Machine Code programming. It assumes you know nothing about micro-processors or how they operate. The MICRO-COMP is the simplest computer to be offered on the market. It uses only 3 chips and a handful of small components to prove that computers can be tackled and mastered by anyone interested in electronics.

As with all our projects, full kits are available and come with a complete back up service.

As we have said, it doesn't displace the TEC but complements it. And yet the MICROCOMP is a stand-alone project. It is self-contained and comes with a range of interesting programs as shown on P. 70.

Ten programs have already been designed and come in the pre-programmed ROM. They include COUNTING, DISPLAYING and GAMES. The readout for these is via the displays on the board but as the games become more complex we have designed plug-in modules which connect to the main board via a wire-wrap/component header plug similar to the arrangement used in the TEC.

This has proven to be the neatest and most rugged way of adding features and allows you to increase and extend the capability of the system to quite high levels.

Some of the plug-in boards run two or three programs and by building all the modules, you will be able to run all the programs in the EPROM.

Programs which have already been completed include:
- MORSE CODE TRAINER
- MORSE RECEIVER
- TELEPHONE DIALER
- COUNTING
- MASTERMIND
- JUMP RELATIVE routine for determining the value of the displacement byte.

The MICRO-COMP can also be combined with the TEC and they go together perfectly. With the assistance of the TEC you can create your own programs, burn them via the EPROM BURNER or hold them in our non-volatile RAM card for running on the Micro-comp. This non volatile RAM project is equivalent to TAPE-SAVE and has the advantage of being able to be transferred instantly or run as a ROM.

It is a battery backed-up 6116 and when in the power-down mode, consumes less than 2 microamps. Two AAA cells power the unit and are capable of holding the information for about a year.

The TEC and MICRO-COMP provide a complete designing system for creating Machine Code programs and you can use the Micro-comp for the execution of the program.

The Micro-comp comes complete with a 2732 EPROM which is filled with lots of programs. All these have been produced entirely on the TEC and tested on the Micro-comp. We have not had the assistance of a compiler, video display or Z-80 simulator, proving that programs can be generated 'by hand'.

Agreed, this means it has taken longer to create the programs but the challenge was well worth it. Even the concept of a half-byte memory for the Phone Dialler was an innovation never before tried.

We admit Machine Code is not a fast method of creating programs but has the advantage that almost anything can be turned into a program. And it can be done with the simplest of equipment. The only limitation is the programmer's skill.

By building both the TEC and Micro-comp, you gain first-hand knowledge of two different methods of designing a micro system. You will also have need for add ons such as the non-volatile RAM and EPROM burner.

In effect you will be a self-contained programming station capable of turning out 'one-offs' or mass copying your own programs.

Please remember: These notes use simple terms and simple explanations to make programming easy. Although they are accurate, they do not cover everything and we suggest you purchase a couple of books on the Z-80. The two best books to buy will be given later.

Since its introduction, the word MICRO has been the most feared in the industry because, up to now, the operation of a microprocessor system has been very much a mystery.

Never has a writer explained or presented a system which could be understood by beginners. They argued it wasn't for beginners but everyone must be a beginner at some time. Because of this, Micro's have been a closed topic to the newcomer and this amazing electronic device has been left for the clever ones.

Now this has all changed.

The MICRO-COMP is here. With only 3 chips it is even simpler than a medium sized 'regular' project and yet its capabilities are beyond belief.

For the 'brains' of the unit we have used the Z-80. The most popular microprocessor on the market. Why the most popular? Because, up to now, industry swallowed them up totally and consumed the entire production. It's only with the slump in computer and games sales that supplies have reached the hobby market. And due to manufacturing efficiency, these truly amazing chips have come down to only a few dollars.

This means the project will be well within your budget.

Apart from the programs already mentioned, we are in the process of producing programs and modules for an Alcohol Breath Tester, a Digital Resistance Meter, Digital Capacitance Meter, a Bio-feedback Unit, a Mini Frequency Counter and a Lung Capacity Meter.

But before we get too carried away, let's look at the project in detail:

THE MICRO-COMP.

This is a 3-chip computer capable of accepting input data, performing operations on this data and displaying the results. The amazing part of this project is the three chip count. To achieve this we have used some very cunning circuit designs, some of which cannot be translated to larger designs. However our aim has been to produce a computer which will execute Z80 Machine Code with the least number of chips. And this we have done.

The reason for the minimum chip count is simple. Most constructors count the chips in a project before starting and anything over six scares nearly everyone away. With 3 chips, many will 'give it a go' and that's where we win. We want lots of readers to try their hand at construction and experience the excitement it offers.

Everyone has seen micro systems in a hard-to-get-at form. The **Personal Computer.** But these have never enabled you to get into the 'works' or let you find out how they operate. You only get to see the end result -the print-out or Video picture.

The Micro-comp is designed to break this barrier. With only 3 chips you will be able to follow a 'minimum parts' system and understand what is going on. Even with 3 chips you can

use nearly all the Z 80 commands and create an endless number of programs.

This project is really a software project. Building the Micro-comp is purely secondary. But how can you learn about programming without experimenting with the real thing? So building the Microcomp is really an essential part of learning to program.

Its price is low enough for everyone to afford and it has an end-use around the home as a controller for lighting or security which would match any commercial unit. You could also use it in your hobby, model railway layout or as a timer-controller in industry.

The MICRO-COMP doesn't do much when compared with a Personal Computer. But that's not its purpose. It is intended to teach Machine Code programming, the code behind all computer instructions.

The only instructions a processor understands is Machine Code. All other high level languages have been invented to allow humans to understand what is going on. Languages such as BASIC and FORTH provide a connecting link between the micro and the human mind. This means all inventors of languages have had to use machine code to write their programs. So why not use Machine Code direct?

Using BASIC is like hiring a scribe. Centuries ago people could not write. So they went to a learned man and told him what they wanted written. After a lengthy discussion he would write a letter. The letter represents Machine Code. The lengthy discussion represents BASIC.

BASIC has its advantages and a number of disadvantage. Its advantage is it gets you into a microprocessor system with very little effort and undertanding. But its disadvantage is it needs the 'scribe' to be present at all times. With machine code its like using the typewriter yourself.

But most important Machine Code is the best way for producing programs for controlling applications. When you consider all video games and industrial machines are Machine Code based, you will see where the future lies.

It is interestng to note that a micro system rivals a 'normal' project (using individual chips) when as few as 10 chips are involved. When you consider a microsystem can be modified and altered to suit changing circumstances, it is clearly the only way to go.

Why this hasn't been the case, is simply due to fear.

Everybody thought microprocessors were complex mysteries and preferred to stay with the building blocks they knew and trusted.

But, in fact, the micro system is simpler. Once the basic design is built, it only requires programming to perform the required function. To change the function, the electronics don't need altering, only the program!

Micro systems are simply thousands upon thousands of building blocks stored in the form of program and to write a program is equivalent to being able to create your own chips.

This is what the MICRO-COMP is. You can get it to execute your own programs and connect all sorts of input-output devices. You can get it do just about anything in the controlling and timing field but first you have to learn how to program.

To help you with this we have produced a number of programs to demonstrate the capabilities of the system and these are contained in the lower half of a 2732 EPROM which comes with the kit. Later you will be able to send it in for re-burning for the additional programs.

Before we get into the construction of the 'Comp, here's a brief discussion on how it works.

HOW A COMPUTER WORKS

This is a very simple explanation to get you started.

The operation of a computer revolves around a chip called the CPU. This applies to any computer and the MICRO-COMP is a computer, even though it is very simple. In our case the CPU is a Z 80. It is the 'brains' or 'clever chip' in the system and controls all the other chips.

CPU stands for Central Processing Unit and the feature which makes it so clever is it is good at organising things. It keeps the whole system operating and running smoothly.

In an audio or radio circuit there is usually only one signal path. In a computer there are lots of signal paths. This is the one striking difference between the two. In a radio, the path can be tapped at any point and you will be able to hear the signal (such as voice or music). If you tap any of the paths in a computer you will hear a series of clicks or tones and they will not make any sense.

This is because a computer requires a number of lines carrying signals AT THE SAME TIME to produce the necessary commands and output effects.

A single line in a computer will sound like a tone because of the high speed of operation but as far as the computer is concerned, the line is producing a HIGH for a very short period of time and then a LOW for the remainder of the time.

Since a single line can only produce a HIGH or LOW, a group of lines is required for the transmission of numbers. This is achieved by assigning the lowest-value line with '1', the next line with '2'. the next with '4', the next with '8', the next with '16' and so on.

By turning on combinations of these lines, almost any value can be transmitted.

A group of lines such as this is called a BUS and a computer has two buses. One consists of 8 lines and is called the DATA BUS, while the other has 12 or more lines and is called the ADDRESS BUS.

The microcomputer starts operating after the reset button has been pressed and released. This action resets the Program Counter inside the Z80 to 0000 and instructs the chip to fetch 8 bits of information from MEMORY.

It does this by putting zero's on all the address lines and turning on the 2732 via the Chip Enable line.

The EPROM responds by delivering the 8 bits of data which are located at 0000 to the data bus. The Z 80 accepts these and places them in a special instruction register which is only accessable to the Z80.

Eight bits of information is called a BYTE and the Z 80 determines what to do with the byte , according to its value.

The Z 80 will do one of two things. It will either carry out the instruction or request another byte. An instruction may consists of one, two, three or four bytes. and the Z 80 waits for a command to be completed before executing it.

Looking at the machine codes on the back of issues 11 and 12 you will notice some of them consists of one byte while other are 2, 3 or 4 bytes long. The Z 80 knows exactly how long each instruction is and knows that some contain a data byte or

displacement byte. This knowledge is inbuilt into the Z80 and only needs to be fed a simple program for it to respond.

The first byte from memory is always interpreted as an instruction and the byte or bytes which follow make up the first command. If you add a byte or delete one, at any time in a program, it will not be interpreted correctly and the Z-80 will carry out totally incorrect commands.

The Z 80 reads a program one byte at a time. It does not look ahead and cannot correct any mistakes. That's why it is important to check a program before offering it to be run.

Information passes out of the Z 80 via the address bus and into it via the data bus. After the Z 80 has processed the data, it will send the result out via the data bus. This means information moves in TWO directions along the data bus, although not at the same time.

In our case, information from the Z 80 is passed to an output latch. This latch is a device which fits between the computer and say a LED, motor or relay. The need for this chip is very important, as you will see. Data could be sent directly to a LED without using a latch and it would work. But the computer would have to stop functioning for the whole time when the LED is to be lit. This is obviously not a solution and so a chip is placed between the two which holds the 'turn-on' pulse for as long as the LED is required to be activated.

This chip is called a latch . It is merely a set of flip flops which hold the bits of information for as long as is required. This enables the Z 80 to get on with its other operations such as turning on a motor via another latch. Output devices such as LEDs and motors cannot be connected directly to any of the data lines for two reasons:
Firstly the current available in these lines will not be sufficient to operate them and secondly, the lines must be available for other purposes.

This means any device wishing to be placed on the data bus must be separated from it until the exact instant when it is required.

This is what an input latch does.

When these chips are not being activated, they place no load on the bus and allow the lines to rise up and down. This feature is called TRI-STATE as they are capable of producing a HIGH or LOW when required.

This is the basis of how a computer starts up. More aspects will be discussed later.

BEFORE YOU BEGIN CONSTRUCTION:

It is posssible to construct the Micro-comp using your own components and on your own PC board.

That's because all the parts are standard and the circuit board is fairly easy to reproduce. The 2732 EPROM can be programmed via an EPROM programmer and everything will operate perfectly.

There are only two hitches to you doing this.

First is the guarantee.
If you make the project from your own parts, it cannot be sent to TE for repair. We guarantee to fix any model made from one of our kits as we have had lots of experience at this. Mainly poor soldering joints, jumper links cut before soldering, parts inserted the wrong way around and broken tracks. Small faults but enough to keep the project from working.

Digital electronics is extremely reliable but not if you make a mistake.

The second hitch is reliability. If you use second-hand or unknown components, how do you know if they are perfect? They may have been over-worked or damaged in a previous project and fail when put in the Micro-comp.

Making your own Board?

The PC board is not as easy to make as it looks. One mistake in its etching and a track may be etched through. Or a hairline crack may be created in one of the lines which will be extremely difficult to spot. You also have to consider the overlay and solder mask. These make the project look neat and professional. You may save a few dollars at the start but end up costing more in the end. We have had a few troubles with home-made boards and unless you have made lots of boards before, we suggest buying a ready-made board.

Building from a kit is the safest way. All parts are absolutely brand-new and chips are transferred from bulk tubes without being handled. Boards are inspected three times during manufacture and made on semi-automatic equipment with very little margin for error. A sample kit is constructed before they are released and at least three prototypes have been made before the project goes to print.

This contributes to the success of our kits and the neatness of the finished project is enhanced by the solder mask on the underside of the board. This prevents solder sticking to unwanted areas and shorting between tracks.

To be sure of success, buy a kit. A number of shops are selling these kits and you will find the cost is less than hunting for the individual bits yourself.

PARTS LIST

1 - 10R
8 - 100R
1 - 330R
1 - 470R
1 - 3k3
1 - 4k7
14 - 10k
3 - 22k
2 - 39k
4 - 100k

1 - 100k mini trim pot

2 - 1n green cap
1 - 100n
1 - 1mfd 63v electro
1 - 1,000mfd 25v electro

9 - 1N 4148
4 - 1N 4002

1 - 5mm red LED (SPEED)
1 - 5mm green LED
24 - 3mm red LEDs
8 - BC 547 transistors
1 - BC 557 transistor
2 - FND 560

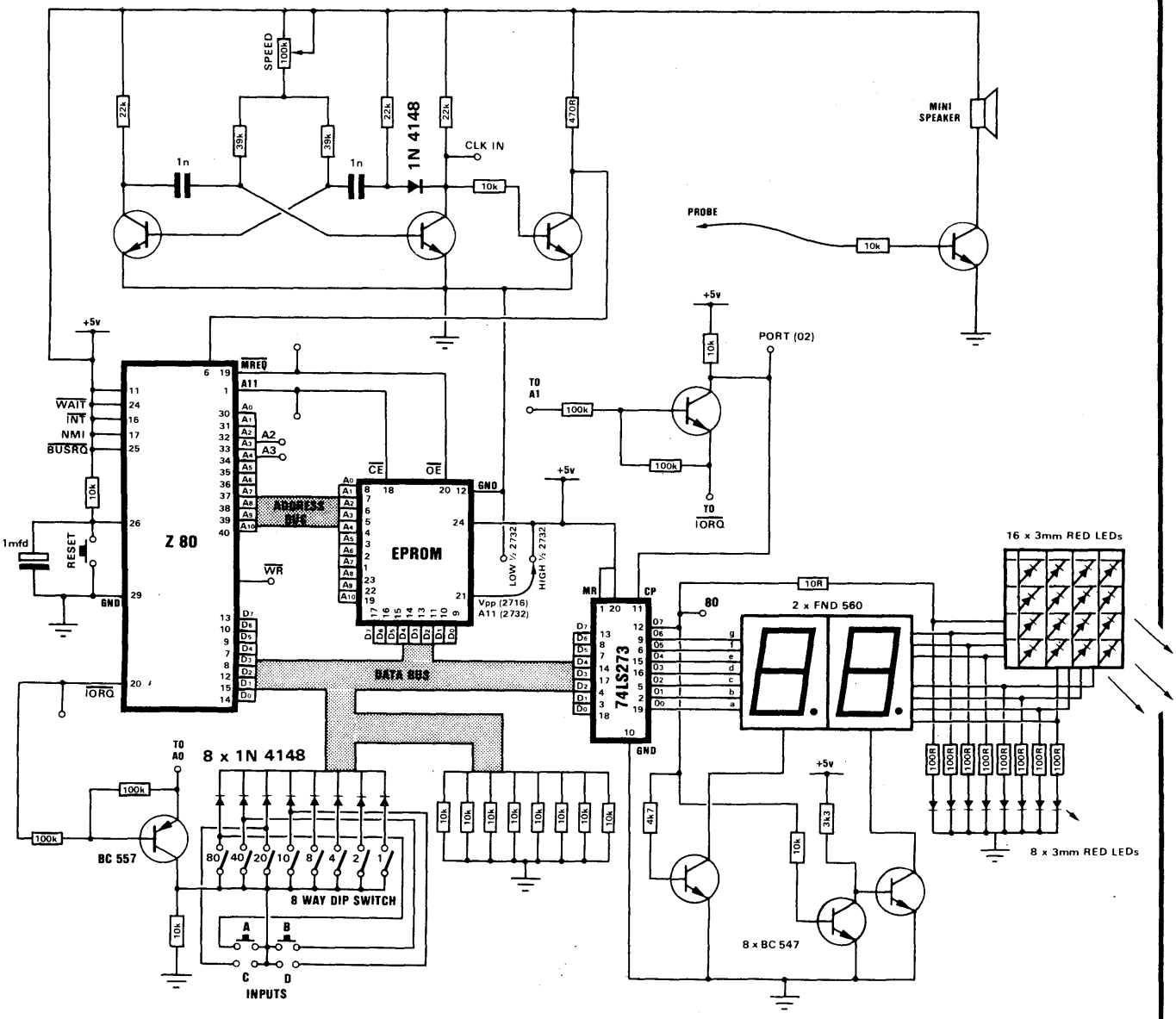
1 - 74LS273 IC
1 - Z-80 CPU
1 - 2732 EPROM (PROGRAMMED)
1 - 7805 regulator

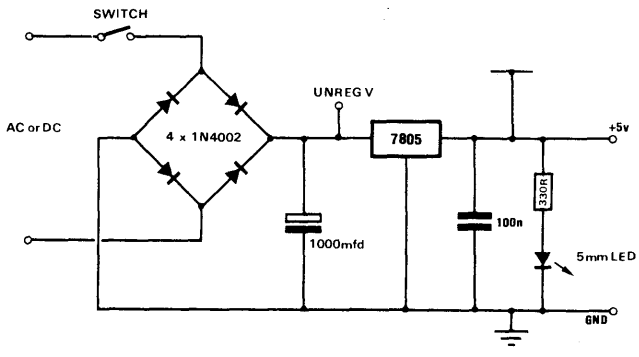
1 - 20 pin IC socket
1 - 24 pin IC socket
1 - 40 pin IC socket
1 - 8 way DIP switch
1 - DPDT slide switch
3 - PC mount push switches

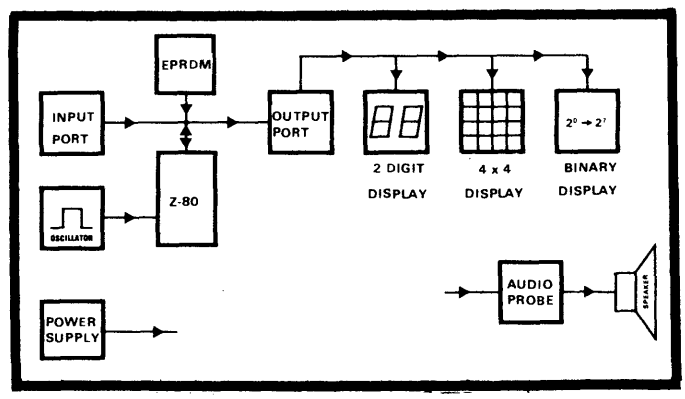
1 - 3.5mm mono socket
1 - mini speaker

1 - 6BA nut and bolt
4 - rubber feet
13 - matrix pins
1 - hollow pin
20cm hook-up flex
1 metre tinned copper wire
1 - female matrix pin connector
2cm heatshrink tubing.

MICROCOMP PC BOARD







BLOCK DIAGRAM OF MICROCOMP

CONSTRUCTION

Lay all the components on a sheet of paper and identify them. Make sure all parts are present.

Start assembly by fitting the jumper links. There are 41 of them and each must be inserted carefully to produce a neat result. For each, cut a piece of tinned copper wire longer than required and bend it to form a staple, with the long lower section kept as straight as possible. The two ends must fit down the holes cleanly and the wire must be able to be pushed right up to the board. This means the bends must be sharp.

If the two ends do not protrude through the board, do not attempt to solder the link as this will produce a dry joint which will be very hard to locate when troubleshooting. We have had two cases of this and it took hours to locate the fault.

Solder the ends of each jumper and cut the ends off with a pair of side cutters so that a little of the wire emerges from the solder. Do not cut through the solder as this will fracture the joint and possibly cause a fault.

Next fit the IC sockets. Make sure each pin fits down a hole before starting to solder. If a pin bends under a socket it will be very hard to rectify after the socket has been soldered. So check before-hand.

Solder one pin at each end to keep the socket in place while you attend to each pin.

Solder each pin very quickly and use fresh solder for each connection. The solder mask prevents the solder running along the leads or touching any of the lines which pass between the pins. It helps give a professional result and makes your soldering 100% neater. But don't use too much solder or blobs will result.

On the other hand don't use too little or the leads will not be fully surrounded by solder.

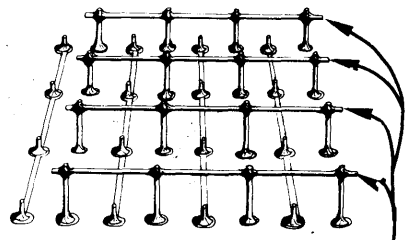
All the components are marked on the PC board and it is possible to build the project without any other help. But as a guide we will go through the assembly and explain everything as we go.

Basically you start with the smallest components which are closest to the board and progress to the highest or largest components.

We have fitted the lowest items and now the smallest. There are 13 connector pins on the board and one hollow pin for the TONE OUTPUT. The pins accept flying leads and female connectors as used in the plug-in modules.

Fit the 16 LEDs in the 4x4 display and 8 LEDs in the single row so that the flat on the side of each LED is on the right. Refer to the markings on the PC board. The cathode leads of the LEDs in the 4x4 are left long and a piece of tinned copper wire soldered across them, about half a cm from the board.

There are 4 individual pieces of tinned copper wire which join the cathode leads of the LEDs to the circuit. Refer to the drawing to see how this is done.



Note the 4 lines connecting the cathodes:

Next add the resistors and signal diode in the clock circuit. All these components touch the board and the leads are trimmed neatly after each is soldered. Next fit the 4 power diodes and eight BC 547 transistors. Almost any NPN small signal transistor will be suitable and BC 547 is only used as a guide. There is one BC 557 transistor used as the input decoder and this is indicated on the board with a white transistor symbol. All transistors should be pushed onto the board leaving a space between body and board equal to about the thickness of a resistor.

Mount the 100k mini trim pot and solder its leads. Push the leads of a LED through the screwdriver slot in the pot and bend them over so that the body becomes a handle. By turning the LED you will notice the trim pot rotates too.

Now comes the need for a careful bit of soldering. The two leads of the LED must be soldered to the rotating part of the pot so that the solder does not run over the edge and touch any other parts. If this happens the pot will be ruined as it will no longer rotate.

Fit the two 1n greencaps into the clock circuit.

Mount the ON-OFF switch and input jack so that they touch the PC board and solder the leads carefully.

The 7805 regulator is mounted under the PC board with a nut and bolt so that it touches the copper laminate. This will act as a heat sink and prevent the regulator getting too hot.

The leads from the regulator fit into the holes on the underside of the board and are snipped off the top side so that they don't protrude.

The two electrolytics must be mounted around the correct way. Observe the negative marking on the component and the positive marking on the board. The 1mfd reset electro is bent over and lays flat on the board to prevent it getting in the way of the reset button. Allow enough lead for this to be done.

A 'POWER-ON' LED is fitted near the regulator to indicate 5v.

Three push buttons are the next components to be fitted. The positioning of these is determined by a flat on the side of the switch aligning with the marking on the PC board. You can use any colour for the switches as they are not colour coded.

The mini speaker can be mounted either way around as it is not polarity sensitive. A 10cm wander lead is required for the probe and it must be long enough to reach over the entire board. A short piece of stiff wire can be soldered to the end of the lead to act as a probe tip or alternatively the wires can be soldered to make them stiff.

One jumper lead is required on the board to select either the upper half of the 2732 or lower half. A female socket is attached to the lead and kept in position with a short piece of heat-shrink tubing.

The last component to be fitted is the 8 way DIP switch. The numbers and/or letters on this switch must be removed before it is fitted to the board as they are not used in this project and may cause confusion.

Use a knife or blade and scrape the numbers until they disappear. Next you must determine which way around the switch is to be inserted as some switches are CLOSED when the lever is UP while others are closed when the lever is DOWN.

We require the switch to be closed or ON when the lever is DOWN so that each of the levers correspond to a number on the PC board. This is not essential and the switch will work satisfactorily around the other way, but to make things simple keep to our suggestions.

Check the operation of the switch with a multimeter before inserting it onto the board and solder it in position when it is correct.

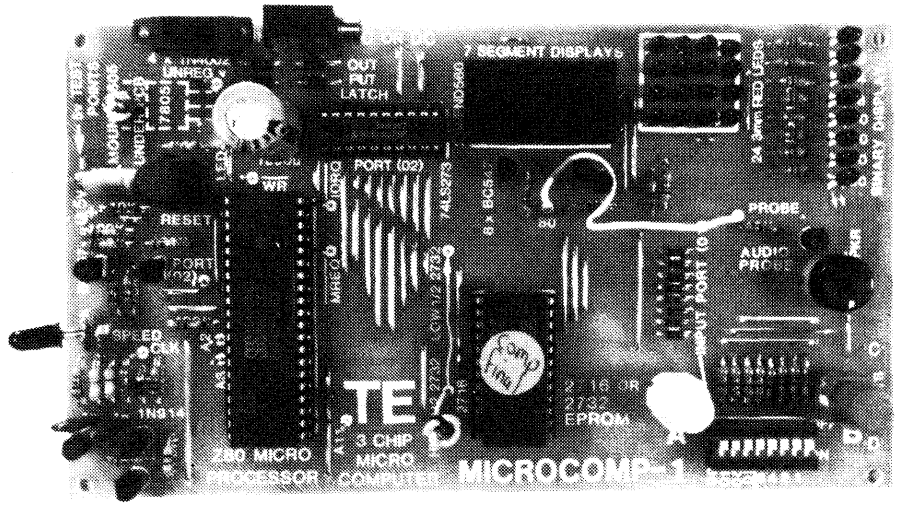
Fit 4 rubber feet to the underside of the board, insert the chips and you are ready for testing.

TESTING

Insert the power plug into the 3.5mm socket and switch the Microcomp ON. The power LED should come on. Make sure all the input switches are OFF. Push button B. The number 99 should appear on the displays. Press button A and the numbers will increment. Push button B and they will decrement. This is a fairly good indication that everything is working perfectly and you can go on to learning about programming.

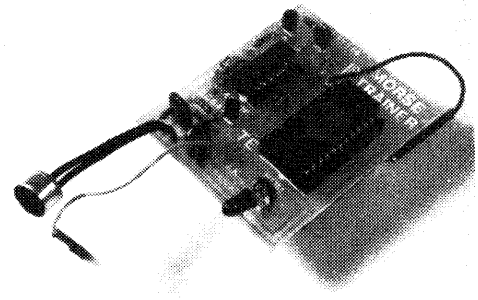
If you do not get 99 on the displays you may have a fault in the system. This will require you going through a trouble-shooting procedure as covered on P. 66.

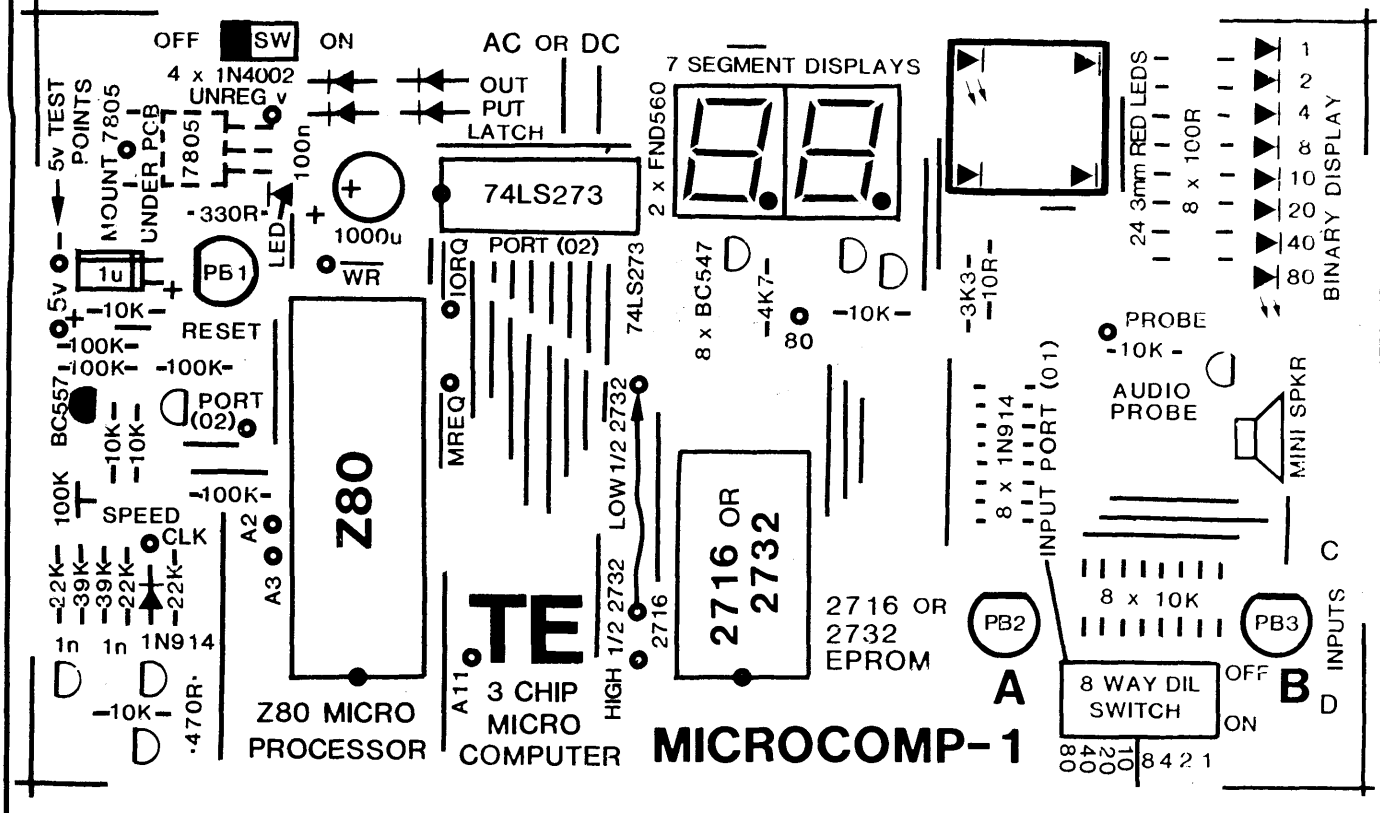
Consider yourself lucky that the computer doesn't work. You will gain a lot by trouble-shooting it yourself and gain experience in finding the fault.



Note the LED used as a knob for the SPEED control. SGS transistors don't work very well in the clock circuit. They freeze at high speed. To prevent this, use 47k base resistors.

The MORSE TRAINER is our first add-on and will be presented as soon as the programs in the lower half of the 2732 have been covered.





The overlay for the Microcomp shows all the component locations and link positions. The large donuts indicate the positions for the matrix pins. A wander lead selects HIGH/LOW 2732, while another is taken from the AUDIO PROBE input pin.

IF IT DOESN'T WORK

As we have said, digital projects are extremely reliable and have an enormous success rate. The chances of this project working as soon as the power is applied, is very high.

However, if it doesn't snap into life, here are some helpful suggestions.

Firstly check the power LED. It should come on as soon as the power is applied. If it doesn't, the fault will lie somewhere in the power supply.

Feel the 7805. It should get warm after operating for a few minutes. If it is very hot, you will have a short in the circuit. Turn off the computer and look for a bridge between two tracks. This may be anywhere at all on the board and this is how to go about it:

Measure between the positive and negative rails with an ohm-meter set to LOW OHMS. It should measure about 30 ohms in one direction and 50 ohms in the other. The values you will get are mainly due to the presence of protection diodes inside the chips and the resistors on the board. The actual value of resistance does not matter. Values such as this do not indicate a short circuit. But if it 10 ohms or less, a short-circuit is present.

Remove one chip at a time. If the low value is still present after all the chips have been removed, you will have to look for a fault on the board itself.

Start by removing the 7805 and then one end of each of the 41 jumper links. Measure the resistance value at each stage. If the short is still present, lift one end of each resistor and capacitor. If it is still there, it will possibly be a short between 2 tracks. You will need a magnifying glass and a sharp knife. Cut between the tracks at every location where you have made a connection to make sure no wiskers of solder are shorting between one land and another.

After this, the short should be removed. Refit the 7805 and switch ON. The LED should light. Refit all the components and jumpers. Use desolder wick to remove the solder from each of the holes so that the leads can be inserted.

If the power LED comes ON but none of the displays, set an input value on the switches of say '40' and reset the computer. This will produce '99' on the displays. If they remain blank, you will have to look into the operation of the system.

This is where the built-in **AUDIO PROBE** comes in. The probe lead will enable you to hear the signals on each of the active pins of the chips. We have specially designed the computer to operate at a speed which can be heard by the human ear. The probe will let you hear the frequency of the clock, the output of the address and data bus and also the activation of the latch.

Firstly turn the clock speed down and probe the 'clock-input' at pin 6 of the Z-80. You should hear a fairly high pitched whistle. As you increase the clock speed, this whistle will increase until it gets too high to hear. Next probe one of the data lines and you will hear a tone which is exactly one-eighth the frequency of the clock. If nothing is heard, it means the Z-80 is not operating or not accepting the input clock waveform. Make sure the reset pin of the Z-80, pin 26, is HIGH, otherwise the Z-80 will be sitting in a reset state.

If nothing is heard on the address or data buses, the fault will lie between the Z-80 and EPROM. They must be talking to one another for the system to start up. Even a blank EPROM (filled with junk of FF's) will produce a tone on the buses.

Test pin 18 of the EPROM to make sure it is being accessed. You should hear a tone on this pin which means the Z-80 is accessing the EPROM and trying to get it to place data on the data bus.

Some of the faults which can occur between the Z-80 and EPROM include non-soldered connections, IC socket pins which do not pass through the PC board and thus do not connect to the circuit, power not reaching the chips (due to a broken track), or a fault in one of the address or data lines near a solder connection.

This generally occurs when you are soldering and may be due to the iron being too hot, taking too long to produce the joint or moving the component while the solder is setting. The result is a hairline crack where the track meets the land and this is very hard to spot. Use a multi-meter set to low ohms to measure the continuity of each of the lines.

If everything seems to be correct, try replacing the Z-80. It does not matter if you use a Z-80 or Z-80A, they will both work equally well.

There is only one remaining possibility. The Z-80 requires a perfect square wave for it to function and we have gone to a lot of trouble to produce a near perfect waveform.

If the rise and fall time is not extremely short, the Z-80 will not accept it. This problem will be almost impossible to determine, even with a 30MHz CRO. If you have come to this conclusion, you should send your project in for a check-up.

Once you have values appearing on the displays, you can check for the correct operation of the programs by accessing our OUTPUT LATCH TEST ROUTINE. Turn on switches 01, 08 and 20. This will give a value of 290. Push reset and the micro will jump to address 0290. Three LEDs should illuminate: 01, 08 and 20. Now turn all switches OFF. All LEDs should extinguish. If any remain ON, the fault could lie in the input port. Check the soldering for shorts and all lines for continuity.

If a fault is present in one of the lines other than 01, 08 or 20, the micro will not address 0290 and the wrong program will appear.

If this is the case you will have to experiment with various settings and try to determine where the micro is jumping to.

If a wrong program is picked up, you cannot be sure it has accessed the beginning of the program and thus you cannot immediately determine which line is at fault.

Turn all switches OFF and press reset. The computer should not address any programs as the jump routine will be loading HL with 00 00 and jumping to the start again. Thus it will run around a loop, back to address zero.

If the 7-segment displays illuminate but not the 4x4 matrix, or the row of 8 LEDs, the fault will lie in the jumper lines which must be connected to the cathode leads of each of the 16 LEDs. See the construction notes for this as it will be the first time you have come across this method wiring the underside of a board.

The decoding transistors for ports 1 and 2 only come into operation when they receive the correct instruction via the program.

When the micro is executing the start-up program, it will be looking at the input port twice per loop and you will able to hear this in the mini speaker.

The output port will not be accessed during this time and probing the Latch Enable line will give no tone.

You must put a value on the input port switches to get the computer out of the start-up routine if you want to probe the output decoding transistor.

This is done at pin 11 of the 74LS273. If no tone is heard, trace the circuit back to the BC 547 (near the Z-80) and probe the base and emitter leads. When the transistor is being turned ON, a tone will be heard in the collector circuit.

If all these suggestions fail, start at the beginning again and solder each connection. Use desolder braid to collect any excess solder and inspect every joint under a bright light.

Make a continuity check of each copper track and make sure each land is not shorting to the one next to it.

Check all the LEDs for correct insertion and all chips for placement around the correct way.

Check the regulator, the 4 power diodes, the clock circuit, the placement of the 9 transistors, the positioning of the 3 push buttons and the value of all the resistors.

Ask a friend to go over the project and carry out the troubleshooting hints.

If all this fails, there is a repair service from TE and for $15.00 plus $4.50 postage, you can get your unit repaired. Send it in a jiffy bag with $19.50 and we will do our best. Up to now every computer sent to us has been repairable.

So, don't despair. Send it in and we'll check it out.

WHAT EACH CHIP DOES

There are 5 major building blocks in the MICRO-COMP. They are:

THE CLOCK - made up of 3 transistors
THE CENTRAL PROCESSOR UNIT - A Z-80.
THE MEMORY - A 2732 EPROM.
THE INPUT PORT - 8-way DIP SWITCH
THE OUTPUT LATCH a 74LS273

There are also a number of other active devices (transistors) which perform inverting and driving functions and also a single transistor connected to a mini speaker to provide an audio probe to listen to the computer in operation.

We have intentionally kept the chip-count down to make the project attractive and in this chapter we will discuss each chip and how it fits into the circuit.

THE CLOCK

Even though this is not a chip, we could have used one. The requirement of a clock is to produce a very fast rise-time waveform at a frequency to suit the project.

The clock in a computer controls the speed at which data flows through the whole system. The Z-80 will operate at a frequency as low as 7kHz and below this its registers will fail to hold information. This is because they are dynamic and have to be 'topped up' many times per second.

At the higher end of the range, the Z-80 will operate at 2.5MHz and a Z-80A at 4MHz.

In our project, we want the Z-80 to operate as slow as possible so that we can 'see' the program run and hopefully listen to the bus lines change tone as the program runs through its steps.

The reason for the clock circuit containing a diode and wave-shaping transistor is to generate a perfect square wave. The Z-80 is very critical as to the shape of the wave it will accept and the rise and fall edges must be extremely fast - especially at this low frequency.

In addition, we have included a speed control in the clock circuit so that the frequency can be adjusted from 7.5kHz to 35kHz. This is nearly a 5:1 ratio and allows each of the programs to be run at high and low speed.

Even at these speeds the Microcomp must be one of the slowest computers on the market as most operate at a clock frequency of 1MHz to 2MHz. But don't worry, even at 8kHz, you will see operations performed faster than you can think.

THE Z-80

This chip is the heart of the computer. It is called the CENTRAL PROCESSING UNIT or CPU for short. This is a truly an amazing chip and we could fill many pages on its workings.

You will pick up a lot more on how the Z-80 works as we progress with the notes and the main fact is it controls all the other chips in the system. It takes information from the 2732 and delivers the result of calculations and operations to the output latch. The speed with which it performs these tasks is controlled by the frequency of the clock.

The Z-80 is capable of controlling over 100 chips and you can see our 'comp is only a very small design.

The Z-80 is like a story-teller. It reads the 2732 like a book and delivers its interpretation to a child (the output latch). The input port is like a child telling the story-teller where to start in the book. The clock circuit is like a watch - telling the story-teller how fast to read.

THE EPROM

Chip number two is the program storage chip. It has been programmed by TE so that a number of programs and effects can be produced on the displays. These chips are bought in a blank condition and programmed by means of an EPROM PROGRAMMER so that they contain the necessary set of HIGHs and LOWs to make the Z-80 perform the required operations.

The EPROM supplied in the kit is ready to operate the computer but you can program your own or get a friend to program one for you and it will work just as successfully. The full listing to do this is supplied in the notes.

This is the main advantage behind the type of programming we are covering. It means you will be able to write programs for your own micro-computer controllers, produce the EPROM and get it running without the need for any outside help.

It is the most efficient type of program available, in terms of memory required. It consumes the least amount of memory and is used in all types of industrial applications and video games.

THE INPUT PORT

This is an interface between the computer and the real world. We have already mentioned the need for this connecting link.

The input port takes in information from a set of switches and loads it into the accumulator in the Z-80. The Z-80 operates on this according to the instructions in the program.

As well as the 8 switches, there are also 2 push buttons which are in parallel with the two highest value switches. Provision for two more switches (external to the board) is also provided on the PC.

The input port is software controlled and thus any of the switches can be programmed to perform any operation you wish. They can start a program, stop it, call up a number, increment a count value, decrement it, sound an alarm, dial a phone number and lots more.

A switch places a HIGH on the data bus, when it is closed, and only when instructed to do so via the program. The instruction for this is: IN A,(01) and the input decoder transistor is activated to allow this loading to take place. At all other times the switches put no load on the bus and allow the lines to rise and fall so that the other instructions in the program can be performed.

THE OUTPUT LATCH

The output latch is the third and final chip in the project. This is the chip which drives the set of output LEDs and displays. We have created three different types of display and each will produce its own special effect according to the program being run. The main purpose of this latch is to hold the information coming from the Z-80 for long periods of time so that we can view it on the displays. This allows the Z-80 to go away and carry out other operations.

A set of transistors turn on one or other of the 7-segment displays via the 8th line so that a two digit number can be displayed.

INPUT/OUTPUT

The Microcomp is capable of accepting information from the outside world as well as delivering to the outside. This capability is called INPUT/OUTPUT.

In a simple system such as ours, for each address line it is possible to connect 8 devices to the data bus and access them individually via the program. These devices must also be gated into operation via the IORQ line.

Devices can have either input or output capability and since the Z-80 has 16 lines, this gives us a lot of devices! This is more than we require and to keep it simple we will consider only one set of 8 on address line A0 and one set on address line A1.

THE INPUT PORT

Input information is obtained from a set of 8 DIP switches and these are connected to the data bus. Eight switches like this gives us the capability of up to 256 combinations.

When address line A0 goes HIGH and IORQ goes LOW the value on these switches is passed to the Z-80 as an input value.

These switches are software programmable and can be instructed to perform many tasks, depending on the instructions in the program. The micro only looks at the switches during the instruction IN A,(01) and during the remainder of the time the switches are allowed to float up and down and don't interfere with the data bus.

THE OUTPUT PORT

The OUTPUT PORT is a latch chip. It must be a latch to hold the output value long enough for us to see the data on the displays. The latch will retain this data until updated.

There are two gating transistors in this project. One controls the input port and the other controls the output port.

Each transistor produces a LOW output when the I/O Request is LOW and the prescribed address line is HIGH.

The I/O Request line does not determine the IN or OUT nature of the signal, it just goes low when the Z-80 requests one of its ports. The circuitry and instruction in the program determines the IN or OUT condition.

THE DISPLAYS

The Microcomp has three different types of displays:

★ **Two 7-segment displays**
★ **A 4x4 matrix of LEDs**
★ **A row of 8 LEDs.**

Each display provides a different effect for any given set of values and you will be able to make a comparison between them as the programs run.

Here are a few facts and hints on producing effects on the displays.

At first you may be surprised to see two 7-segment displays operating from one latch chip. Normally this is not possible as all the lines from one latch are required to drive the LEDs in the display.

But by using only 7 lines to drive the segments, we have one line left over to switch between the two displays. This eighth line is normally used to drive the decimal point but this is the sacrifice we have had to make.

In our arrangement only one display will illuminate at a time and to make them both appear to be illuminated at the same time we must switch rapidly between them. This will create a two-digit number and allow us to produce a readout for a 00 to 99 COUNTER. It will also give us a number of other effects as you will see in the programs.

The 4x4 also connects to the latch and because the LEDs are connected in a different way to the 7-segment displays, a completely different effect will be created. A program for the 4x4 will not be recognisable on the 7-segment displays and vise versa.

The 4x4 matrix can be thought of as a miniature display board. It is connected to the latch via 4 horizontal lines and 4 vertical lines. The anodes of the LEDs are connected to the 4 lower bits of the latch such that the first column goes to bit 0. Column 2 goes to bit 1, column 3 to bit 2 and column 4 to bit 3.

The anodes of all the LEDs are connected to the 4 higher bits of the latch such that the lowest row connects to bit 4. The second row connects to bit 5, the third row connects to bit 6 and the top row to bit 7.

This means bit 0 sources 4 LEDs and so does bit 1, 2 and 3. Bit 4 sinks 4 LEDs and so does bit 5, 6 and 7.

To turn on a LED, the source bit must be HIGH and the sink bit must be LOW. This arrangement will allow any individual LED to be illuminated and even certain combinations of LEDs. But it does not permit absolutely any combination to be illuminated due to our wiring.

We can overcome this by a trick in programming called multi-plexing. This will be covered later and can be seen in the dice project.

To see exactly how the LEDs are accessed, address the program at 0290. By switching off the input switches you will turn the matrix off. Load input values into the switches and you will see the rows and columns of LEDs illuminate.

The third display is a row of 8 LEDs. This display can be referred to at any time for both the binary value and hex value being outputted from the latch. The binary value is simply obtained by looking along the row of LEDs and noting the on-off pattern. By adding their value in binary you obtain the decimal value of the latch.

But decimal values are of no real use to us in this project as we are concentrating on hexadecimal notation.

To find the hex value of the output latch, add the hex values alongside each LED. This is easy to do after a little practice.

Using the three displays together you will see the hex value required to produce letters and numbers on the

7-segment displays and also see what the micro in inputting and outputting in binary form to create these numbers and letters.

In all, it gives a graphic picture of what is going on.

THE AUDIO PROBE

The audio probe consists of a single transistor and a mini speaker. Its prime function is to enable you to listen to the 'computer in operation'.

This is possible when the clock speed is turned down and the probe touched on each of the pins of the chips.

It is interesting to hear the HIGHs being sent along the lines, especially the address bus where each line is running at half the frequency of the previous. The Z-80 is acting like a 16-stage divider and you can hear this on the probe.

The probe is also used for determining the operation or non-operation of the Z-80. This is one of the tests you will be required to do when setting up the project as the Z-80 requires a near-perfect square wave for it to operate.

The easiest way to see if it is accepting the clock pulses is to listen to the address or data lines.

The only way to know if the Z-80 is accepting the clock is to use the probe on pin 6 of the Z-80 and then on one of the address lines.

The audio probe is also used during the course of the experiments. By comparing the program with the tones on the buses and the Latch Enable pin, you can determine how often the chip is being accessed.

The audio probe also connects to pin '80' on the PC board which is bit 7 of the output latch. The Tone program at 0010 outputs a HIGH to this line and then a LOW to produce a click in the speaker. This is the basis to producing tones and by varying the speed control, the pitch can be altered.

WHAT IS THE 2732?

The 2732 is a memory chip containing 32,768 individual cells which can be programmed to contain a small charge.

Each cell is a single P-channel MOS transistor capable of detecting the presence of a charge.

This charge is held on a conducting layer above the transistor, on a thin film of insulating material. When the charge is present the transistor outputs a HIGH. When the charge is not present, the transistor outputs a LOW.

We can access each of these 32,768 cells and supply them with a small charge during programming. The charge remains in place for many years because it has no where to jump to as each area is surrounded by insulation.

Exposure to ultra violet light will give the charges sufficient energy to jump off, leaving the plate in a neutral state.

When you look through the quartz window you can see the array of cells. It seems incredible that over 32,000 cells can be seen, but that's the reality of electronics.

We access these cells 8 at a time and this is equal to one BYTE. This is the basic unit which is fed into a processor and is the basis of all Machine Code programs.

One byte can have up to 256 possibilities due to the fact that each of the 8 cells can be either ON or OFF.

To output these 8 bits of data from the chip we need 8 lines and these form the DATA BUS.

We need another set of lines into the chip so that we can locate these 8 cells. For a 2732 we need 12 lines and these are called the address bus.

There is one interesting feature about the address and data lines. Even though they are identified as A0, A1, A2, ...D0, D1, D2 etc. they can be connected to the microprocessor in any order. This is because the cells are uncommitted and provided you read in the same order as it was programmed, the correct data will be outputted.

The only reason for keeping to an accepted pin-out arrangement is so the EPROM will work in other designs and on common programming equipment.

THE GATING TRANSISTORS

Input and output ports must only come into operation when requested. At all other times they must not put a load on the data bus as it is required for other communications.

However when an instruction such as IN port 1 is sent to the Z-80, there are two lines which will be held in a stable condition and can be used to activate the port latch. These are I/O Request and address line A0. These can be gated together and the resulting pulse used to activate the port.

This is called simple decoding and since the Z-80 has a number of address lines it is possible to connect lots of input/output devices.

We have used only the first two lines, A0 and A1 and they provide a simple way to achieve an end result.

With this arrangement, the first device will be activated with the instruction: IN A,(01) and the second by IN A,(02). Further devices would be activated via IN A,(04) IN A,(08) and IN (10),A. By adding port values together, more than one port can be activated at the same time, should this be necessary.

USING THE DIP SWITCHES

The 8 dip switches are connected to the input port and are capable of providing up to 256 different combinations.

Eight lines like this is equal to one byte and depending on the program being run, this value can be used in many ways. Examples can be seen in the programs contained in the EPROM that comes in the kit.

We will now explain the meaning of the values on the PC board, alongside each of the switches.

You will see numbers: 80, 40, 20, 10, 8, 4, 2, 1. These are hex values and are an easy way for us to give values to a set of binary switches. The other option is to write: 1, 1, 1, 1, 1, 1, 1, 1.

Hex is a successful solution to writing values from 1 to 256 in a form which is easy to read and only requires 2 digits. To input a value such as 234 refer to the Hex Conversion table on P 16 of issue 11. It is equivalent to EA. Once you are in Hex notation, you stay in Hex. This makes it awkward when you see values such as 10, 20 45, 80, 100 but you must remember these are also Hex values and a number such as 10 (one-oh) is really 16 in decimal notation.

To place EA on the switches, you need to know about Hex addition. For instance E is made up of: 8, 4, 2, and 1. This is how it is done on the input switches: The switches are separated into two banks of four. The low value switches are labelled 8, 4, 2, 1. The high value switches are 80, 40, 20, 10.

The value EA is placed so that E will be loaded into the high section and A into the low section. To enter E turn

on switches 80, 40 and 20. This gives E0. To produce the value A, turn on switches 8 and 2. The input switches now hold EA.

After you have used them a few times you will become familiar with their operation.

One of the main uses is to generate a JUMP VALUE to get to the programs in EPROM. The computer interprets the value on the switches as a START ADDRESS by multiplying the value by 10 (one-oh) and jumping to the address of the value created.

The multiplication value of 10 is a hex value and is equal to 16 in decimal.

For example if we load the switches with the value '1', the start-up program will convert it to 10 and produce the address 0010. This is the address of the first program in memory - a TONE routine. To address the RUNNING NAMES routine, load the switches with 8. This will make the Z-80 jump to 0080, when the reset button is pressed.

In a similar way, the start of each of the programs can be accessed via the switches. For instance, the Final Message at 07A0 is addressed by loading 7A.

Although we can only address every 16th location, the programs have been written to start at an even Hex value and end before an addressable location. Some programs occupy 80 or more bytes while other take less than 8. This means some locations will be unused but this is the limitation of the system.

Experiment by loading the start address of various programs and run them to see how they operate.

THE PROGRAMS

We now come to the programs themselves.

The list shows all the programs in the lower half of the 2732. The number in the first column is the START ADDRESS which is loaded into the DIP switches. Once the program has been accessed, you can use the push buttons and any of the DIP switches to operate the program.

Whether you have burnt your own EPROM from the listing or bought a kit, you will want to know how the programs are put together and how they run. That's the whole purpose of this project.

Study each program carefully, running it at different speeds and answer any questions associated with the listing.

LIST OF PROGRAMS:

0000	JUMP ROUTINE
0010	TONE
0020	QUICK DRAW
0080	RUNNING NAMES
00D0 - 00F4	RUNNING LETTER ROUTINE (can be called)
100 - 1FF	LIST OF NAMES
200 - 28F	LOOKING AT DATA
290 - 29F	FROM INPUT TO 8 LEDs
2A0 - 2BF	INCREMENT VIA BUTTON A
2C0 - 2CC	AUTO INCREMENT (fast)
2D0 - 2DD	AUTO INCREMENT (variable)
2E0 - 2EC	AUTO DECREMENT
2F0 - 2FF	AUTO DECREMENT (variable)
300 - 36F	4x4 EFFECTS
370 -	0 - 9 COUNTER
390 -	0 - F COUNTER
3A0 -	A - Z, 0 - F COUNTER
3F0 - 3FF	VERY LONG DELAY
400 - 469	00 - 99 COUNTER
470 - 51F	DICE
520 - 52F	EPROM IN BINARY
530 - 623	POKER
630 - 6BF	BINARY CLOCK
6C0 - 6CB	ONE MINUTE TIMER
6D0 - 6DB	3 MINUTE TIMER
6E0 - 6EB	1 HOUR TIMER
6F0 - 738	ADJUSTABLE TIMER
740 - 760	1 MINUTE DELAY
765 - 79D	Table for adjustable Timer
7A0 - 7FF	FINAL MESSAGE

These programs occupy the lower ½ of a 2732 EPROM.

at 0000:
THE JUMP ROUTINE

This routine will be used every time you want to access one of the programs.

Set the address value on the input switches and press reset. The micro will then jump to the program you have selected.

Each program is a loop and the Microcomp will run around this loop.

The input switches can now be used for other functions according to the demands of the program. Don't push reset as this will cause the micro to jump out of the program. Only buttons A and B are used during the course of the programs. These are equivalent to switches 8 and 7.

THE JUMP PROGRAM

1.	**LD B,00**	**0000**	**06 00**
2.	**IN A,(01)**	**002**	**DB 01**
3.	**LD HL 00 00**	**004**	**21 00 00**
4.	**LD L,A**	**007**	**6F**
5.	**ADD HL,HL**	**008**	**29**
6.	**ADD HL,HL**	**009**	**29**
7.	**ADD HL,HL**	**00A**	**29**
8.	**ADD HL,HL**	**00B**	**29**
9.	**JP (HL)**	**00C**	**E9**

This routine looks at the input port (01) and jumps to the address set on the input switches.

The program multiplies the value set on the switches by 10 (one-oh) and jumps to this value.

If no switches are set, the program constantly loops back to 0000, looking for an input from the switches.

If '1' is loaded on the switches, the program jumps to 0010. If '2' is set, to program jumps to 0020 etc. If switches 20, 8 and 1 are set, the program jumps to 0290.

In this way we can access from 0010 to 07F0 in blocks of 10 hex bytes. This is equal to every 16 bytes and gives us a very good coverage of the EPROM.

The way in which the program works is this:

Line 1 loads the B register with 00 ready for a **DJNZ** statement as required in some of the programs. It has nothing to do with this program.
Line 2. The program looks at the input port and loads the value it finds on the switches into the accumulator.
Line 3. The HL register pair is zeroed.
Line 4. The accumulator is loaded

into the L register, which is the LOW register of the pair.
Lines 5, 6, 7 and 8 add the contents of the HL register pair to itself four times. Each **ADD** doubles the result, making a total increment of 16 times. A multiple of 16 is equal to **10** in hex.

Line 9. The micro jumps to the address given by the value of the HL register pair.

QUESTIONS:

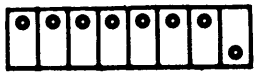
1. Set the switches to address values which are not the start addresses of a program. Why do some of them work?
2. Why does button B address the start of the 00 - 99 counter?
3. Could the DIP switches be replaced with push buttons?
4. Explain what we mean by the input switches are software programmed:
5. Name a few devices which can be connected to the input port:

ANSWERS

1. Sometimes you can start part-way through a program and it will run. This is because the micro jumps into a location it understands and it follows the program to the end. It then jumps to the start of the program and produces a full display on the screen.
2. Button B has the same value as '40' on the switches and this corresponds to address 400 in the EPROM.
3. Yes, but remember up to seven buttons would have to be pressed at the same time to achieve the result of the DIP switches.
4. The input switches can be programmed to do anything, as requested by the program.
5. Any device which has a set of contacts such as a relay, morse key, micro-switch, pressure mat or even transistors acting as switches can be used.

THE TONE ROUTINE

The TONE routine is located at 0010 and this is addressed by switching the lowest value switch ON thus:



The principle behind creating a tone is to toggle an output bit. The speed with which the bit is toggled, produces the frequency of the tone. To produce a 1kHz tone requires a minimum clock frequency of about 50kHz. This is because the clock frequency is divided by eight to run the data bus and further clock cycles are required for the load and output instructions. Since the maximum frequency of the Microcomp is about 35kHz, the highest tone which can be produced is 700Hz.

This is not sufficient for a musical scale or a tone generator and only a sample tone has been included in the EPROM.

By inserting the lead of the AUDIO PROBE into terminal '80' on the board, below the 7-segment displays, the tone will be reproduced in the mini speaker.

You can compare this tone with the Latch Enable pin and the data bus and see if the tones are different.

The TONE routine is a loop, starting at 0010 and ending at 001F. The first instruction **AF** is a single byte instruction which clears (zeros) the accumulator so that this value can be outputted to port 2. The accumulator is then loaded with 81 which

TONE ROUTINE:

XOR A	0010	AF
OUT (02),A	0011	D3 02
LD A, 81	0013	3E 81
OUT (02),A	0015	D3 02
XOR A	0017	AF
OUT (02),A	0018	D3 02
LD A, 81	001A	3E 81
OUT (02),A	001C	D3 02
JR 0010	001E	18 F0

produces a HIGH to the AUDIO PROBE input pin and also turns on segment 'a' of the first display. This is the complete TONE routine. The sequence has been repeated again to use up the available memory before jumping back to 0010 via a JUMP RELATIVE instruction.

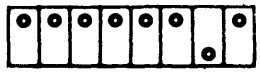
The program will loop continually until the reset button is pressed. The input switch must be OFF to prevent the program being accessed again.

QUICK DRAW PROGRAM

LD C,02	0020	0E 02	This is the COUNT register for 2 rotations of the display.
LD D,08	0022	16 08	D is the COUNT register for the 8 LEDs
LD HL,00F5	0024	21 F5 00	Load HL with the start of the byte table.
LD A,(HL)	0027	7E	Load the accumulator with the value POINTED TO by HL.
OUT (02),A	0028	D3 02	Output the accumulator to port 2.
DJNZ 002A	002A	10 FE	Register B contains 00 (via jump program) **DJNZ** is a DELAY.
INC HL	002C	23	Increment HL to look at the second byte in the table.
DEC D	002D	15	Increment the BYTE-TABLE COUNTER.
JR NZ 0027	002E	20 F7	If end of table not reached, jump to line 4. Otherwise next line.
DEC C	0030	0D	Decrement C and illuminate 8 LEDs again.
JR NZ 0022	0031	20 EF	If C is zero, advance to next line.
LD A,00	0033	3E 00	The accumulator is zeroed to blank the display.
OUT (02),A	0035	D3 02	The Accumulator is outputted to port 2.
LD DE, 0602	0037	11 02 06	The DE register pair is available for a long DELAY.
DEC DE	003A	1B	Decrement DE
LD A,D	003B	7A	Load D into A
OR E	003C	B3	OR E with the accumulator
JR NZ 003A	003D	20 FB	Jump if both D and E are not zero.
IN A,(01)	003F	DB 01	Input the two switches.
BIT 6,A	0041	CB 77	Test BIT 6 to see if switch B is pressed.
JP NZ 0020	0043	C2 20 00	Jump to start of program if button B is pressed.
BIT 7,A	0046	CB 7F	Test BIT 7 to see if button A is pressed.
JP NZ 0020	0048	C2 20 00	Jump to start of program if A is pressed.
LD A,0F	004B	3E 0F	Load A with **0F** to produce a 'backward C'.
OUT (02),A	004D	D3 02	Output **0F** to port 2.
LD B,08	004F	06 08	Load B with 08 for a short DELAY ROUTINE.
DJNZ 0051	0051	10 FE	**DJNZ** decrements register B to zero.
LD A,B9	0053	3E B9	Load the accumulator with **B9** to produce 'C' in display 1.
OUT (02),A	0055	D3 02	Output **B9** to port 2.
IN A,(01)	0057	DB 01	Input the two switches to see if either is pressed.
BIT 6,A	0059	CB 77	Test BIT 6 to see if B is pressed.
JR NZ 0066	005B	20 09	Jump if button B is pressed.
BIT 7,A	005D	CB 7F	Test BIT 7 to see if button A is pressed.
JR Z,004B	005F	28 EA	Jump back to line 24 if not pressed and loop constantly.
LD A,B0	0061	3E B0	If button A is pressed, load the accumulator with **B0**.
OUT (02),A	0063	D3 02	Output **B0** to port 2 to give '1' on display ONE.
HALT	0065	76	The program HALTS. Reset by pressing reset button.
BIT 7,A	0066	CB 7F	Test bit 7 to see if button A is also pressed.
JR Z,0074	0068	28 0A	Jump if button A is not pressed.
LD A,06	006A	3E 06	Load Accumulator with **06**
OUT (02),A	006C	D3 02	Output **06** to get '1' on display two.
LD A,B0	006E	3E B0	Load accumulator **B0**.
OUT (02),A	0070	D3 02	Output **B0** to port 2 to get '1' on display ONE.
JR 006A	0072	18 F6	Jump back to display 1's on both displays. Keep looping.
LD A,06	0074	3E 06	Load the accumulator with **06**.
OUT (02),A	0076	D3 02	Output **06** to port 2.
HALT	0078	76	Halt. Press reset button to reset game.

QUICK DRAW

Quick Draw is located at 0020 and this is addressed by switching ON the second lowest switch thus:



Quick Draw is a reaction game for two players. Player 'one' uses button A and player 'two' button B.

The game is played on the two 7-segment displays and the program starts by illuminating segments around the two displays. Then the perimeter of the two displays illuminate.

The first player to press his button is the winner and this is shown by a '1' appearing in the appropriate display.

If both players press at the same time, both displays illuminate.

If a player 'beats the gun', the game resets.

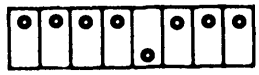
Press the reset button to start a new game.

Data Bytes at 00F5:

01
02
04
08
88
90
A0
81

RUNNING NAMES

To access this program, switch 8 must be ON. This will produce address-value 0080. Do not turn on switch 80 as this will produce 0800! Once the program has started, the switches can be turned OFF or set to the value necessary to access the name you want to appear on the screen.



Running names is a program which you use soon after the Microcomp has been completed.

It displays a message saying the builder of the project is YOU!

To do this we have included a list of about 30 names and these are accessed by loading the input port with a particular value, once the program is running.

Hopefully your name is amongst the list, but if not, there are a few general names at the end of the table to cover those excluded. Names containing M and W have been left out due to the difficulty in displaying them on the 7-segment displays. But for the majority, a name can be added to the message to add a personal flavour to the project.

The main program consists of 4 different sections. The first produces the message: "3-Chip uP built by". The second looks at the list of names and counts the FF's separating the names. It compares this with the value set on the input switches and displays the chosen name.

Part 3 of the program flashes 'C' on the screen to represent copyright and the 4th part of the program produces the date: **1985.**

The letters running across the displays are produced by a sub-routine which is used for the first, second and fourth parts of the program.

This sub-routine picks up the first two bytes in the table and displays them on the two displays. When the

RUNNING NAMES:

MAIN PROGRAM:

LD IX 0100	0080	DD 21 00 01	The IX register points to the start of the byte table.
LD HL 008A	0084	21 8A 00	The HL register provides a return address for the sub-routine.
JP 00D0	0087	C3 D0 00	Jump to the RUNNING LETTER sub-routine.
LD C,00	008A	0E 00	'C' is our COUNT register and is compared with an input value
LD IX 0114	008C	DD 21 14 01	IX is loaded with the start of the NAMES table.
IN A,(01)	0090	DB 01	Input the value on the switches, to the accumulator.
CP 00	0092	FE 00	If the input value is 00, the program increments to line 8 and
JR Z 00A9	0094	28 13	the micro jumps to line 20. If input value is NOT zero, to to 9:
LD D,A	0096	57	The input value is SAVED by loading it into register D.
LD A,(IX	0097	DD 7E 00	The data byte pointed to by the IX register is loaded into A.
CP FF	009A	FE FF	The accumulator is compared with FF to detect end of name.
JR Z 00A2	009C	28 04	If end of name is reached, the program jumps to line 15.
INC IX	009E	DD 23	If end of name not reached, INC IX and jump to line 10, where
JR 0097	00A0	18 F5	the next byte is loaded into A and compared with FF.
INC C	00A2	0C	The C register is incremented, indicating end of word.
LD A,C	00A3	79	Load C into the accumulator.
CP D	00A4	BA	Compare accumulator with D to see if word has been located.
JR NZ,009E	00A5	20 F7	Jump if word is not found.
JR 00AB	00A7	10 02	Jump OVER line 20.
DEC IX	00A9	DD 2B	This line only applies to the first word in the list.
LD HL,00B3	00AB	21 B3 00	Load HL with the return address for the sub-routine.
INC IX	00AE	DD 23	Increment IX for the first letter of the name.
JP 00D0	00B0	C3 D0 00	Jump to the LETTER RUNNING routine and display name.
LD C,08	00B3	0E 08	The C register is used to count 'COPYRIGHT' flashes.
LD A,58	00B5	3E 58	Load accumulator with **58** to produce letter 'C' on display.
OUT (02),A	00B7	D3 02	Output 58 to port 2.
DJNZ	00B9	10 FE	C remains ON for 256 loops of **DJNZ** (B register).
LD A,00	00BB	3E 00	Accumulator is loaded with zero.
OUT (02),A	00BD	D3 02	Zero is outputted to turn OFF 'C'.
DJNZ	00BF	10 FE	Display is OFF for 256 loops of **DJNZ** .
DEC C	00C1	0D	The ON-OFF count register (RegisterC) is decremented.
JR NZ 00B5	00C2	20 F1	ON-OFF effect is repeated 8 times.
LD IX,01F8	00C4	DD 21 F8 01	Register IX is loaded with 01F8, data for '1985'.
LD HL,0080	00C8	21 80 00	Register HL is loaded with return address (re-start address)
JP 00D0	00CB	C3 D0 00	Program jumps to RUNNING LETTER routine.

RUNNING LETTER ROUTINE: - sub routine

LD C,0B	00D0	0E 0B	Each letter appears **0B** times (11 times)
LD A,(IX +00)	00D2	DD 7E 00	Load accumulator with byte pointed to by IX
SET 7,A	00D5	CB FF	SET bit 7, to tun on left-hand display
OUT (02),A	00D6	D3 02	The accumulator is outputted to port 2.
LD B,20	00D7	06 20	Load B with 20 (for 32 loops of **DJNZ**) for time delay.
DJNZ	00D9	10 FE	Perform 32 loops of decrementing register B.
LD A,(IX + 01)	00DB	DD 7E 01	Load the accumulator with next data byte in table.
OUT (02),A	00DD	D3 02	Output the accumulator to port 2.
LD B,20	00E0	06 20	Load B with a value of 20. (32 in decimal)
DJNZ	00E2	10 FE	Decrement B 32 times.
DEC C	00E4	0D	Decrement C
JR NZ,00D2	00E6	20 E9	If C is NOT zero, jump to line 2 and repeat **0B** times.
INC IX	00E7	DD 23	Increment the IX register
LD A,(IX + 01)	00E9	DD 7E 01	Load accumulator with next byte in table
CP FF	00EB	FE FF	Compare accumulator with FF.
JR NZ,00D2	00EE	20 DE	If accumulator is not FF, jump to start and shift letters across.
JP (HL)	00F0	E9	When FF is detected, micro jumps to address contained in HL.

clock speed is HIGH they will appear to be on at the same time. When the clock speed is LOW, they will produce a flickering effect.

The routine displays the letters for 0B cycles (11 cycles) and then looks at the next byte. If it is FF, the micro jumps back to the main program. If it is not FF, the sub-routine picks up the next byte and displays bytes 2 and 3 on the displays.

A table of names is situated at the end of the sub-routine, which is accessed by the main program and used by the sub-routine.

TABLE OF NAMES:

```
3    4F
     40
CHIP 39
     76
     06
     73
     00
up   1C
     73
     00
BUILT 7C
      3E
      06
      38
      78
      00
BY    7C
      6E
      00
      FF
ANDY  77
      37
      5E
      6E
      FF
BASIL 7C
      77
      6D
      06
      38
      FF
BERT  &c
      79
      33
      78
      FF
BILL  7C
      06
      38
      38
      FF
BOB   7C
      3F
      7C
      FF
BRUCE 7C
      33
      3E
      39
      79
      FF
CARL  39
      77
      33
      38
      FF

CHARLES 39
        76
        77
        33
        38
        79
        6D
        FF
        00
        00
ENTER   79
        37
        78
        79
        33
        00
        06
INPUT   37
        73
        3E
VALUE   78
        00
        3E
        77
        38
        3E
        79
        00
        00
        00
        00
        FF
CLIFF   39
        38
        06
        71
        71
        FF
CLIVE   39
        38
        06
        3E
        79
        FF
CRIS    39
        33
        06
        6D
        FF
COLIN   39
        3F
        38
        06
        37
        FF

CRAIG 39
      33
      77
      06
      3D
      FF
DAVID 5E
      77
      3E
      06
      5E
      FF
DOUG  5E
      3F
      3E
      3D
      FF
ED    79
      5E
      FF
EVAN  79
      3E
      77
      37
      FF
GEORGE 3D
       79
       3F
       33
       3D
       79
       FF
GLEN  3D
      38
      79
      37
      FF
GREG  3D
      33
      79
      3D
      FF
IAN   06
      77
      37
      FF
JOHN  1E
      3F
      76
      37
      FF
PAT   73
      77
      78
      FF

PETER 73
      79
      78
      79
      33
      FF
PHILIP 73
       76
       06
       73
       FF
RALPH 33
      77
      38
      73
      76
      FF
ROY   33
      3F
      6E
      FF
SCOTT 6D
      39
      3F
      78
      78
      FF

STAN  6D
      78
      77
      37
      FF
TONY  78
      3F
      37
      6E
      FF
LITTLE 38
       06
       78
       78
       38
       79
       00
OL    3F
      38
I     00
      06
      FF
???   53
      53
      53
      FF

-     40
-     40
-     40
GUESS 3D
      3E
      79
      6D
      6D
      40
      40
      40
      FF
AN    77
      37
      00
OLD   3F
      38
      5E
      00
PRO   73
      33
      3F
      FF
1     06
9     6F
8     7F
5     6D
      00
      00
      FF
```

The list of names in the table and the corresponding Hex value which must be placed on the input switches. If '8' is on the input, the message will read 'ENTER INPUT VALUE'.

1	ANDY
2	BASIL
3	BERT
4	BOB
5	BRUCE
6	CARL
7	CHARLES
8	ENTER INPUT VALUE
9	CLIFF
A	CLIVE
B	CRIS
C	COLIN
D	CRAIG
E	DAVID
F	DOUG
10	ED
11	EVEN
12	GEORGE
13	GLEN
14	GREG
15	IAN
16	JOHN
17	PAT
18	PETER
19	PHILIP
1A	RALPH
1B	ROY
1C	SCOTT
1D	STAN
1E	TONY
1F	LITTLE 'OL I
20	???
21	- - GUESS - -
22	AN OLD PRO

NUMBERS AND LETTERS

To produce numbers and letters on the displays, you cannot load a data value of **01** and hope to get the figure '1' on the screen. You will get segment 'a' illuminated. This means the hex value of the required segments must be added together to achieve the required figure.

For example, to produce the figure '1', we must turn on segments 'b' and 'c'. The hex value for 'b' is 02 and for 'c' it is 04. Add these together to get 06. To create the figure '2' on the screen, we must illuminate segments a, b, d, e and g. The hex values for these are: 01, 02, 08, 10 and 40. Adding these together we get 5B.

This process has been continued for the alphabet and numbers as shown in the following table.

Some of the letters are hard to create on a 7-segment display and the closest possible resemblance has been created.

```
A  77
B  7C
C  39
D  5E
E  79
F  71
G  3D
H  76
I  06
J  1E
K  75
L  38
M  55
N  37
O  3F
P  73
Q  2F
R  33
S  6D
T  70
U  3E
V  1C
W  1D
X  64
Y  6E
Z  1B
1  06
2  5B
3  4F
4  66
5  6D
6  7D
7  07
8  7F
9  67
0  3F
```

This table gives you the full alphabet and numbers, along with the Hex value needed to produce the character. Most of the letters will be quickly recognised with 'M' and 'W' having a bar over the character to indicate it is repeated again to create the letter.

LOOKING AT DATA

This program lets you look at data in the EPROM. This way you can check each of the programs we have listed.

The program is located at 0200 and is accessed by turning on switch '20'. Push reset to access the program. Page zero address 0000 will be displayed. To access page 1, 2, 3, 4, 5, 6 or 7, the appropriate switches at the input port must be switched ON.

For page 1, turn on switch 1. For page 2, turn on switch 2. For page 3, turn on switches 1 and 2. etc. Switches 8, 10, 20, 40, and 80 are masked OFF via the instruction at **206** and thus they do not affect the page-accessing.

The program is designed to loop around FF bytes and at page '2' the program is capable of reading itself!!

At page zero (or any other page) the program starts by displaying the address value. This will be shown with LOW BRIGHTNESS. Pushing button A will display the value of data at the address. This will be shown with FULL BRIGHTNESS.

Pushing button A again will advance to address 01 and pressing button A again will show the data at this address.

A fast-forward facility is provided by pushing button B when the address value is being displayed. This will enable you to fast-forward around a page to pick up a missed location.

You can select a different page number at any time and the correct data will be displayed.

This program is very handy for reading the contents of the EPROM and proving the data to be as stated.

The display values are generated from a byte table situated at the end of the program and is as follows:

BYTE TABLE at 0280:

0 = 3F
1 = 0B
2 = 5B
3 = 4F
4 = 66
5 = 6D
6 = 7D
7 = 07
8 = 7F
9 = 67
A = 77
B = 7C
C = 39
D = 5E
E = 79
F = 71

All too soon we have run out of space. There are lots more programs in the EPROM and these will be covered in the next issue.

When you buy a kit you will be able to access these programs and see how they work.

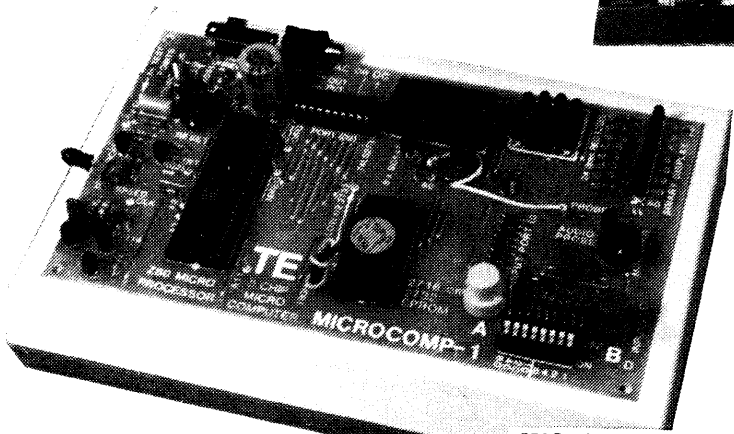
The Microcomp is designed to fit into a cassette case and be stored like a book. Hopefully you will be using it all the time and it won't see the book-shelf. I hope I have encouraged you sufficiently to buy one of the kits. I'm sure it'll be the best decision you will ever make.

```
LD C,00          200  0E 00     C is our TEST register. BIT's are SET or RESET in the program.
LD E,00          202  1E 00     Register E holds the count, from 00 to FF. Zeroed at start.
IN A,(01)        204  DB 01     The value on the switches is loaded into the accumulator.
AND 07           206  E6 07     The accumulator is ANDed with 7 - only 01, 02 & 04 detected.
LD D,A           208  57        The value on the switches (up to 07) is saved in 'D'.
LD A,E           209  7B        The COUNT REGISTER is loaded in the accumulator.
AND 0F           20A  E6 0F     AND 0F removes the 4 upper bits leaving the 4 lower bits.
LD HL 0280       20C  21 80 02  Load HL with the start of the BYTE TABLE.
ADD A,L          20F  85        ADD 80 to the accumulator.
LD L,A           210  6F        A new value for L is created (for later use).
LD A,00          211  3E 00     The accumulator is zeroed.
OUT (02),A       213  D3 02     The accumulator is outputted to port 2.
LD B,10          215  06 10     B is loaded with 10 (16 in decimal)
DJNZ 0217        217  10 FE     DJNZ A delay of 16 is created.
LD A,(HL)        219  7E        The accumulator is loaded with the value pointed
                                   to by HL and outputted to port 2.
OUT (02),A       21A  D3 02
LD A,E           21C  7B        The count register is loaded into the accumulator.
RRA              21D  1F        The accumulator is rotated RIGHT. The 4 high bits
RRA              21E  1F            move down to the 4 lower places and are ANDed
RRA              21F  1F            with 0F .
RRA              220  1F
AND 0F           221  E6 0F     AND 0F removes the 4 upper bits
LD HL,0280       223  21 80 02  HL is loaded with 0280.
ADD A,L          226  85        The L register is ADDed to the accumulator.
LD L,A           227  6F        A new value for L is created.
LD A,00          228  3E 00     Zero the accumulator.
OUT (02),A       22A  D3 02     Output the accumulator to port 2.
LD B,10          22C  06 10     Load B with 10 for a delay routine
DJNZ 022E        22E  10 FE     DJNZ for 16 loops.
LD A,(HL)        230  7E        Load the accumulator with the value POINTED TO by HL.
SET 7,A          231  CB FF     SET bit 7 of the accumulator to turn on display 1
OUT (02),A       233  D3 02     Output the accumulator to port 2.
IN A,(01)        235  DB 01     Look at the input switches
BIT 7,A          237  CB 7F     Test bit 7 to see if switch A is pressed.
JR Z 0243        239  28 08     JUMP if it is not pressed.
SET 1,C          23B  CB C9     SET bit 1 of the C register indicating A pressed.
BIT 2,C          23D  CB 51     Test bit 2 of register C to see if it '1' or '0'.
JR NZ 0204       23F  20 C3     If it is '1', jump to line 3. If it is zero, jump to next loop.
JR 024E          241  18 0B     Jump to start of loop '2'.
RES 2,C          243  CB 91     Reset bit 2 of register C.
BIT 6,A          245  CB 77     Test bit 6 to see if button B is pressed.
JR Z, 0204       247  28 BB     If it is not pressed, jump to line 3.
INC E ,NOP, NOP  249  1C 00 00  Increment register E
JR 0204          24C  18 B6     Jump to line 3.
LD HL,0280       24E  21 80 02  Load HL with start of byte table.
LD A,(DE)        251  1A        Load A with the data pointed to by DE.
AND 0F           252  E6 0F     And the accumulator with 0F.
ADD A,L          254  85        Add register L to the accumulator.
LD L,A           255  6F        Create a new value for L.
LD A,(HL)        256  7E        Load A with the data pointed to by HL.
OUT (02),A       257  D3 02     Output this data to port 2.
LD A,(DE)        259  1A        Load A with the data byte pointed to by DE.
RRA              25A  1F        Rotate the accumulator RIGHT so that the 4 high order bits
RRA              25B  1F            are shifted to the 4 lower positions.
RRA              25C  1F
RRA              25D  1F
AND 0F           25E  E6 0F     AND the accumulator with 0F to remove the 4 upper bits.
LD HL, 0280      260  21 80 02  Load HL with start of DATA TABLE.
ADD A,L          263  85        Add register L to the accumulator.
LD L,A           264  6F        Create a new value for L.
LD A,(HL)        265  7E        Load A with the value pointed to by HL.
SET 7,A          266  CB FF     SET bit 7 of the accumulator to turn on display 1.
OUT (02),A       268  D3 02     Output the accumulator to port 2.
IN A,(01)        26A  DB 01     Look at the switches
BIT 7,A          26C  CB 7F     Detect if button A has been pressed.
JR Z 027B        26E  28 0B     JUMP if button A has not been pressed.
SET 2,C          270  CB d1     SET bit 2 of register C indicating button A pressed.
BIT 1,C          272  CB 49     Test bit 1 of register C to see if button A has been released.
JR NZ 024E       274  20 D8     Jump if bit 1 of register C is '1'.
INC E ,NOP, NOP  276  1C 00 00  Increment register E.
JR 0204          279  18 89     Jump to loop 1.
RES 1,C          27B  CB 89     Reset bit 1 of register C.
JR 024E          27D  18 CF     JUMP to start of loop 2.
```

MICRO COMP

A 3-CHIP Z-80 COMPUTER

$59.95 COMPLETE
COMES WITH FREE STORAGE BOX!!



MICROCOMP CASE $15.00

COMPLETE MICROCOMP MOUNTED ON RETEX CASE RA-1.

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

Part 1

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.

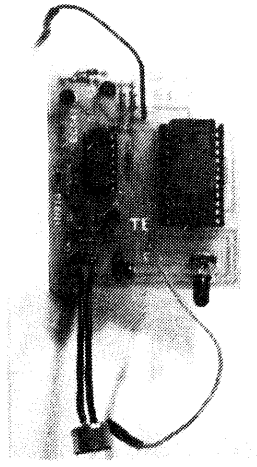
These values are called MACHINE CODE values and are used by the micro directly. It doesn't need spaces or stops and starts as it is pre-programmed within and knows exactly what to do.

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.

The Morse Trainer is our first add-on and will be covered in the next issue. It is capable of picking up morse from a comminications receiver and displaying the message on the displays. It separates the numbers from the letters and indicates the end of words. Speed of reception can be adjusted from 5 words per minute (or less) to about 17 words per minute.

TO BE RELEASED



MORSE TRAINER
$13.30 complete

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.

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

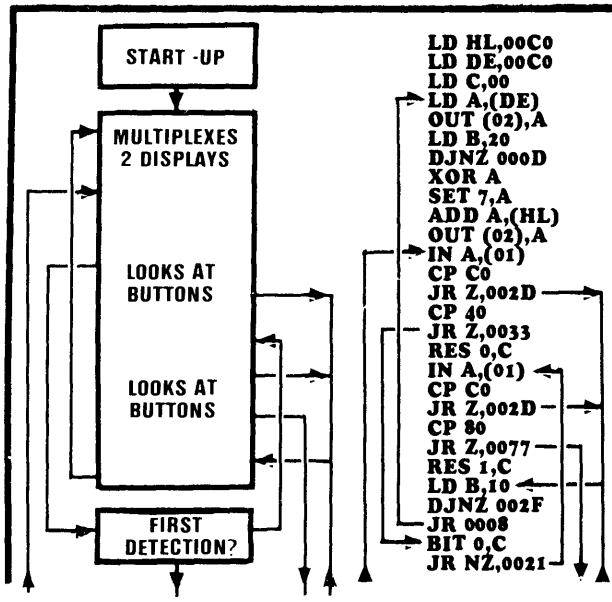
cont. P. 62 . . .

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:**

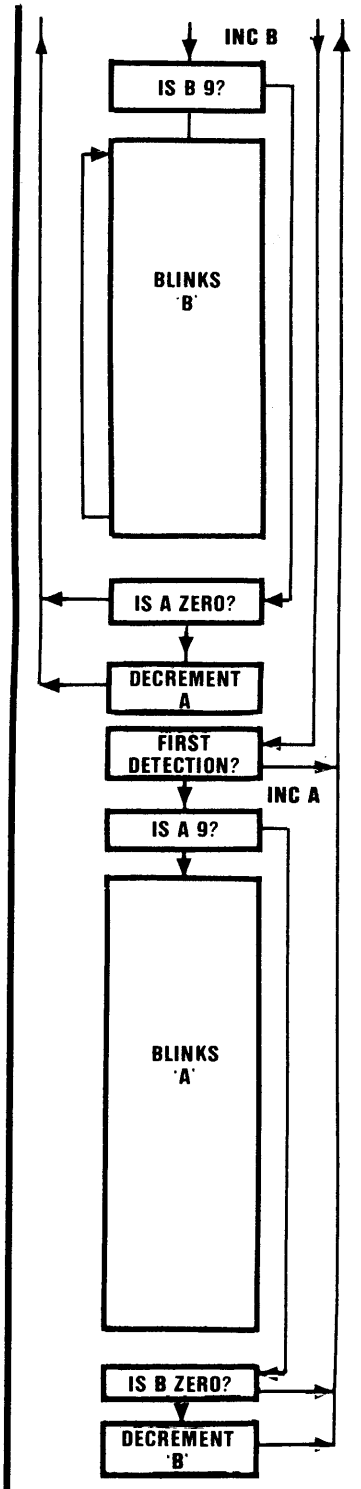
AT C0:

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

TUG O' WAR PROGRAM:



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.




```
SET 0,C
INC DE
LD A,(DE)
CP 67
JR NZ,006D
LD C,10
LD A,(DE)
OUT (02),A
LD B,10
DJNZ 0046
XOR A
SET 7,A
ADD A,(HL)
OUT (02),A
LD B,10
DJNZ 0050
DEC C
JR NZ,0041
LD C,10
XOR A
OUT (02),A
LD B,10
DJNZ 005C
XOR A
SET 7,A
ADD A,(HL)
OUT (02),A
LD B,10
DJNZ 0066
DEC C
JR NZ,0057
JR 003F
LD A,(HL)
CP 3F
JR NZ,0074
JR 0015
DEC HL
JR 0015
BIT 1,C
JR NZ,002D
SET 1,C
INC HL
LD A,(HL)
CP 67
JR NZ,00AE
LD C,10
LD A,(DE)
OUT (02),A
LD B,10
DJNZ 008A
XOR A
SET 7,A
ADD A,(HL)
OUT (02),A
LD B,10
DJNZ 0094
DEC C
JR NZ,0085
LD C,10
LD A,(DE)
OUT (02),A
LD B,10
DJNZ 00A0
XOR A
OUT (02),A
LD B,10
DJNZ 00A7
DEC C
JR NZ,009B
JR 0083
LD A,(DE)
CP 3F
JR NZ,00B6
JP 002D
DEC DE
JP 002D
```

```
0037  CB C1
0039  13
003A  1A
003B  FE 67
003D  20 2E
003F  0E 10
0041  1A
0042  D3 02
0044  06 10
0046  10 FE
0048  AF
0049  CB FF
004B  86
004C  D3 02
004E  06 10
0050  10 FE
0052  0D
0053  20 EC
0055  0E 10
0057  AF
0058  D3 02
005A  06 10
005C  10 FE
005E  AF
005F  CB FF
0061  86
0062  D3 02
0064  06 10
0066  10 FE
0068  0D
0069  20 EC
006B  18 D2
006D  7E
006E  FE 3F
0070  20 02
0072  18 A1
0074  2B
0075  18 9E
0077  CB 49
0079  20 B2
007B  CB C9
007D  23
007E  7E
007F  FE 67
0081  20 2B
0083  0E 10
0085  1A
0086  D3 02
0088  06 10
008A  10 FE
008C  AF
008D  CB FF
008F  86
0090  D3 02
0092  06 10
0094  10 FE
0096  0D
0097  20 EC
0099  0E 10
009B  1A
009C  D3 02
009E  06 10
00A0  10 FE
00A2  AF
00A3  D3 02
00A5  06 10
00A7  10 FE
00A9  0D
00AA  20 EF
00AC  18 D5
00AE  1A
00AF  FE 3F
00B1  20 03
00B3  C3 2D 00
00B6  1B
00B7  C3 2D 00
```

Set bit 0 of C before processing button B.
Increment pointer for RH display.
Load A with second byte in table.
Compare the accum. with 67 to see if end of table has been reached.
Jump if end of table NOT reached. Increment if reached.
Load C with 10 for multiplexing time-length.
Load the accumulator with data pointed to by DE.
Output to the latch.
Load B with 10 for short delay.
Decrement B 16 times.
Zero the accumulator.
Set the highest bit to turn on the LH display.
ADD the byte pointed to by HL, to the accumulator.
Output to the latch.
Load B with 10 for short delay.
Decrement B 16 times.
Decrement C.
Jump if C not zero. Increment if C is zero.
Load C with 10.
Zero A to turn off RH display to create BLINK.
Output to the latch.
Load B with 10 to create a short delay.
Decrement B 16 times.
Zero A.
SET the highest bit of the accumulator to turn on the LH display.
ADD the byte pointed to by the HL pair, to the accumulator.
Out put to the latch.
Load B with 10.
Decrement B 16 times.
Decrement C.
Jump if C not zero.
Jump to start of BLINKING ROUTINE.
Load A with data byte pointed to by HL.
Compare with **3F** to see if LH display is zero.
Jump if not zero.
Jump to start of program if zero.
Decrement player A pointer.
Jump to start of program.
TEST bit 1 of the C register.
Jump if bit 1 is SET. Increment to next instruction if not set.
SET bit 1 of the C register.
Increment player A pointer.
Load the data byte into the accumulator.
Compare the accumulator with 67.
Jump if the two are not the same. Go to next instruction if the same.
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.

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

Load the accumulator with the value pointed to by DE.
Compare the accumulator with 3F to see if the RH display is zero.
Jump if player B is zero, increment to next instruction if not zero.
Jump to start of program.
Decrement player B pointer.
Jump to start of program.

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.

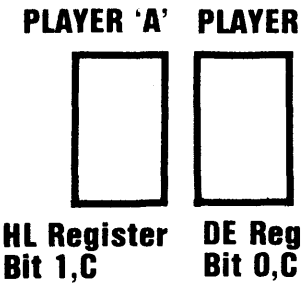
In our program, the register in charge of the table is incremented only after a multiplexing operation (which may involve a number of passes of a loop).

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.

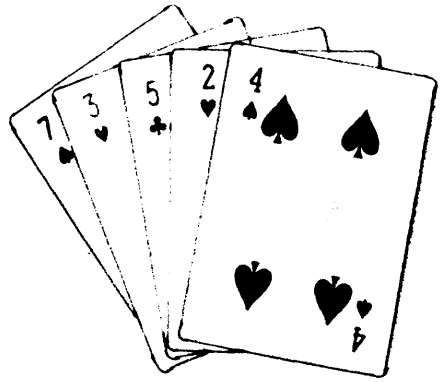


The diagram shows the two displays and their associated register pair. The Debounce is done in register 'C'.

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.

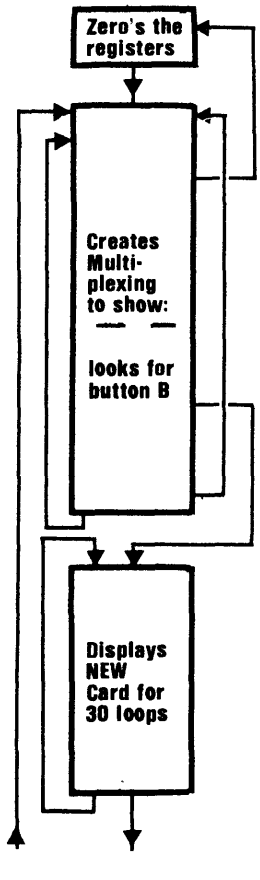
Here's the program: Type it on the TEC, hold it in the non-volatile RAM and play it on the Microcomp.

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

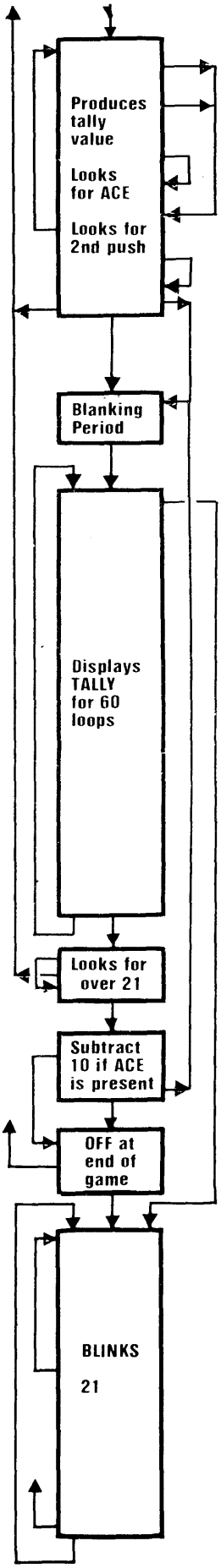
At 0110:
The first two bytes produce the 'CARDS' and the third byte holds the value of the card.

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	




```
XOR A
LD I,A
LD E,A
LD C,A
LD IX 0110
IN A,(01)
CP 40
JR Z,0000
LD IY,0113
LD H,0D
LD A,(IX + 00)
OUT (02),A
LD B,08
DJNZ 001C
XOR A
SET 7,A
ADD A,(IX + 01)
OUT (02),A
IN A,(01)
CP 40
JR Z,003B
LD B,04
DJNZ 002E
INC IY
INC IY
INC IY
DEC H
JR NZ,0015
JR 000F
LD D,30
LD A,(IY + 00)
OUT (02),A
LD B,20
DJNZ 0044
XOR A
SET 7,A
ADD,(IY + 01)
OUT (02),A
LD B,20
DJNZ 0050
DEC D
JR NZ,003D
```

0000	AF	Zero the Accumulator.
0001	ED 47	The I register must be loaded via A. I reg. detects 2nd push of button.
0003	5F	Zero E. Reg E is our tally register to detect '21' etc.
0004	4F	Zero C. Reg C is our TEST register for ACE detection.
0005	DD 21 10 01	Load IX with start of DISPLAY TABLE.
0009	DB 01	Button B must not be pressed when micro passes this point otherwise
000B	FE 40	program will jump to start of routine. This prevent cheating if
000D	28 F1	the button is kept pressed.
000F	FD 21 13 01	Load IY with start of table for displaying value of card.
0013	26 0D	H counts the number of groups of bytes in the table. There are 0D
0015	DD 7E 00	Load the accumulator with the first byte in the table. groups.
0018	D3 02	Output this value to the output latch.
001A	06 08	Load B with a value to produce a short delay.
001C	10 FE	Create 8 loops of decrementing register B.
001E	AF	Zero the accumulator before advancing to the next two operations.
001F	CB FF	SET the highest BIT in the acc. so that the LH display will illuminate.
0021	DD 86 01	ADD the value of the second byte in the table to the accumulator.
0024	D3 02	Output the result to the latch. The LH display will illuminate.
0026	DB 01	Input the value on the switches to the accumulator.
0028	FE 40	Compare the accumulator with '40'.
002A	28 0F	Jump if the accumulator is equal to 40.
002C	06 04	Load B with 04 ready for a short delay.
002E	10 FE	Create 4 loops of decrementing reg B to display the LH digit.
0030	FD 23	Increment the IY register 3 times so that it looks at the start of the next
0032	FD 23	group. This register is our random number generator and increments
0034	FD 23	constantly, while the displays are displaying.
0036	25	Register H will detect the end of the byte table.
0037	20 DC	Jump to displaying RH then LH digit, if H is not zero.
0039	18 D4	When H is zero, IY and IX register go to start of table.
003B	16 30	D will govern the length of time for displaying the random number.
003D	FD 7E 00	The accumulator is loaded with the display value for the random No.
0040	D3 02	This value is outputted to port 02.
0042	06 20	The RH display will illuminate for a delay determined by the value of B.
0044	10 FE	
0046	AF	The accumulator is zeroed ready for the next two instructions.
0047	CB FF	Bit 7 is SET to turn on the LH display.
0049	FD 86 01	The value of the second byte in the group is added to the accumulator
004C	D3 02	and outputted to port 02.
004E	06 20	The LH display is illuminated for a period of time as determined by the
0050	10 FE	value of B.
0052	15	D is decremented by one and the program loops again.
0053	20 E8	When D is zero, the micro advances to the next instruction.




```
LD A,(IY + 02)
INC E
BIT 1,E
JR Z,0066
BIT 3,E
JR Z,0066
LD B,06
INC E
DJNZ 0063
DEC A
JR NZ,0058
LD A,(IY + 00)
CP 77
JR NZ,0072
SET 7,C
LD A,I
INC A
LD I,A
CP 02
JR NC,007D
JR 0009
LD D,60
XOR A
OUT (02),A
LD B,FF
DJNZ 0084
LD HL,0100
LD A,E
CP 21
JR Z,00D4
AND 0F
ADD A,L
LD L,A
LD A,(HL)
OUT (02),A
LD B,10
DJNZ 0097
LD A,E
RRA
RRA
RRA
RRA
AND 0F
LD HL,0100
ADD A,L
LD L,A
LD A,(HL)
SET 7,A
OUT (02),A
LD B,08
DJNZ 00AC
DEC D
JR NZ,0086
LD A,E
CP 22
JR NC,00B9
JP 0009
BIT 7,C
JR Z,00C4
SUB 10
LD E,A
RES 7,C
JR 007D
XOR A
OUT (02),A
LD B,FF
DJNZ 00C9
DJNZ 00CB
DJNZ 00CD
DJNZ 00CF
JP 0000
LD C,10
LD A,06
OUT (02),A
LD B,10
DJNZ 00DC
LD A,DB
OUT (02),A
LD B,10
DJNZ 00E4
DEC C
JR NZ,00D6
XOR A
OUR (02),A
LD B,FF
DJNZ 00EE
IN A,(01)
CP 40
JP Z,0000
JR 00D4
```

```
0055  FD 7E 02  Load A with the 3rd byte in the group. We know the byte must have a
0058  1C        value of one or greater and so we can safely INCrement E.
0059  CB 4B     Register E is our TALLY register. We require it to add the values of the
005B  28 09     cards and hold the result in decimal form. The problem comes when
005D  CB 5B     you add one to 9. The register will show 0A. We must convert 0A to ten.
005F  28 05     This can be done by a DAA instruction or by software. We have opted
0061  06 06     for software. We detect 0A via bit 1 and 3 being HIGH and then
0063  1C        increment the E register 6 times. Each time the tally register is
0064  10 FD     incremented (apart from the decimal adjusting loop), the accumulator
0066  3D        is decremented and when the accumulator is zero, the program
0067  20 EF     advances.
0069  FD 7E 00  Load the accumulator with the first byte of the group.
006C  FE 77     Compare 77 with the accumulator. We are looking for an ACE.
006E  20 02     If the accumulator is not 77, the micro will jump to LD A,I. If the
0070  CB F9     accumulator is 77, the program will advance to the next instruction
0072  ED 57     and SET bit 7 of the C register.
0074  3C        The I register counts the number of presses of the B button. We are
0075  ED 47     looking for 2 or more presses so that the tally can be displayed. This is
0077  FE 02     the advantage of using the CARRY command.
0079  30 02     The micro jumps when I is 2 or MORE.
007B  18 8C     Jump to start of program (button B has been pressed once.)
007D  16 60     Register D produces the time for the tally to appear.
007F  AF        Blank the display.
0080  D3 02     Output to latch.
0082  06 FF     Load B with maximum delay value.
0084  10 FE     Perform FF loops of decrementing register B.
0086  21 00 01  Load HL with start of display values.
0089  7B        Load the tally register into the accumulator.
008A  FE 21     Compare 21 with the accumulator.
008C  28 46     If accumulator is 21, the micro jumps to 'BLINKING 21'
008E  E6 0F     If not 21, remove high nibble by ANDing with 0F.
0090  85        L contains '00' from address above, ADD 00 to accumulator.
0091  6F        Load the result back into L so that the micro looks at one of the
0092  7E        addresses of the table. Load the value it finds, into A.
0093  D3 02     Output the byte to the latch.
0095  06 10     Load B with a low value.
0097  10 FE     Create 16 loops of decrementing register B.
0099  7B        Load the tally register into the accumulator.
009A  1F        Rotate the accumulator right, effectively bringing the 4 bits of the
009B  1F        HIGH nibble to occupy the 4 lower places.
009C  1F
009D  1F
009E  E6 0F     Remove the 4 high bits by ANDing with 0F.
00A0  21 00 01  Load the HL register with start of display table.
00A3  85        ADD L to accumulator to create a new value for L so that we look at one
00A4  6F        of the addresses in the table.
00A5  7E        Load the byte from the table into the accumulator.
00A6  CB FF     Set bit 7 of the accumulator so that the LH display turns on.
00A8  D3 02     Output this value to the latch.
00AA  06 08     Load B with a short delay value.
00AC  10 FE     Create 8 loops of decrementing register B.
00AE  15        Decrement D and go to start of multiplexing loop. When D is zero,
00AF  20 D5     increment to next instruction in program.
00B1  7B        Load the tally register into the accumulator.
00B2  FE 22     Compere with 22.
00B4  30 03     If tally is 22 or MORE, increment to next instruction.
00B6  C3 09 00  Jump to start of program. If tally is less than 22, jump to BIT 7,C.
00B9  CB 79     Test bit 7 of the C register to see if an ACE has been dealt.
00BB  28 07     Jump if no ACE. Increment if an ACE is present.
00BD  D6 10     Subtract ten from tally, making ACE equal to ONE.
00BF  5F        Load the new tally into the tally register.
00C0  CB B9     Reset bit 7 to show ACE has been turned into ONE.
00C2  18 B9     Jump to displaying new tally.
00C4  AF        Zero the accumulator.
00C5  D3 02     Output to latch for a delay period equal to 4 DJNZ's (with B =FF) to
00C7  06 FF     indicate END OF GAME.
00C9  10 FE
00CB  10 FE
00CD  10 FE
00CF  10 FE
00D1  C3 00 00  Jump to start and re-load all registers.
00D4  0E 10     Load C with 10 for 16 loops of multiplexing '1' and '2'.
00D6  3E 06     Load A with 06 to create '1' on RH display.
00D8  D3 02     Output this to latch.
00DA  06 10     Create short delay.
00DC  10 FE     Decrement B to zero.
00DE  3E DB     Load the accumulator with DB to create '2' in LH display.
00E0  D3 02     Output to latch.
00E2  06 10     Create short delay.
00E4  10 FE     Decrement B to zero.
00E6  0D        Decrement C and if not zero, jump to start of multiplexing the displays.
00E7  20 ED
00E9  AF        Zero A
00EA  D3 02     Output to latch to turn off both displays.
00EC  06 FF     Load B with FF to produce a short delay for the OFF time.
00EE  10 FE     The only way of jumping out of 'BLINKING 21' is to push button B or
00F0  DB 01     reset the computer. The program inputs from the set of buttons and if
00F2  FE 40     B is pressed, the program jumps to 0000. Otherwise the program
00F4  CA 00 00  keeps looping.
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.

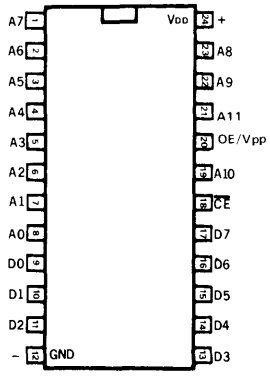
Burning a program is only done after you are thoroughly satisfied with its performance, as it is very difficult (if not impossible) to change the program, once it is burnt. For this reason it is best to get the program up and running via a medium which can be easily altered, as a program quite often 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 important 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.

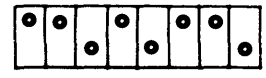
Tables should be placed well away from the program so that you don't run out of room. When everything works perfectly, they can be moved up and the pointers changed accordingly.

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.

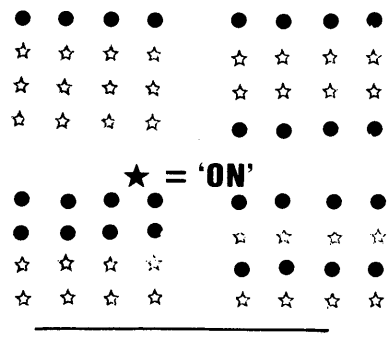
This means it is rapidly looping around the program and will update the displays as soon as the input values are changed.

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).

Create these effects on the 4x4 matrix:



FROM INPUT PORT TO 8LEDS

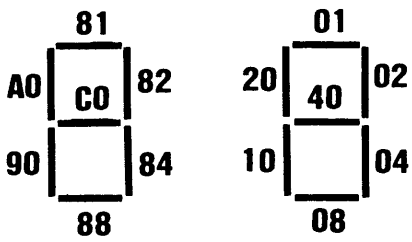
```
IN A,(01)     0290   DB 01   Looks at input switches and places the value in the accumulator.
OUT (02),A    0292   D3 02   Outputs accumulator to the latch.
JR 0290       0294   18 FA   Jumps to start of program.
```

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.

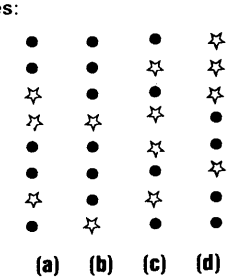
Button 'A' is connected to **80** and will make the figures jump from one display to the other.

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.

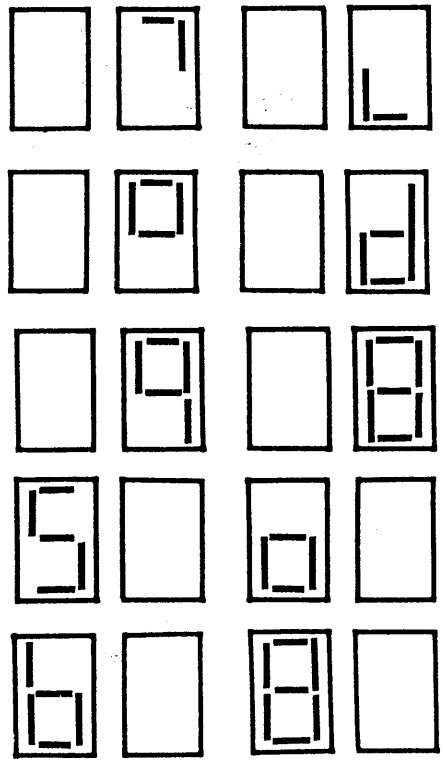
There are some limitations as to what combinations of LEDs can be turned on and this is something you must be aware of.

Experiments:

Create these effects by using the input switches:

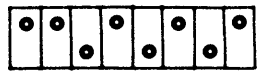


Create these on the 7-segments displays:



INCREMENT via BUTTON A

This program at 02A0 increments the display each time button A is pressed.

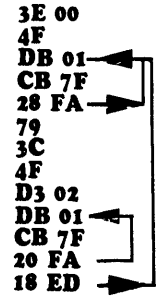


```
LD A,00       02A0   3E 00   Load the accumulator with zero.
LD C.A        02A2   4F      Load zero into C.
IN A,(01)     02A3   DB 01   Input the value on the switches to the accumulator.
BIT 7,A       02A5   CB 7F   Test BIT 7 of the accumulator to see if button A is pushed.
JR Z 02A3     02A7   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.

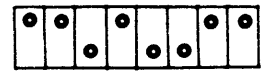
The program contains two small loops and the micro is constantly executing the top one when button A is not pressed and the lower one when the button is pressed. The micro jumps from one loop to the other during the time when the button is travelling from one state to the other.



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.

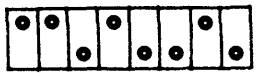
Notice that they produce entirely different effects due to the placement of the LEDs and this can be remembered when designing displays for advertising etc.

```
LD A,00       02C0   3E 00
INC A         02C2   3C
OUT (02),A    02C3   D3 02
DJNZ 02C5     02C5   10 FE
DJNZ 02C7     02C7   10FE
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.

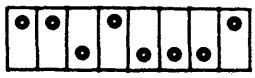
Load the value **01** and compare it with **00. 00** is a much longer delay and it appears to be as long as **FF!** In fact this is the case! The longest delay is produced when a register is loaded with **00** since the first operation to be performed on the register is to decrement it. The result is **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

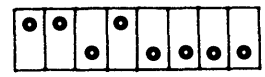
counter. The best effect of decrementing can be seen on the 8 LEDs. Adjust the speed control to view the effect in slow motion.

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.

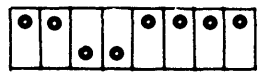


02F0

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

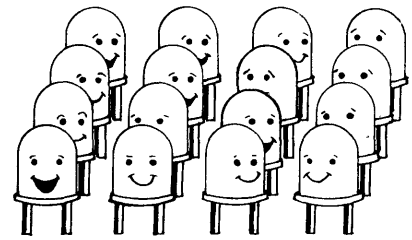
will produce almost no interpretable effects on either of the other displays.

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.

```
LD B,08          0300   06 08
LD HL,0338       0302   21 38 03
LD C,18          0305   0E 18
DEC C            0307   0D
JR Z,0318        0308   28 0E
LD A,(HL)        030A   7E
OUT (02),A       030B   D3 02
INC HL           030D   23
LD DE,0080       030E   11 80 00
DEC DE           0311   1B
LD A,D           0312   7A
OR E             0313   B3
JR NZ,0311       0314   20 FB
JR 0307          0316   18 EF
DJNZ 0302        0318   10 E8
LD B,08          031A   06 08
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
```

B is the COUNT REGISTER for the number of loops in the first program.
Load HL with the address of the start of the BYTE TABLE.
Load C with the number of bytes for the program (There are 24 bytes.)
Decrement the number of bytes remaining in the table to detect the end of table.
If no bytes remain, decrement the number of loops and start program again.
Load the accumulator with the byte pointed to b the HL register pair.
Output this value to port 2.
Increment HL to point to the next byte in the table.
Load DE with a short delay value.
Decrement DE.
Load D into A.
Logically OR the accumulator with E to see when BOTH D and E arezero.
Jump to **0311** if the answer is NOT ZERO.
Jump to **DEC C** and repeat for the second byte in the table.
Decrement the number of loops and start the byte table again.
Load B with 8 for the second part of the program.

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.

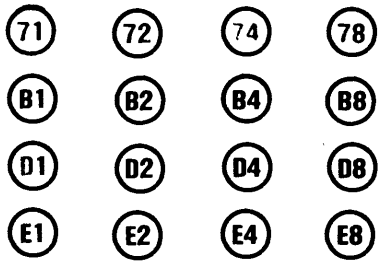
At 0338:		At 0350:		
01	CF	0F	B8	D4
02	3F	FF	D8	D2
04	CF	0F	E8	B2
08	3F	FF	E4	B4
EF	96	0F	E2	D4
DF	FF	FF	E1	D2
BF	96	0F	D1	B2
7F	FF	FF	B1	B4
03	33	71	71	
0C	CC	72	72	
03	C3	74	74	
0C	3C	78	B4	

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:



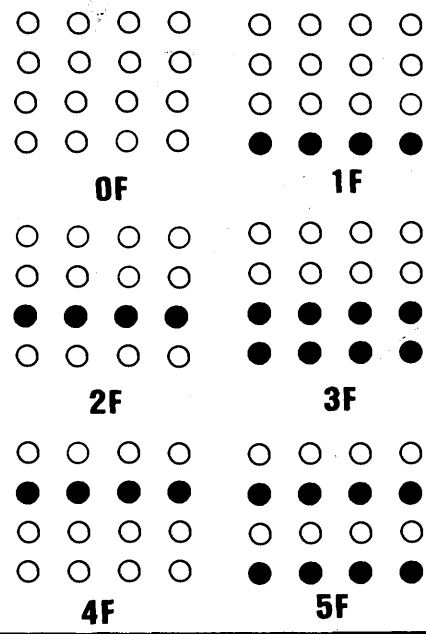
To access the LEDs we have separated the output latch into two halves with the 4 lower bits connected to the anodes and the 4 upper bits to the cathodes.

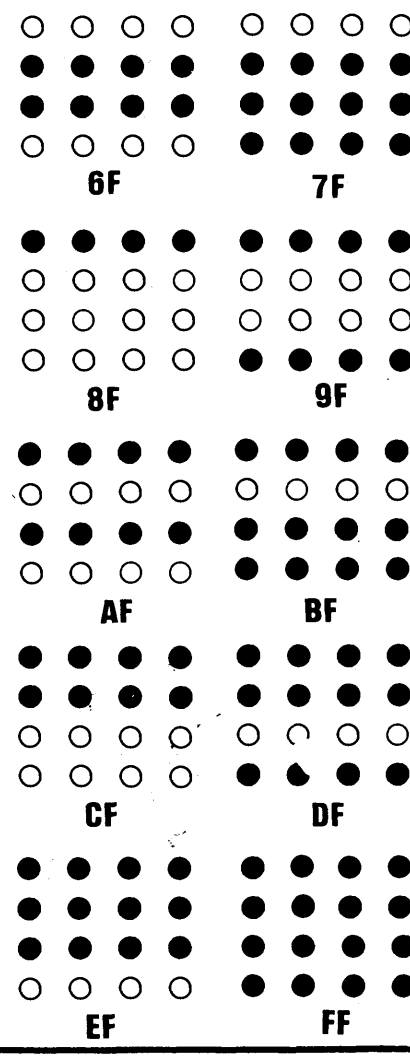
The following diagrams give you the values required to turn ON one or more LEDs:

ALL ON "0F"
ALL OFF "FF" or "00".

LEGEND:

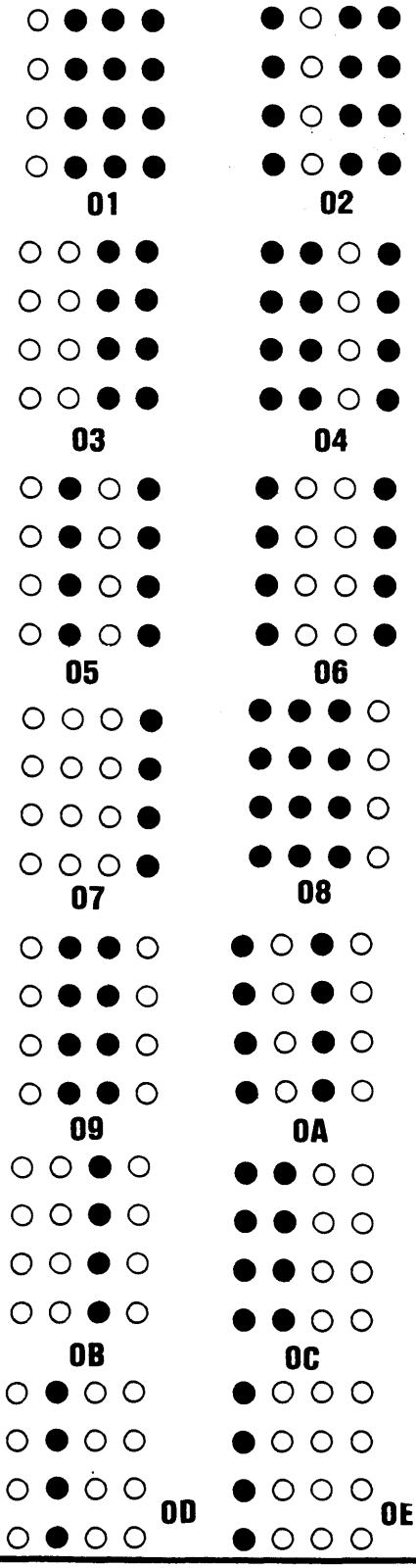
○ = ON.
● = OFF.

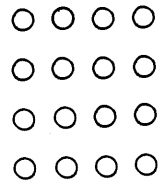




If you don't want all the LEDs in a row to be illuminated, refer to the diagrams on this page for the hex value needed to illuminate an individual column or column(s).

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).



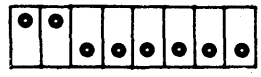


When the two diagrams are placed on top of each other, the LEDs that are common to both, will be illuminated. Due to the sinking and sourcing limitations of the output latch, all the LEDs in the 4x4 can not be illuminated at the same time.

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.**

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.**

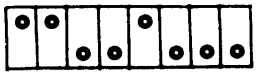
DJNZ's can be grouped thus:

```
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.

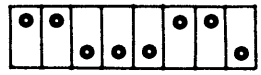
Because the program is running at high speed, we must include a DEBOUNCE feature to prevent more than ONE COUNT being registered when a button is pressed.

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.

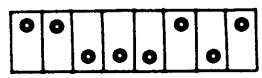


The main program for producing the letters on the display is located at **03A8** and the micro jumps to this address via the instruction **JR 03A8.** The main program is also used by the **A -Z, 0 - F** counter and shows how the same table and output program can be accessed by two different START-UP 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

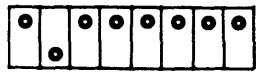
LD C,0A	0370	0E 0A	Register C is the counter for the BYTE TABLE. There are ten bytes.
LD DE,03DF	0372	11 DF 03	The DE register pair is loaded with the start-address of the byte table.
IN A,(01)	0375	DB 01	The input latch is looked at and the value it holds is placed into the accumulator.
BIT 7,A	0377	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.

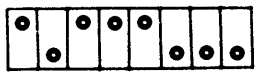
This is necessary to keep the displays illuminated while at the same time preventing the program from incrementing the displays if a button is kept pressed.

```
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 042A. 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
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
```

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.

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	E(
72	E4
74	E2
78	E1
B8	D1
D8	B1

```
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.
```

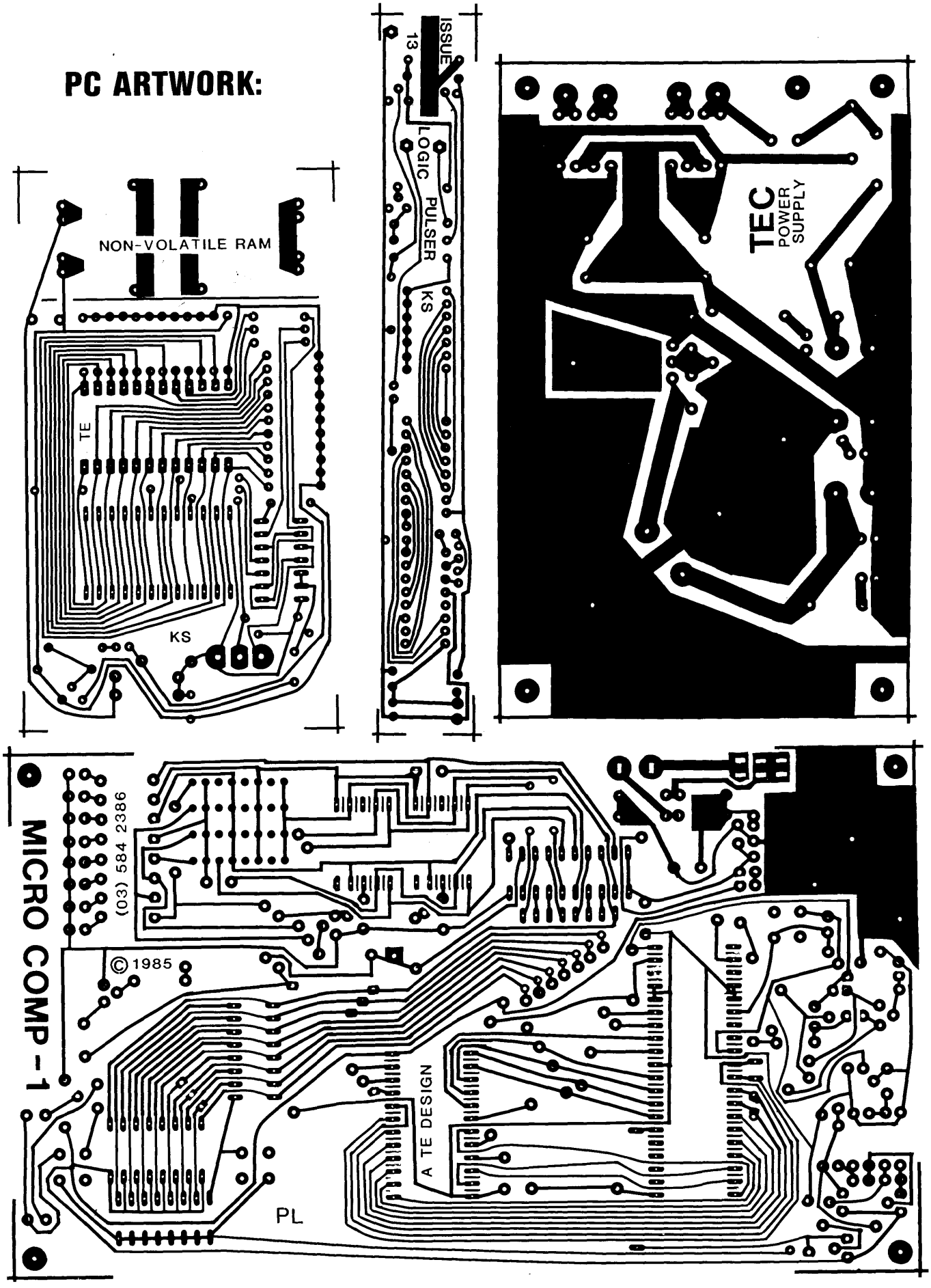
At 04E0:

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.

B4	D2	72	52	52	52
00	00	B4	00	B4	54
00	78	D8	58	58	58

```
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.
```

PC ARTWORK:


NON-VOLATILE RAM
TE
KS
ISSUE
13
LOGIC PULSER
KS
TEC
POWER
SUPPLY
MICRO COMP-1
(03) 584 2386
©1985
A TE DESIGN
PL