and emitter terminals of the BD 679 and the interior light does not produce its full brightness, however it is almost as bright as when the door was open.

We use this naturally-occurring 3v to do two things.

Firstly we use it to provide Q_2 with a voltage to drive the base of the BD 679.

At the moment we have 3v - .65v - .65v = 1.7v from collector to emitter of Q_2 to do this. This is just enough for the transistor to work and although it will not give a high value of gain, it will provide enough for our situation.

We tried connecting the base of the BD 679 to the power rail (of the circuit) via a 100k resistor, but this did not provide enough base current to turn the transistor on.

The reason for this is two-fold. We have already mentioned in other articles that the current required to turn on a lamp is up to 6 times more than its operating current and when a heavy current is required, the gain of the BD 679 transistor is reduced considerably. It is only about 1,000 in our case.

Secondly, the voltage between the base of the BD 679 and the rail is 1.7v, when the supply is 3v, so a 100k resistor does not allow enough current to flow in to the base to turn it on fully.

We had to reduce the resistor to 4k7 to get it to turn on fully.

Instead of using a very low resistor, like 4k7, the alternative is to add a transistor to make a triple super-alpha arrangement and this is the function of Q_2.

Going over the action of the circuit again; the door switch discharges the 22u via the signal diode so that when the switch opens, the electrolytic has only about .6v on it and via the voltage divider, the base of Q_1 does not have sufficient voltage on it to turn it on. The only components to be turned on are Q_2 and the BD 679.

The 3v rail voltage begins to charge the 22u via the 1M resistor and when the voltage on the electrolytic is 1.2v, the voltage at the middle of the 2M2 voltage-divider resistors is .6v and Q_1 begins to turn on.

This action robs Q_2 of base current and both Q_2 and the Darlington transistor begin to turn off.

This causes the interior light to fade and the voltage across our circuit increases.

This gives the circuit more voltage with which to charge the 22u and very soon the voltage and current at the centre of the voltage divider resistors is enough to fully turn on Q_1.

This turns off Q_2 and Q_3 and the interior light goes out completely. The consumption of the circuit drops to a few microamps, through resistors R_1 and R_2 and that is how it sits, ready for the door to be opened again.

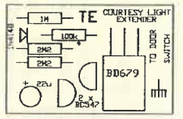
ASSEMBLY

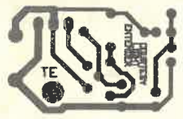
This project uses very few components, however you must be careful with fitting them as they may be damaged if they are around the wrong way when the power is applied.

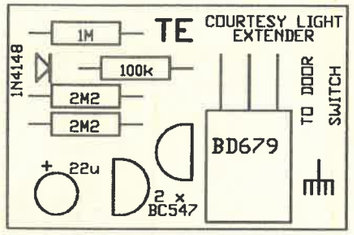
Check the overlay on the PC board for the correct placement (you can use the enlarged layout to help you). The BD 679 does not need a heatsink as it is operating for only a short period of time.

FITTING

This project is very easy to fit to a car. All you have to do is connect one lead to the door-switch and the other to the chassis.







The delay can be increased by changing the 22u to 47u or the 1M to 1M5.

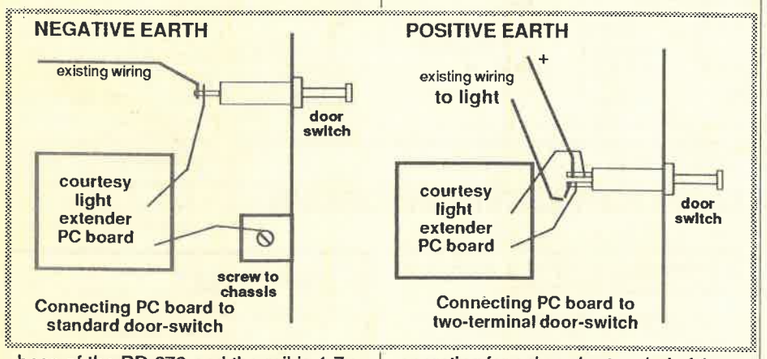
Or it can be decreased by changing the 1M to a lower value, such as 820k.

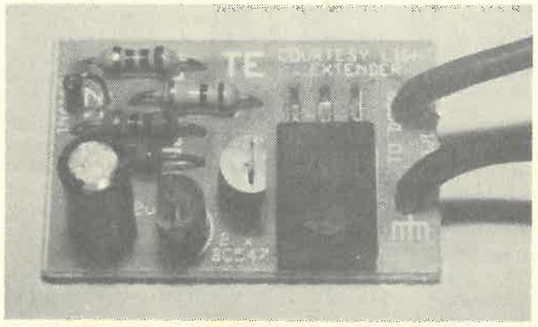
IF IT DOESN'T WORK

If the circuit fails to work, make sure you have connected it correctly to the door switch and chassis.

If you are not satisfied with the timing, set up a test-jig by connecting a 5 watt lamp to the board and use a 12v car battery for the supply. This will allow you to take voltage measurements and change the components until the timing is correct.

You can safely experiment with the circuit like this as the lamp will limit the current. Once you get it working perfectly, you can fit it to the car. But it must not be allowed to "float around." Either heatshrink the board or fit it into a small box. Then fix the box to the chassis with a small piece of double sided tape.





Completed Courtesy Light Extender Project

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 micro-processor system with very little effort and undertanding. But its dis-advantage 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
8 - 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

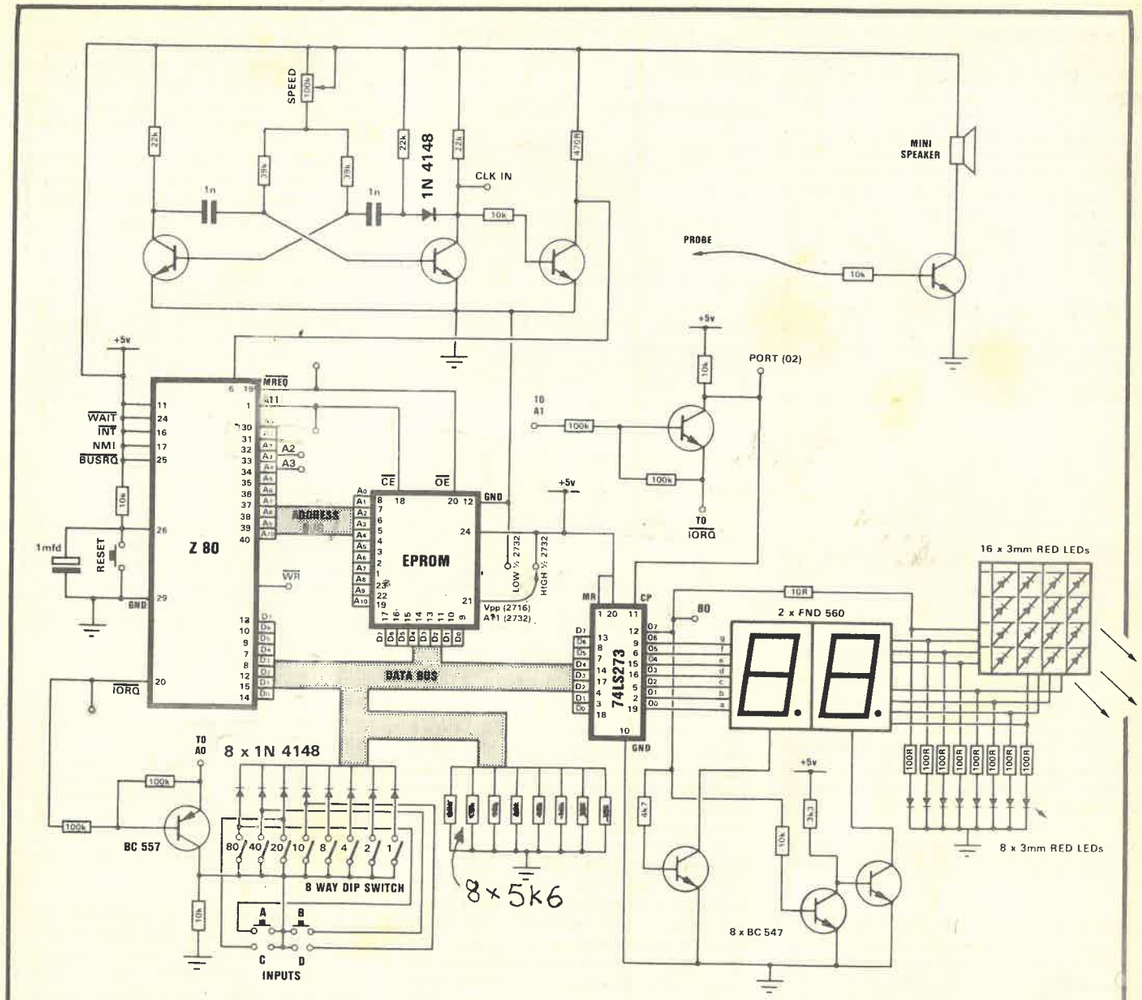
1 - 74LS273 IC
1 - Z-80 CPU
1 - 2732 EPROM (PROGRAMMED)
1 - 7805 regulator

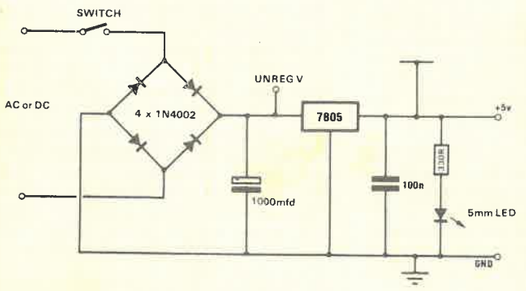
1 - 20 pin IC socket
1 - 24 pin IC socket
1 - 40 pin IC socket
1 - 8 way DIP switch
1 - DPDT slide switch
3 - PC mount push switches

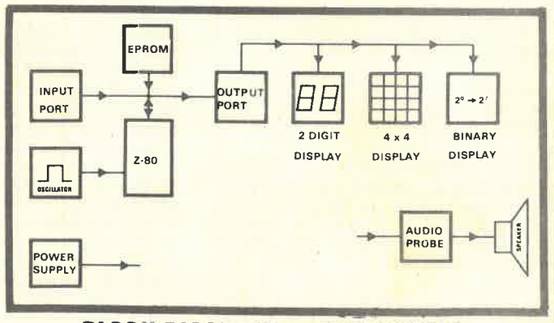
1 - 3.5mm mono socket
1 - mini speaker

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.

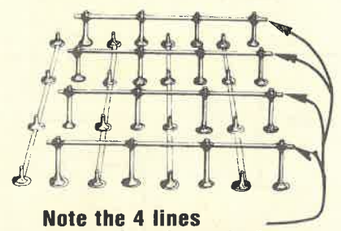
All the components are marked on the PC board and it is possible to build the project without any other help. But as a guide we will go through the assembly and explain everything as we go.

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.

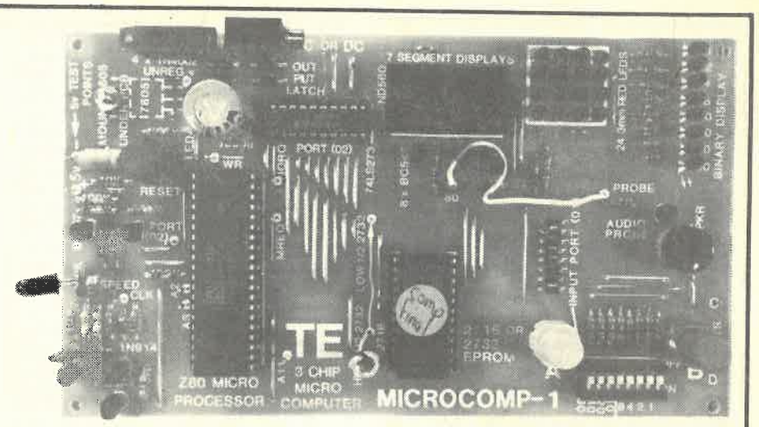
Fit 4 rubber feet to the underside of the board, insert the chips and you are ready for testing.

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.

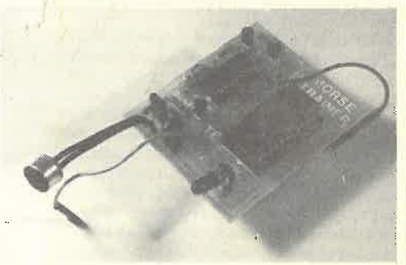
If you do not get 99 on the displays you may have a fault in the system. This will require you going through a trouble-shooting procedure as covered on P. 66.

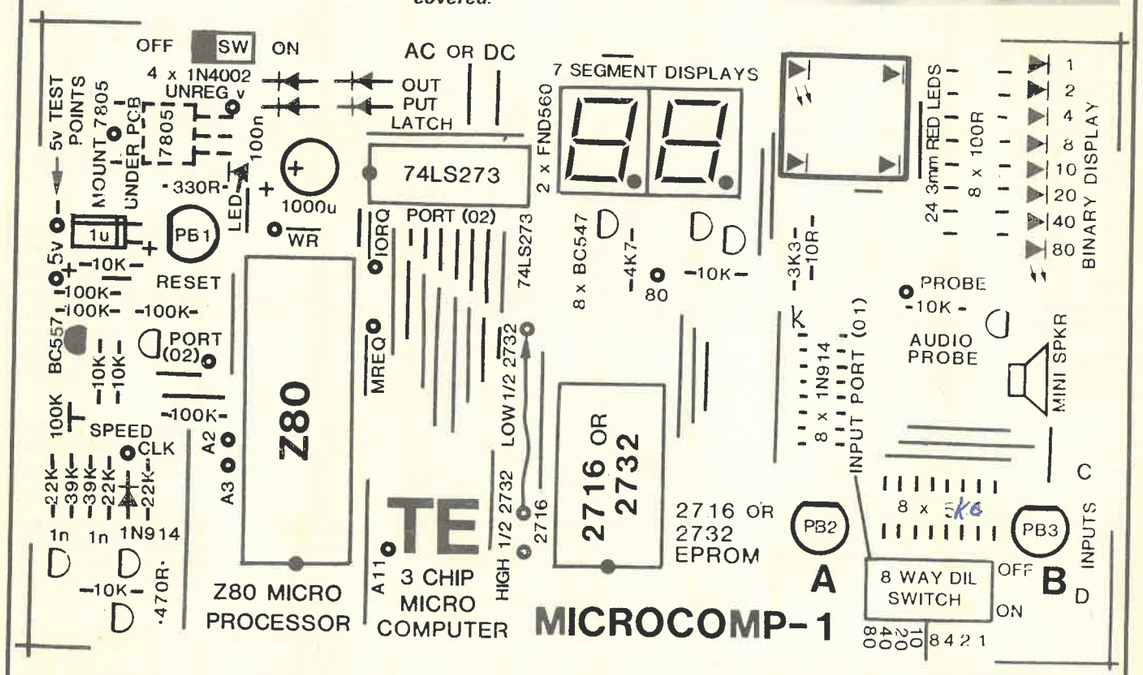
Consider yourself lucky that the computer doesn't work. You will gain a lot by trouble-shooting it yourself and gain experience in finding the fault.



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:

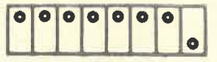
1. Set the switches to address values which are not the start addresses of a program. Why do some of them work?
2. Why does button B address the start of the 00 - 99 counter?
3. Could the DIP switches be replaced with push buttons?
4. Explain what we mean by the input switches are software programmed: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

0020

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.

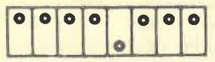
Press the reset button to start a new game.

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.



0080

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 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.

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.

TABLE OF NAMES:

3	4F	C	39	C	39	P	73	S	6D	.	40
	40	H	76	R	33	E	79	T	78	.	40
C	39	A	77	A	77	T	78	A	77	.	40
H	76	R	33	I	06	E	79	N	37	G	3D
I	06	L	38	G	3D	R	33		FF	U	3E
P	73	E	79		FF		FF	T	78	E	79
	00	S	6D	D	5E	P	73		3F	S	6D
u	1C		FF	A	77	H	76	O	37	S	6D
P	73		00	V	3E	I	06	L	FF		40
	00		00	I	06	L	1E	I	6E		40
B	7C	E	79	D	5E	I	75	T	FF		FF
U	3E	N	37		FF	P	38	T	38	A	77
I	06	T	78	O	3F		06	L	55	N	37
L	38	E	79	U	3E	R	78	E	37	O	00
T	78	R	33	G	3D		78		6F	L	3F
	00		00		FF	A	77	L	7F	D	38
B	7C	I	06	E	79	N	37		38		5E
Y	6E	N	37	V	3E	D	79	I	79	P	00
	00	P	73	A	77		FF	T	00	R	73
	FF	U	3E	N	37	R	33	T	3F	O	33
A	77	T	78		FF	O	3D		3F		3F
N	37		00	G	3D	B	33	O	FF		FF
D	5E	V	3E	E	79	E	79	L	06	1	06
Y	6E	A	77	O	3F	R	33	D	6F	9	6F
	FF	L	38	R	33	T	77		7F	8	7F
B	7C	U	FF	G	3D		38	O	6D	5	6D
A	6D	E	79	E	79	P	73	L	00		00
S	06		00	O	3F	H	76	D	00		00
I	38		00	R	33	I	06		FF		FF
L	FF		00	G	3D	L	1E				
	&c		00	E	79	L	75				
B	79		FF		FF	I	38				
E	33	C	39	G	3D	T	55				
R	78	L	38	L	3D	T	37				
T	FF	I	06	E	38	E	3F				
	7C	F	71	N	79		73				
B	06		FF		37	R	2F				
I	38	C	39	G	FF	O	33				
L	38	L	38	R	3D	B	6D				
L	FF	I	06	E	79	E	70				
	7C	V	3E	G	3D	R	3E				
B	3F	E	79	L	FF	T	1C				
O	7C		FF	E	3D		1D				
B	FF	C	39	N	79	W	64				
	7C	L	38		37	X	6E				
B	6E	I	06	G	FF	Y	6E				
R	00	V	3E	R	3D	Z	1B				
U	FF	E	79	E	79	1	06				
C	7C		FF	G	3D	2	5B				
E	6E	C	39	R	33	3	4F				
	00	L	38	E	79	4	66				
C	FF	I	06	G	3D	5	6D				
A	7C	V	3E		FF	6	7D				
R	6E	E	79	I	06	7	07				
L	00		FF	A	77	8	7F				
	FF	C	39	N	37	9	67				
	39	L	38		FF	0	3F				
	77	I	06	J	1E						
	33	V	3E	O	3F						
	38	E	79	H	76						
	FF		FF	N	37						
		C	39		FF						
		O	3F	P	73						
		L	38	A	77						
		I	06	T	78						
		N	37		FF						
			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 bookshelf. 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.
```

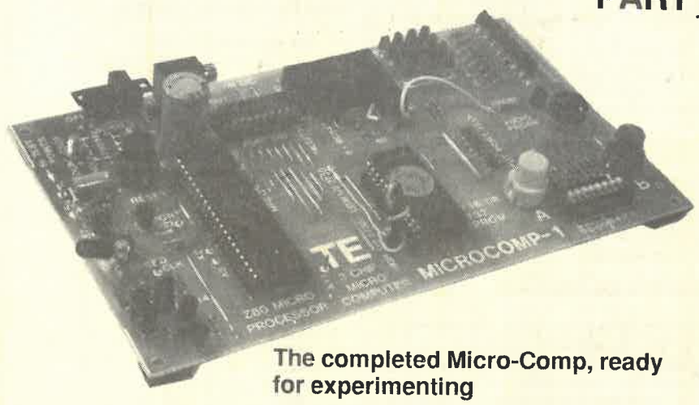

MICRO COMP

A 3-CHIP Z-80 COMPUTER

$72.30
Parts and PC board
You also need a plug pack

The Micro-Comp in its storage box.

PART II



The completed Micro-Comp, ready for experimenting

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

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

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

By now you will have some idea of how the bytes appear in EPROM. They come in a continuous string - without spaces or identification as to the beginning or end of a sequence. If you jump into the middle of a program and look at a byte, you will not know if it is an instruction, part of an instruction or a piece of data. That's why you must start at the beginning of a listing.

When trying to disect a program, write down the values, byte by byte and you will soon see groups which you recognise. From there you can place the others in groups and start to see a program emerging.

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

The difficulty you would experience in disecting a program is understandable. You are not a micro and cannot keep track of the flow of the program. This is a very difficult direction to work in. The way we will be working is from IDEA-to-machine-code-listing. This is the forward direction and is much easier.

Most programs are made up of lots of small building blocks and the quickest way to learn about these is to study a few programs.

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

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

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

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

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

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

TUG O' WAR & BLACK JACK

TWO programs for the MICROCOMP.

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

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

TUG O' WAR

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

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

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

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

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

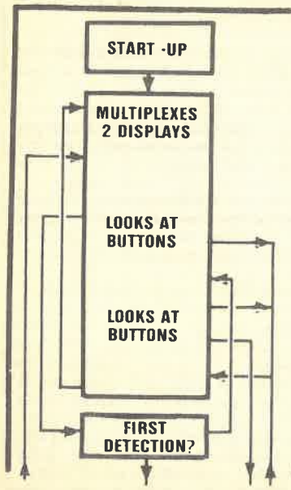
cont. P. 56...

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:**

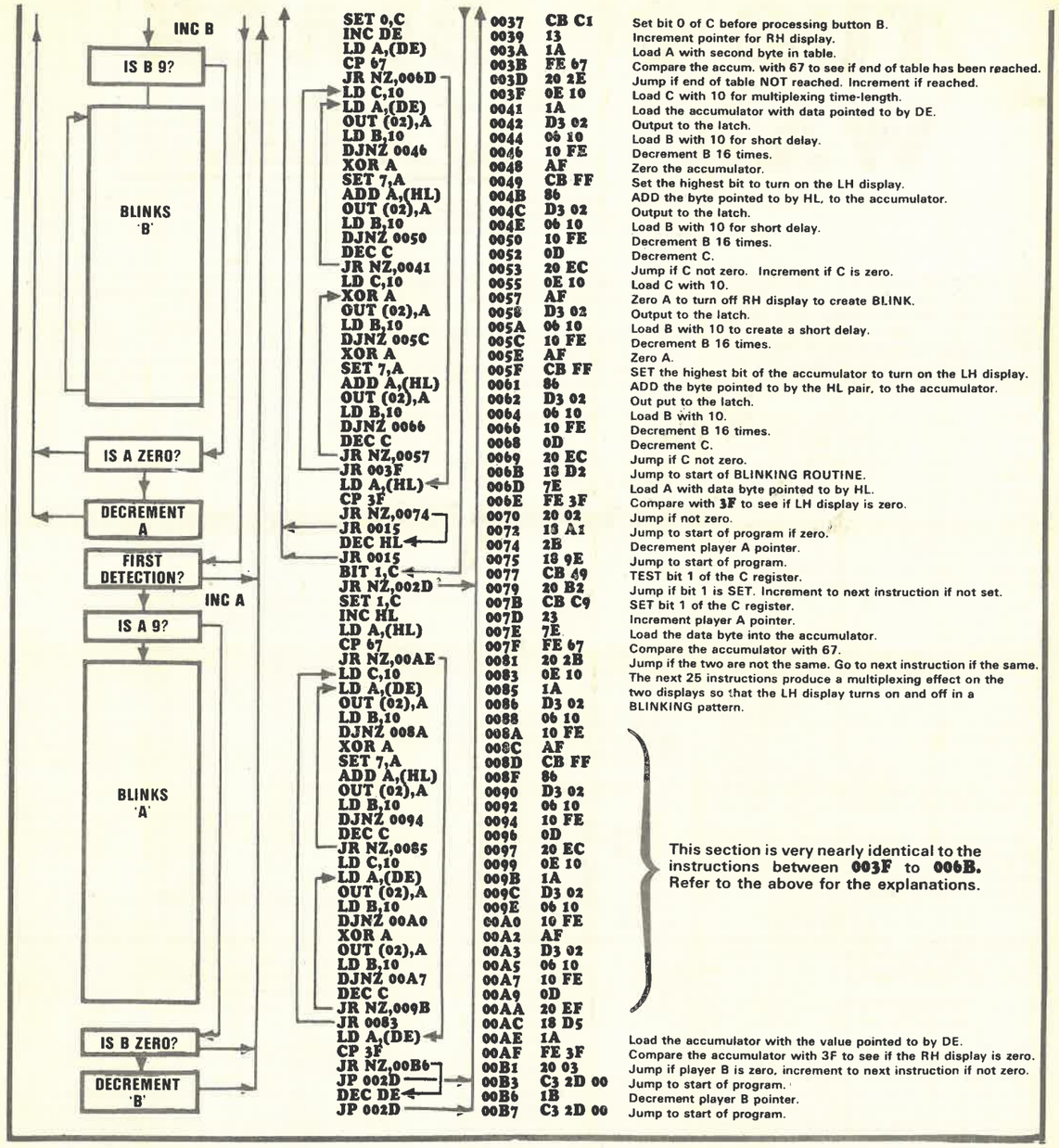
AT C0:

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	07
66	7F
66	7F
6D	7F
6D	7F
6D	7F
6D	7F
6D	7F
6D	7F
7D	7F
7D	07

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

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.

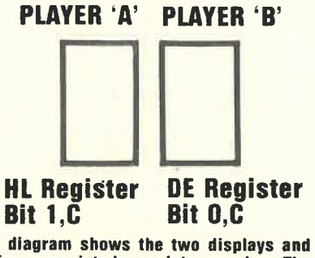
In our program, the register in charge of the table is incremented only after a multiplexing operation (which may involve a number of passes of a loop).

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.

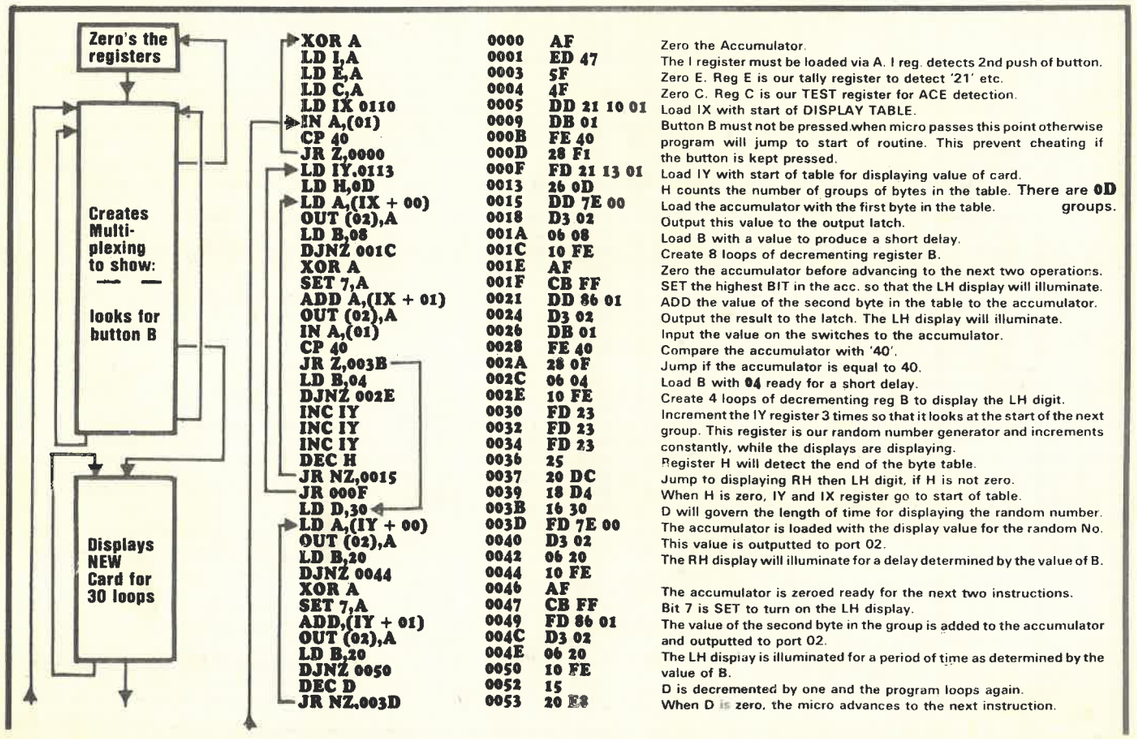
Here's the program: Type it on the TEC, hold it in the non-volatile RAM and play it on the Microcomp.

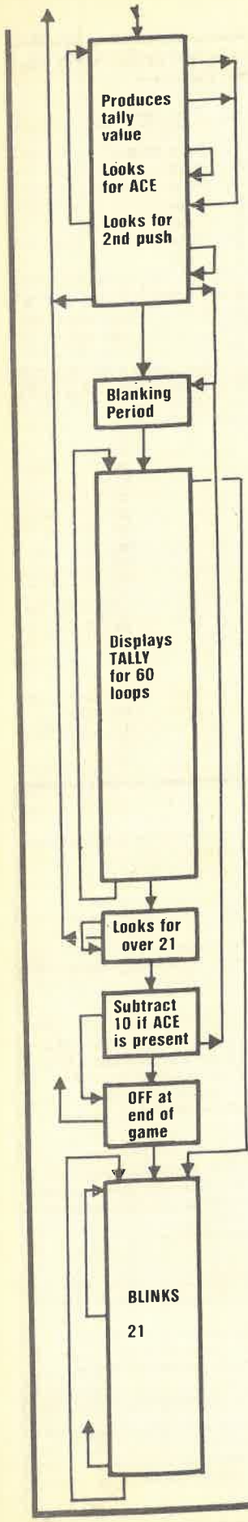
At 0100:
Each hex value produces a number from 0 to 9:

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	






```
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.

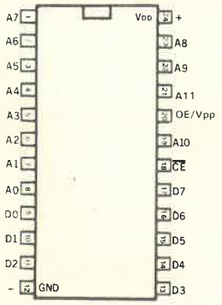
Burning a program is only done after you are thoroughly satisfied with its performance, as it is very difficult (if not impossible) to change the program , once it is burnt. For this reason it is best to get the program up and running via a medium which can be easily altered, as a program quite often 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 pro-gramming 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 pro-gramming. 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 back-wards through the program using the DISASSEMBLY codes on the back of issue 12 (and also in Notebook No 3) and make sure you get the same instructions as in the original production of the program.

Next check the JUMP and JUMP RELATIVE values to confirm that the microprocessor is actually landing on the address intended. Read the section on Jump Relative in issue 12 of TE, because these are the trickiest bytes to add to a program. Remember, they are the LAST bytes to be inserted as you need to count the number of bytes between the present address and the address to be jumped to.

Machine Code programming allows you to create your own system - with pen, paper and op-codes.

When creating a program, you will not know the value of a displacement byte initially and it is important to put a line in place of the byte thus: _____ so that it can be inserted later. This line lets you know that one byte must be counted when working out the displacement values.

If the display still fails to illuminate, you can create your own display value by loading the accumulator and outputting it to the display and then adding a HALT instruction. This is a last resort! and lets you know how far the program is progressing.

I hope you don't have troubles of this complexity but if so, this will get you out.

Start with simple programs and get your ideas flowing. It's not as difficult as you think to convert ideas into visual effects and its very rewarding to see them running.

When writing a program for the Microcomp, you start at address 0000. This is where the processor naturally starts when the reset button is pressed.

It can then be shifted to a higher location and a jump routine used to access it.

Creating a program which RUNS takes a certain amount of skill. By 'runs' we mean it completes one pass of the program and displays the appropriate information on the displays. After you get it to run you can concentrate on adjusting the values of timing to achieve the most pleasing effects.

But the main problem is getting the program to run and we have already mentioned how to get into the program and force it to display. There are a couple of other points which we forgot to mention and they involve the placing of tables.

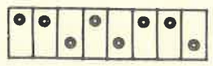
Tables should be placed well away from the program so that you don't run out of room. When everything works perfectly, they can be moved up and the pointers changed accordingly.

The idea is to get everything into a compact block and relative addressing uses less bytes than absolute addressing, so use it whenever possible. . Also remove any **NOPs** and any holes or spaces. Closing up a program and neatening it up takes time but it makes it much more presentable in the end.

We will now continue with the programs in the monitor, explaining each and every instruction and how the program is intended to work.

FROM INPUT PORT TO 8 LEDS

This routine is located at 0290 and is addressed by switching the switches ON thus:



This program is very handy for checking the operation of the computer in the early stages. This may be too late for some constructors, but for those with a problem in the displays, it will help locate the fault.

The program checks each line of the input port and outputs it to the displays.

Each time you turn an input switch ON, the corresponding LED, in the row of 8 LEDs, will be illuminated.

If this does not happen, you can trace through the particular line and locate the fault.

The program at **0290** contains 6 bytes. That's all, just 6 bytes! It inputs the data on the input port and loads it into the accumulator. It then outputs it to port 2 to turn on the appropriate LEDs and then jumps back to the start of the program.

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

The program can also be used to compare between the row of 8 LEDs, the 7-segment display(s) and the 4x4 matrix.

Experiment by inputting a hex value and see the effect you get on each of the displays.

In this way you can create any effect you want on the 4x4 (within limits).

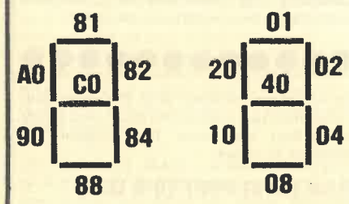
FROM INPUT PORT TO 8LEDS

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.

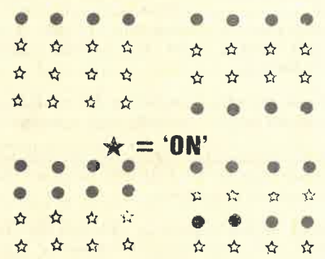
There are some limitations as to what combinations of LEDs can be turned on and this is something you must be aware of.

Experiments:

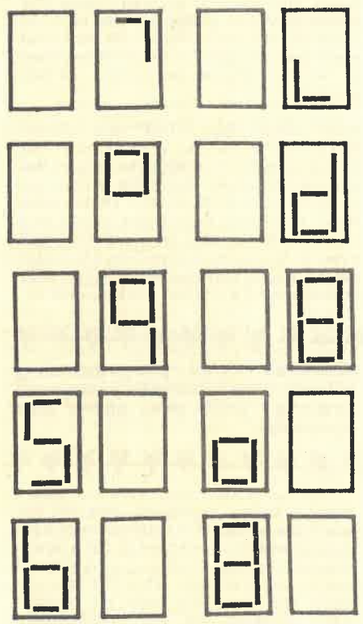
Create these effects by using the input switches:

(a) (b) (c) (d)

Create these effects on the 4x4 matrix:

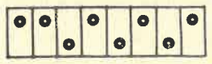


Create these on the 7-segments displays:



INCREMENT via BUTTON A

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



LD A,00	**02A0**	**3E 00**	Load the accumulator with zero.
LD C.A	**02A2**	**4F**	Load zero into C.
IN A,(01)	**02A3**	**DB 01**	Input the value on the switches to the accumulator.
BIT 7,A	**02A5**	**CB 7F**	Test BIT 7 of the accumulator to see if button A is pushed.
JR Z 02A3	**02A7**	**28 FA**	Jump to **2A3** if NOT pressed. Go to **2A9** when pressed
LD A,C	**02A9**	**79**	Load C into the accumulator.
INC A	**02AA**	**3C**	Increment the accumulator.
LD C,A	**02AB**	**4F**	Load the answer into the TALLY register 'C'.
OUT (02),A	**02AC**	**D3 02**	Output the accumulator to the displays.
IN A,(01)	**02AE**	**DB 01**	Input the switches to the accumulator.
BIT 7,A	**02B0**	**CB 7F**	Test BIT 7.
JR NZ 02AE	**02B2**	**20 FA**	Jump to **2AE** if A is pressed. Go to **2B4** when released.
JR 02A3	**02B4**	**18 ED**	Jump to **2A3.**

This will enable you to see the effects on the display without having to manually input values via the switches.

The accumulator is required for two functions. It outputs the value of the count and then looks to see if a switch is pressed. That's why we need another register to hold the value of the count, so that the accumulator can be loaded with other information. Thus the C register has been used for temporary storage.

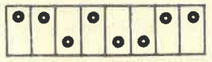
The program contains two small loops and the micro is constantly executing the top one when button A is not pressed and the lower one when the button is pressed. The micro jumps from one loop to the other during the time when the button is travelling from one state to the other.

```
3E 00
4F
DB 01 ←┐
CB 7F  │
28 FA ─┘
79
3C
4F
D3 02
DB 01 ←┐
CB 7F  │
20 FA  │
18 ED ─┘
```

This is a very simple way of creating a debounce condition and prevents more than one count being registered on each press of the button.

AUTO INCREMENT (fast)

This program is located at **02C0** and lets you sit back and watch the displays



increment automatically. You will be interested to know that the program takes 256 steps before it repeats!

Compare the effect on the row of 8 LEDs with the 4x4 and seven segment displays.

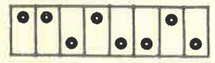
Notice that they produce entirely different effects due to the placement of the LEDs and this can be remembered when designing displays for advertising etc.

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

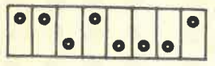
Load the value 01 and compare it with 00. 00 is a much longer delay and it appears to be as long as FF! In fact this is the case! The longest delay is produced when a register is loaded with 00 since the first operation to be performed on the register is to decrement it. The result is FF and that's why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

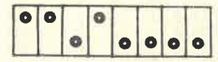
counter. The best effect of decrementing can be seen on the 8 LEDs. Adjust the speed control to view the effect in slow motion.

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.

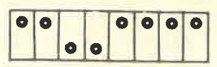


02F0

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

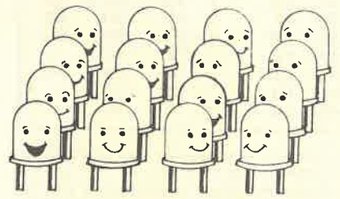
will produce almost no interpretable effects on either of the other displays.

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.

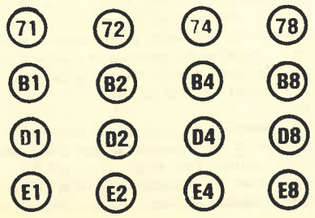
At 0338:		At 0350:		
01	CF	0F	B8	D4
02	3F	FF	D8	D2
04	CF	0F	E8	B2
08	3F	FF	E4	B4
EF	96	0F	E2	D4
DF	FF	FF	E1	D2
BF	96	0F	D1	B2
7F	FF	FF	B1	B4
03	33	71	71	
0C	CC	72	72	
03	C3	74	74	
0C	3C	78	B4	

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:



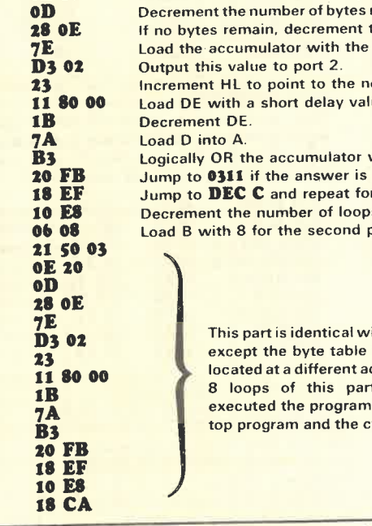
To access the LEDs we have separated the output latch into two halves with the 4 lower bits connected to the anodes and the 4 upper bits to the cathodes.

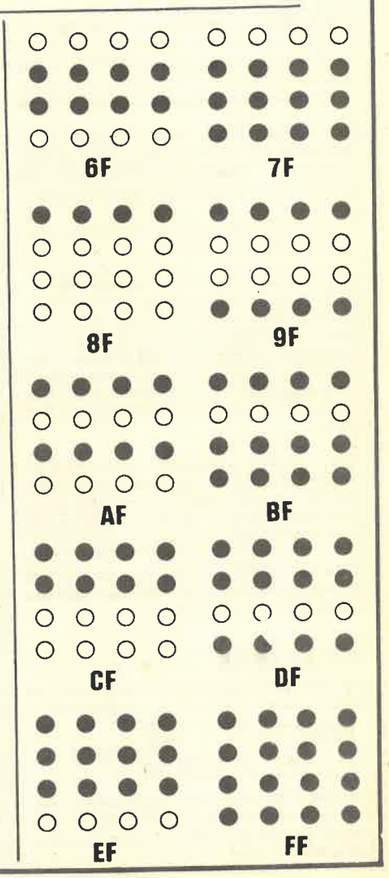
The following diagrams give you the values required to turn ON one or more LEDs:

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

LEGEND:

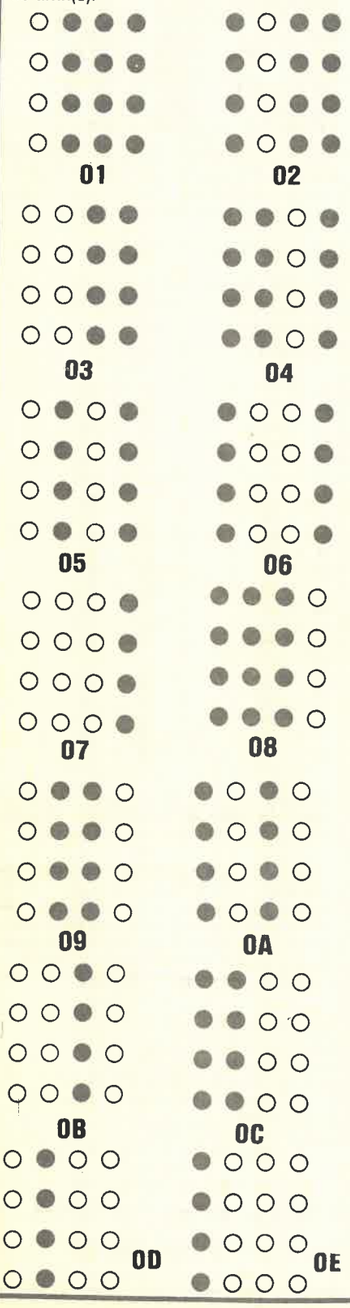
○ = ON.
● = OFF.

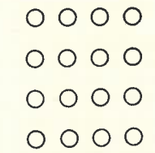




If you don't want all the LEDs in a row to be illuminated, refer to the diagrams on this page for the hex value needed to illuminate an individual column or column(s).

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).



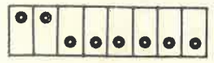


When the two diagrams are placed on top of each other, the LEDs that are common to both, will be illuminated. Due to the sinking and sourcing limitations of the output latch, all the LEDs in the 4x4 can not be illuminated at the same time.

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.**

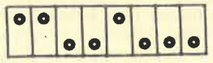
DJNZ's can be grouped thus:

```
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.

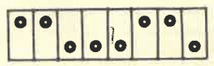
Because the program is running at high speed, we must include a DEBOUNCE feature to prevent more than ONE COUNT being registered when a button is pressed.

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.

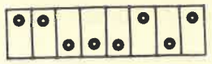


The main program for producing the letters on the display is located at **03A8** and the micro jumps to this address via the instruction **JR 03A8.** The main program is also used by the **A -Z, 0 - F** counter and shows how the same table and output program can be accessed by two different START-UP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

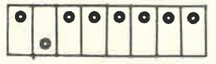
```
LD C,0A       0370   0E 0A    Register C is the counter for the BYTE TABLE. There are ten bytes.
LD DE,03DF    0372   11 DF 03 The DE register pair is loaded with the start-address of the byte table.
IN A,(01)     0375   DB 01    The input latch is looked at and the value it holds is placed into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

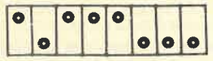
This is necessary to keep the displays illuminated while at the same time preventing the program from incrementing the displays if a button is kept pressed.

```
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 024A. If it pressed,increment to next instruction.
LD A,E        0424   7B       Load E into the accumulator, ready for an INCrement operation.
INC A         0425   3C       Increment the accumulator.
DAA           0426   27       Decimal adjust the accumulator. This means an A will be changed into 10.
LD E,A        0427   5F       Save the new count value by loading it into the E register.
JR 0432       0428   18 08    Jump to 0431.
BIT 6,A       042A   CB 77    From 0422, the program jumps to this address and tests for button B.
JR Z,0402     042C   28 D4    If not pressed, the program jumps to 0402. If pressed, the program increments.
LD A,E        042E   7B       Load the COUNT REGISTER into the accumulator.
DEC A         042F   3D       Decrement the accumulator.
DAA           0430   27       Decimal adjust the accumulator. This will change a zero into a 9.
LD E,A        0431   5F       Save the count value by loading it into the E register.
LD A,E        0432   7B
AND 0F        0433   E6 0F
LD HL,03E0    0435   21 E0 03
ADD A,L       0438   85
LD L,A        0439   6F
LD A,(HL)     043A   7E
OUT (02),A    043B   D3 02
LD A,E        043D   7B
RRA           043E   1F
RRA           043F   1F
RRA           0440   1F
RRA           0441   1F
AND 0F        0442   E6 0F
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.

```
          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 if A is pressed.
          JP 0470          051A  C3 70 04  Jump to start of DICE program.
```

At 04D3:

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

MICRO-COMP PART III

This is the third article on the Micro-Comp and covers the remaining set of programs in the lower half of the EPROM.

Most likely you will have addressed these programs by now and I'm sure you will like to see how they have been put together.

Remember, the programming techniques at this stage are very simple, to enable you to understand how a program is put together. As we advance to more complex programming, each instruction will require more thought and it may take you 5 or 10 minutes to see what the programmer has done. I don't think you're up to that yet but we will take it one stage further than the last article and explain the programs in a broad sense so you can see how they operate. It's important for you to work your way through each program, line-by-line (instruction by instruction) and be sure you

Sometimes you can remove bytes and other times you can see a better way of structuring the program. This is what we will be expecting soon. But for now let's go to it.

MODS

Before we do, there is a mod you can check on your board. The pull-down resistors on the input port should be 5k6 and not 10k to make sure the input port is zero when the slide switches are off. The cathode ends of the 8 input diodes are on the left side and the mini speaker is replaced with a piezo diaphragm and 10k in parallel with it.

The resistor is soldered under the board and the diaphragm attached to the top with a small piece of blue-tack.

Some output latches cannot drive all the displays to their full brightness, at the one time. To improve the brightness of the LEDs on the 4x4, connect a jumper lead between the base of the driver transistor of the first display and ground. This will turn off the first 7-segment display and increase the brightness on the 4x4.

We have started to fill the top half of the EPROM with some interesting programs and these will be available very soon, along with some of the add-ons.

It's demand from you that enables us to bring out more of the add-ons so please enquire to see if they have been released.

Now, to the programs:

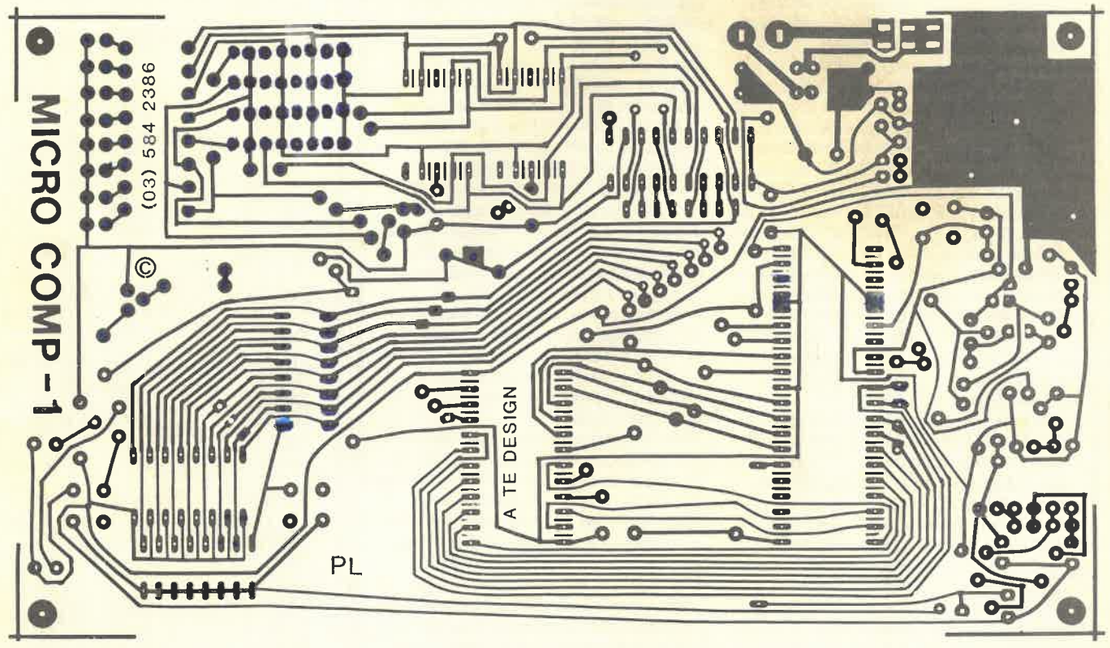
0520 EPROM IN BINARY

This is a program that displays the contents of the EPROM on the set of 8 LEDs.

It starts to look at address 0000 and outputs each value for a short time then increments to the next location.

The input switches are positioned thus:

Let the program run and you will see random numbers, figures and odd characters appear on the 7-segment displays. If you wait long enough, you will see names and other effects such as the 4x4 routines and end message appear. This program shows how the HL register pair points to an address and the value at that address is loaded into "A" and outputted to the display.



Artwork for Micro-Comp

You can see how fast 3xDJNZ's are executed by turning the speed control up and then turning it down. The program is very simple. Here it is:

EPROM DISPLAYED IN BINARY:

LD HL,0000	0520	21 00 00	The HL register pair is loaded with the first address (0000) which the program will view.
LD A,(HL)	0523	7E	The accumulator is loaded with the contents
INC HL	0524	23	of this address and outputted for a delay of 3
OUT (02),A	0525	D3 02	lots of 256 decrements of register B. The prog
DJNZ,0527	0527	10 FE	ram starts again, this time looking at the next
DJNZ,0529	0529	10 FE	location as register pair HL has been incre-
DJNZ,052B	052B	10 FE	mented. The program continues for ever.
JR 0523	052D	18 F4	

POKER PROGRAM:

LD DE,05CA	0530	11 CA 05	By now you will be able to work out what
LD HL,05DD	0533	21 DD 05	each instruction does by looking up the
LD IX,05F0	0536	DD 21 F0 05	machine-code instruction so we will go one
LD IY,060A	053A	FD 21 0A 06	stage further and explain the overall operation
LD C,20	053E	0E 20	of the program.
LD A,(DE)	0540	1A	Firstly refer to the diagram in the second part
OUT (02),A	0541	D3 02	of the article, under the heading 4x4 display,
LD A,(HL)	0543	7E	and see how you can select each of the 16
OUT (02),A	0544	D3 02	LEDs in the 4x4 by outputting a value such as
LD A,(IX+00)	0546	DD 7E 00	71, B2, D4, E8 etc.
OUT (02),A	0549	D3 02	At 0530, four register pairs are loaded with
LD A,(IY+00)	054B	FD 7E 00	the start address of 4 tables. DE will display
OUT (02),A	054E	D3 02	column one, HL will display column two, IX
IN A,(01)	0550	DB 01	will display column three and IY will display
BIT 7,A	0552	CB 7F	column four.
JR NZ,055B	0554	20 05	At 053E, the loop counter, register C is
DEC C	0556	0D	loaded with 20H (32) loops and one of the
JR NZ,0540	0557	20 E7	LEDs in each column is outputted in turn be-
JR 055E	0559	18 03	tween 0540 and 054E.
LD A,C	055B	79	The input "button A" is looked at (0550) to
LD I,A	055C	ED 47	see if it is pressed and if not, 20H loops are
LD A,I	055E	ED 57	displayed before the program jumps to the
RRA	0560	1F	next loop.
INC A	0561	3C	This loop creates a random number by look-
LD I,A	0562	ED 47	ing at the value of C and/or the value in the I
AND 07	0564	E6 07	register and juggles it around to get a value in
CP 05	0566	FE 05	A. This is ANDed with 07 to get a result that
JP Z,05CF	0568	CA CF 05	is 1, 2, 3, 4, 5, 6 or 7. If it is 5, 2, 3 or 4, one of
CP 02	056B	FE 02	the LEDs is incremented and the program
JP Z,05E2	056D	CA E2 05	goes to a longer loop (3x256 loops) at 0581.
CP 03	0570	FE 03	At 059A the program looks for a jackpot by
JP Z,05F5	0572	CA F5 05	checking to see if the second row (B1, B2, B4
CP 04	0575	FE 04	B8) is lit. If so, the screen flashes (at 05B6). If
JP Z,060F	0577	CA 0F 06	not, the Z-80 goes back to 053E and advan-
IN A,(01)	057A	DB 01	ces the LEDs down the 4x4.
BIT 7,A	057C	CB 7F	Another small part of the program is respon-
JP Z,053E	057E	CA 3E 05	sible for advancing the pointer register look-
LD C,03	0581	0E 03	ing for an FF. When this is detected it decre-
LD B,00	0583	06 00	ments the pointer to start at the beginning of
LD A,(DE)	0585	1A	the table. These 4 sub-routines are at 05CF,
OUT (02),A	0586	D3 02	05E2, 05F5 and 060F.
LD A,(HL)	0588	7E	That's all there is to it. Go through the pro-
OUT (02),A	0589	D3 02	gram a number of times and you will see how
LD A,(IX+00)	058B	DD 7E 00	it works. The jackpot flash at 05B6 is a closed
OUT (02),A	058E	D3 02	loop and is the end of the main program.
LD A,(IY+00)	0590	FD 7E 00	Push reset to get out of the loop. Your aim is
OUT (02),A	0593	D3 02	to completely re-design the program to take
DJNZ,0585	0595	10 EE	less memory. For instance, the two main
DEC C	0597	0D	loops can be made into one, with a loop
JR NZ,0585	0598	20 EB	within a loop and instead of decrementing the
LD A,(DE)	059A	1A	pointer registers, they can be re-loaded. In all
CP B1	059B	FE B1	you can take 40 -50 bytes out and still keep
JP NZ,053E	059D	C2 3E 05	the same features. Try it.

0530 POKER

Poker is played on the 4x4 matrix. It is intended to simulate the wheels of a poker machine revolving and stopping on a JACKPOT!

The input switches are positioned as follows:

0530

LDA,(HL)	05A0	7E
CP B2	05A1	FE B2
JP NZ,053E	05A3	C2 3E 05
LD A,(IX+00)	05A6	DD 7E 00
CP B4	05A9	FE B4
JP NZ,053E	05AB	C2 3E 05
LD A,(IY+00)	05AE	FD 7E 00
CP B8	05B1	FE B8
JP NZ,053E	05B3	C2 3E 05
LD A,0F	05B6	3E 0F
OUT (02),A	05B8	D3 02
DJNZ,05BA	05BA	10 FE
DJNZ,05BC	05BC	10 FE
LD A,FF	05BE	3E FF
OUT (02),A	05C0	D3 02
DJNZ,05C2	05C2	10 FE
DJNZ,05C4	05C4	10 FE
JR 05B6	05C6	18 EE

at 05CA:	at 05DD:	at 05F0:	at 060A:
B1	72	D4	E8
D1	B2	E4	78
E1	D2	74	B8
71	E2	B4	D8
FF	FF	FF	FF

INC DE	05CF	13
LD A,(DE)	05D0	1A
CP FF	05D1	FE FF
JP NZ,053E	05D3	C2 3E 05
DEC DE	05D6	1B
DEC DE	05D7	1B
DEC DE	05D8	1B
DEC DE	05D9	1B
JP 053E	05DA	C3 3E 05
INC HL	05E2	23
LDA,(HL)	05E3	7E
CP FF	05E4	FE FF
JP NZ,053E	05E6	C2 3E 05
DEC HL	05E9	2B
DEC HL	05EA	2B
DEC HL	05EB	2B
DEC HL	05EC	2B
JP 053E	05ED	C3 3E 05

INC IX	05F5	DD 23
LD A,(IX+00)	05F7	DD 7E 00
CP FF	05FA	FE FF
JP NZ,053E	05FC	C2 3E 05
DEC IX	05FF	DD 2B
DEC IX	0601	DD 2B
DEC IX	0603	DD 2B
DEC IX	0605	DD 2B
JP 053E	0607	C3 3E 05

INC IY	060F	FD 23
LD A,(IY+00)	0611	FD 7E 00
CP FF	0614	FE FF
JP NZ,053E	0616	C2 3E 05
DEC IY	0619	FD 2B
DEC IY	061B	FD 2B
DEC IY	061D	FD 2B
DEC IY	061F	FD 2B
JP 053E	0621	C3 3E 05

Jackpots are usually paid when 4 identical characters appear in a direct line across the display. This is the same in our version. When you STOP the four LEDs across the second row from the top, the program will signal a jackpot by flashing the screen.

Button "A" is pressed to freeze the LEDs and after a lot of careful button-pushing, you will notice that the button must be kept in play to prevent the LEDS slowing down to a crawl. If you fail to keep the pace up, one or two of the LEDs will stop.

It will take about 10 to 15 minutes to get a jackpot and it's best to keep playing until you win, so that you can see the screen flash.

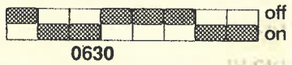
The aim of the program is to produce multiplexing on the 4x4 so that each column can be incremented separately. The program is long because we haven't taken any short cuts.

It uses 4 register pairs, one for each column, to point to an address for the value of the next LED. We cannot use INCrement or shift as each LED has a particular value. Read through the notes on the program to understand this.

Note: Some 74LS273 latch chips are not able to drive all the displays to full brightness, at the same time. To increase the brightness of the 4x4 display, short the base of both cathode driver transistors to emitter and this will divert more current to the 4x4 LEDs.

0630 BINARY CLOCK

Here's the switch positions:



0630

This is one for the electronics buff. It is a binary clock.

You use your skill at numbers to interpret a display and tell the time.

The readout appears on the 4x4 in binary form and once you know how to read it, you will never be late for an appointment.

Hopefully, to your great delight, your family will be unable to decipher it.

It may seem like a gimmick but a binary clock has been presented on a couple of occasions in electronics magazines and some schools have them in their electronics class.

A gimmick it may be but it highlights the capability of programming.

All you have to do is think of an idea and with a little bit of programming skill, you will be able to get it onto the displays.

The accuracy of the clock is controlled by the setting of the speed control. To set this accurately is almost impossible however there is a fine-tune adjustment via the input switches to get the time as accurate as possible.

See over for Binary Clock tables.

BINARY CLOCK PROGRAM

LD DE,0100	0630	11 00 01	D is the hours register, E is the minutes register.
LD SP,09D0	0633	31 D0 09	SP creates the 1 minute count.
LD HL,06A0	0636	21 A0 06	HL points to the tables for the LEDs on the 4x4.
LD A,E	0639	7B	Load the minute counter into A and
AND 0F	063A	E6 0F	AND with 0F to remove the high nibble.
ADD A,L	063C	85	Add the table address to the minutes value and
LD L,A	063D	6F	increment the pointer to look at the correspond-
LD A,(HL)	063E	7E	ing value in the table. Load that value into A and
OUT (02),A	063F	D3 02	output to port 2.
LD HL,06AA	0641	21 AA 06	Point to the table for the 10's minutes.
LD A,E	0644	7B	Load the minutes counter into A and rotate the
RRA	0645	1F	accumulator right four times to get the high nib-
RRA	0646	1F	ble into the low position (10's minutes) so that it
RRA	0647	1F	can be operated on.
RRA	0648	1F	
AND 0F	0649	E6 0F	AND the accumulator with 0F to remove the high
ADD A,L	064B	85	nibble and add the base of the table to the ac-
LD L,A	064C	6F	cumulator. Loading this value back into L in-
LD A,(HL)	064D	7E	creases the pointer register to look at the ap-
OUT (02),A	064E	D3 02	propriate byte and output to the display.
LD HL,06B0	0650	21 B0 06	Point to the hours table.
LD A,D	0653	7A	Load the hours register into the accumulator and
AND 0F	0654	E6 0F	mask off the high nibble.
ADD A,L	0656	85	ADD the table base to the accumulator and
LD L,A	0657	6F	increment the pointer to look at the correct byte.
LD A,(HL)	0658	7E	Load the value into the accumulator and output
OUT (02),A	0659	D3 02	to port 2.
LD HL,06BA	065B	21 BA 06	Load the pointer to look at the 10's hours table.
LD A,D	065E	7A	Load the hours register into the accumulator.
RRA	065F	1F	Shift the high nibble to the low position so that it
RRA	0660	1F	can be operated on.
RRA	0661	1F	
RRA	0662	1F	
AND 0F	0663	E6 0F	Mask off the high nibble.
ADD A,L	0665	85	Load the table base into the accumulator.
LD L,A	0666	6F	Increase the pointer to look at the appropriate
LD A,(HL)	0667	7E	byte and load the value into the accumulator.
OUT (02),A	0668	D3 02	Output to port 2
IN A,(01)	066A	DB 01	Look at the input port for button "A" pressed.
BIT 7,A	066C	CB 7F	Test bit 7 (button "A" is connected to bit 7.
JR Z,0674	066E	28 04	If pressed, execute a delay and jump to 0688 so
DJNZ,0670	0670	10 FE	that the minutes can be incremented rapidly.
JR 0688	0672	18 14	If not, go to 1 minute timing loop.
DEC SP	0674	3B	Start of 1 min timing loop.
LD HL,0000	0675	21 00 00	Zero the HL register pair and transfer the value
ADD HL,SP	0678	39	in the stack-pointer register to HL. The program
LD A,L	0679	7D	looks to see if this pair is zero and the program
OR H	067A	B4	loops again so that the LEDs on the 4x4 are kept
JP NZ,0636	067B	C2 36 06	illuminated.
IN A,(01)	067E	DB 01	This is the fine-tune loop for the 1 minute inter-
LD B,A	0680	47	val. It adds 1/2 second for each unit set on the
LD C,FF	0681	0E FF	input switches. "40" adds about 36 seconds and
DEC BC	0683	0B	"20" adds about 18 seconds etc. The displays
LD A,C	0684	79	turn off when this delay is executing. That's how
OR B	0685	B0	you can tell how long the delay is.
JR NZ,0683	0686	20 FB	
LD A,E	0688	7B	
INC A	0689	3C	The final part of the program decimal adjusts the
DAA	068A	27	D and E registers (via the accumulator) to get a
LD E,A	068B	5F	count-to-10. The program also detects "60" for
CP 60	068C	FE 60	the minutes and "12" for the hours as is neces-
JP NZ,0633	068E	C2 33 06	sary with a clock. The DAA (Decimal Adjust Ac-
LD E,00	0691	1E 00	cumulator) operation needs careful attention for
LD A,D	0693	7A	it to work correctly. WE have used INC A before
INC A	0694	3C	decimal adjusting it.
DAA	0695	27	
LD D,A	0696	57	
CP 13	0697	FE 13	Note: the register pairs at the beginning of the
JP NZ,0633	0699	C2 33 06	program must be in the right order so that the
JP 0630	069C	C3 30 06	jump commands here will execute correctly.

To see how the program works, click the input switches to 0630 and push reset. The starting time is 1 o'clock. Push button "A" quickly and you will see the minute LED come on. Push it again and 1:02 will be shown. Keep your finger on the button and you will see the time increment. The minute LEDs increment to 9 then the 10's LED comes on.

From this you should be able to see how the time is read.

ADJUSTING FOR 1 MINUTE

Switch all input switches to "off" and adjust the speed control until the display updates every 59 seconds. Now add 1/2 second by turning ON the lowest input switch. Keep adding to the delay until 60 seconds is reached.

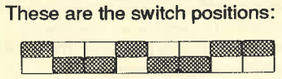
Make sure the brightness of the 4x4 is maximum by shorting between base and emitter of both cathode driver transistors.

at 06A0:	at 06AA:	at 06B0:
F8	F4	F2
E8	E4	E2
D8	D4	D2
C8	C4	C2
B8	B4	B2
A8	A4	A2
98		92
88		82
78		72
68		62

at 06BA:
F1
E1

06C0 ONE MINUTE TIMER

These are the switch positions:



06C0

The 1 minute timer program is a short program that uses a main program at 0740. It loads the accumulator with 1 and jumps to the delay program to execute 1 complete loop and then produces a tone. The time is adjusted by setting the speed control as close as you can to 1 minute and fine-tuning by adding very small increments via the input switches (1-40). Although it is not stop-watch accuracy, it's quite suitable for simple timing and you can get it very close to 1 minute.

1 MINUTE TIMER PROGRAM

IN A,(01)	06C0	DB 01
LD B,A	06C2	47
LD A,01	06C3	3E 01
LD IX,06C0	06C5	DD 21 C0 06
JP 0740	06C9	C3 40 07

The setting on switches 1-40 fine tunes the 1 minute time interval. This value is loaded into B for use by the main program. Load A with 1 for 1 minute in the delay program. Load IX with 06C0 to tell the main program to jump back to here after the tone. Jump to the main program.

06D0 3 MINUTE TIMER

The switch settings:

06D0

This program is identical to the one above except the accumulator is loaded with 3 for three loops of the main delay program.

3 MINUTE TIMER PROGRAM

IN A,(01)	06D0	DB 01
LD B,A	06D2	47
LD A,03	06D3	3E 03
LD IX,06D0	06D5	DD 21 D0 06
JP 0740	06D9	C3 40 07

06E0 1 HOUR TIMER

The switch settings:

06E0

This is the same as the above except the accumulator is loaded with 60 (3C) and the program jumps to the main delay routine.

1 HOUR TIMER PROGRAM

IN A,(01)	06E0	DB 01
LD B,A	06E2	47
LD A,3C	06E3	3E 3C
LD IX,06E0	06E5	DD 21 E0 06
JP 0740	06E9	C3 40 07

06F0 ADJUSTABLE TIMER

The switch settings:

06F0

ADJUSTABLE TIMER PROGRAM

LD IX,0765	06F0	DD 21 65 07
LD HL,06FA	06F4	21 FA 06
JP 00D0	06F7	C3 D0 00
IN A,(01)	06FA	DB 01
CP 00	06FC	FE 00
JR NZ,06F0	06FE	20 F0
LD IX,0778	0700	DD 21 78 07
LD HL,070A	0704	21 0A 07
JP 00D0	0707	C3 D0 00
IN A,(01)	070A	DB 01
BIT 6,A	070C	CB 77
JR Z,0700	070E	28 F0
LD IX,0782	0710	DD 21 82 07
LD HL,071A	0714	21 1A 07
JP 00D0	0717	C3 D0 00
IN A,(01)	071A	DB 01
BIT 7,A	071C	CB 7F
JR Z,0710	071E	28 F0
LD B,80	0720	06 80
LD A,81	0722	3E 81
OUT (02),A	0724	D3 02
LD A,00	0726	3E 00
OUT (02),A	0728	D3 02
DJNZ,0722	072A	10 F6
LD B,80	072C	06 80
IN A,(01)	072E	DB 01
AND 7F	0730	E6 7F
LD I,A	0732	ED 47
LD IX,072A	0734	DD 21 2A 07
JP 0740	0738	C3 40 07

MAIN DELAY PROGRAM:

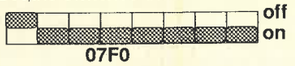
LD I,A	0740	ED 47
LD DE,8BFF	0742	11 FF 8B
DEC DE	0745	1B
LD A,E	0746	7B
OR D	0747	B2
JR NZ,0745	0748	20 FB
LD A,I	074A	ED 57
DEC A	074C	3D
JR NZ,0740	074D	20 F1
DJNZ,074F	074F	10 FE
LD A,81	0751	3E 81
OUT (02),A	0753	D3 02
LD A,00	0755	3E 00
OUT (02),A	0757	D3 02
IN A,(01)	0759	DB 01
BIT 7,A	075B	CB 7F
JR Z,0751	075D	28 F2
JP (IX)	075F	DD E9

The main delay program is basically a one-minute delay routine that is accessed by the 1 minute, 3 minute, 1 hour and adjustable timer routines.

The program for the adjustable timer contains its own set of instructions that appar on the screen. Flick the input switches to 06F0 and follow the instructions. You can set the input to any one-minute interval between 1 and 127 minutes. The instructions are: "SET ALL TO ZERO" "PUSH B" "SET DELAY VALUE" "PUSH A."

07F0 FINAL MESSAGE

Set the switches thus:



07F0

LD IX,07AA	07A0	DD 21 AA 07
LD HL,07A0	07A4	21 A0 07
JP 00D0	07A7	C3 D0 00

The final message reads: "ROM ENDS AT 07FF CHANGE LEAD FOR UPPER HALF AND USE FOR YOUR OWN IDEAS. CHEERS. COLIN."

This completes the Micro-comp lower half. The top half of the EPROM is being programmed at the moment and will contain some more complex programs. Keep a look-out for its release.

MICROCOMP TUTORIAL

Some more notes on the Microcomp programs

Both the Poker and Binary Clock programs use the 4x4 display as a read-out.

The 4x4 is wired in a rather unusual way, with 4 lines from the output latch driving the columns and 4 lines sinking the rows.

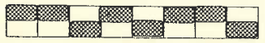
Although this arrangement may never be used in a full-size design, it shows how 8 lines can be arranged to drive a matrix of 16 LEDs and produce the beginnings of a large display.

Our job is to decode the value needed to access each LED individually.

To turn on any LED, one of the column lines must be high and one of the row lines must be low.

You can see this yourself on the Microcomp by accessing "From input to 8 LEDs" program at 0290 and work out the value required to turn on each LED.

Switch the input switches to 0290:



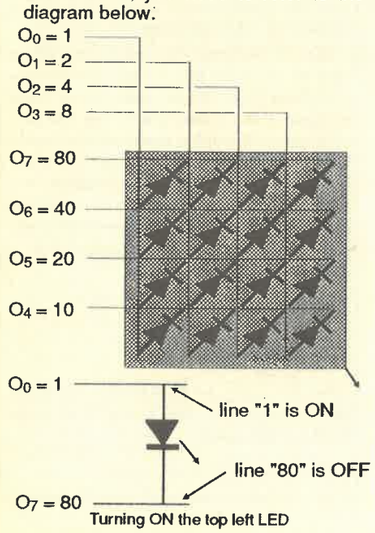
and push reset. Turn all the switches off and the LEDs on the 4x4 will go out.

Turn ON switches 1, 2, 4, 8 and the display will be fully lit. This means "0F" turns all LEDs ON.

Turn ON switches 10, 20, 40, 80 and one row at a time will go out. When all switches are ON, the display is OFF.

This is not getting us very far as we are accessing only rows and columns.

We need to concentrate on getting one LED at a time lit and to understand how this is done, you need to refer to the diagram below.



Turning ON the top left LED

When line "1" is high, and line "80" is low, the LED connected between the two lines is illuminated.

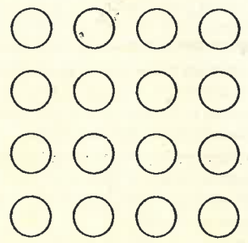
The simplified diagram shows this more clearly. If "80" is off, it means switches 40, 20, 10 are ON and thus the value needed to turn on the LED is 40 + 20 + 10 + 1 = 71.

Prove this by turning on the appropriate switches on the Micro-Comp and the top left-hand LED will be lit.

From the same diagram you can see the two lines required to turn on the top LED in the second row. Switch "2" must be high and 40, 20, 10 must be low.

Continue with each of the LEDs, getting them to turn on individually by working out the value from the circuit diagram and proving the result by using the switches.

Place your results here:



It's important to understand how to access each element of a display so that you can create your own characters to produce effects and animation.

Take the effect in the Poker program for example. It's similar to the revolving wheels of a poker machine and to be able to produce this effect you must know the value of each LED and access it in a particular order.

Next you need to work out what type of effect you want to create. It may be flashing, running, rotating or random illumination.

Once the program has been worked out, the timing for each part of the sequence must be determined.

This involves creating delay loops or "do nothing" loops so that the LEDs can be illuminated long enough for the effect to be observed.

If the delay is too short, the LEDs will flash too fast and the effect will be missed. If they change too slow the effect will drag.

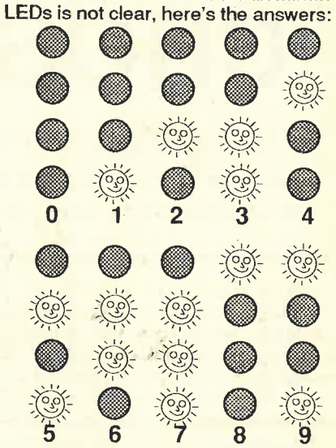
One of the last jobs in finalising the program is to adjust the delay lengths to get the speed right.

CLOCK PROGRAM

This program uses the display to produce a binary readout of the time.

As we have suggested in the notes on the previous pages, you will be able to work out the time by accessing the program and using the increment button to see what comes up on the display.

But in case the value of the illuminated LEDs is not clear, here's the answers:



CREATING MESSAGES

One of the advantages of a 7-segment display is its ability to reproduce almost all of the letters of the alphabet.

If a number of displays are connected together, it is possible to display words and sentences to give on-screen instructions etc.

All the necessary characters must be contained in tables at the end of the program and this usually makes the listing very long. One of the programs that provides on-screen information is the ADJUSTABLE TIMER. Address 06F0 and follow the instructions to see how easy they are to carry out.

Don't forget, we are using only 2 displays and it becomes much easier to read words when 4 or more displays are used.

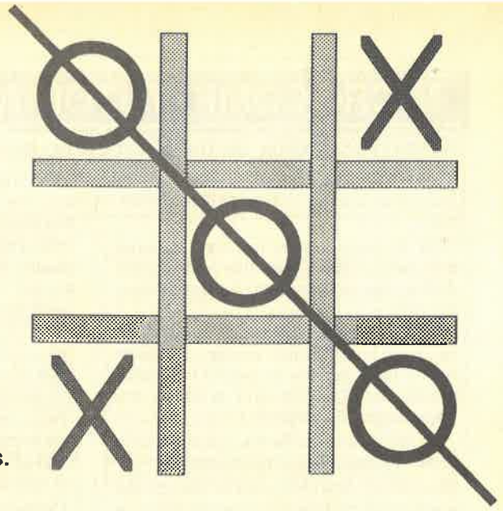
When more displays are used, the possibilities for displaying all kinds of effects becomes quite challenging. You can get running letters, flashing words and complete sentences appearing at any speed you wish.

The next stage in this project involves creating your own programs and burning them into the other half of the EPROM.

To do this you will need the TEC-1B computer, non-volatile RAM and eprom burner projects. See the current prices in the latest issue of TE and start experimenting.

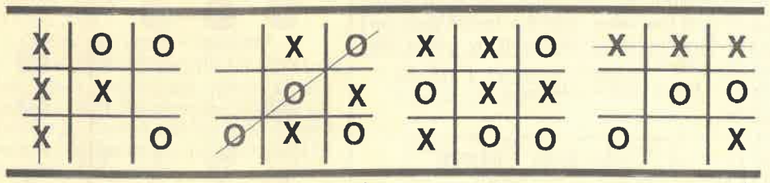
Next time we will have a great add-on for the Micro- Comp. Look out for it.

NOUGHTS & CROSSES



AN "ADD-ON" FOR THE MICROCOMP

Plays an intelligent game and keeps you on your toes. Maybe that's why they call it TIC TAC TOE!



Parts & PC: $29.40

PC board only: $4.80

Noughts and Crosses is one of the simplest and most challenging games to be invented. I don't know who invented it but I remember playing it at the back of a boring class and whenever I had an idle moment!

With just a choice of nine locations, two players can pit their wits and very quickly work out who is the superior player.

Even though it's very simple, it has an enormous fascination. From an early age I have had the desire to produce a machine capable of playing the game and hopefully winning!

Until now, this has eluded me. But with the introduction of the Micro-comp, it has become a reality. Using a Z-80 microprocessor and two other chips, a micro system has been created that offers all the capabilities of a real-life computer (in a manner of speaking).

Although our program does not use high-level strategy, it plays a sufficiently powerful game that wins, every time the computer goes first. When a player makes the first move, it replies with a random move so that the player will win only some of the time.

We could have tightened up the program to professional standards and the computer would never lose, but that would limit the number of ways of playing and spoil much of the challenge.

There are more than 5 ways of winning (with the various rotations) and it will take quite a lot of games to discover them.

This project is double-edged. Not only does it produce a good game, but it shows how to program in Z-80 machine code (assembly code) language.

The program has been kept as simple as possible to show how the micro advances through the program, making decisions as it goes.

There are a number of clever aspects to our design and these include a half byte memory (nibble memory) for storing the board values, a "jump to" sub-routine using one of the register pairs for the return address and an 8-line output latch to drive 11 LEDs.

The intention has been to produce a project and a program that can be understood by the beginner (near beginner) so that the possibility of modification and improvement can be considered.

Each instruction of the program is explained fully, together with what it does.

THE RULES

Everyone knows the rules of Noughts and Crosses so we won't need to describe them here. But there is one point we should explain. There are various levels of play to the game.

The simplest is to make a series of moves in the hope of achieving a win. The other is to "set up" the board to create a win in two directions. In this ploy there are two approaches. One is an obvious 'split' by taking opposite corners, the other is more sneaky. It involves placing pieces on either side of your opponent and thereby creating a non-threatening situation. The next move will place a key piece to link-up the two directions and the game is won.

ABOUT THE MICRO-COMP

Before starting this project, it is essential to realise the Noughts and Crosses project is an "add-on" for the 3-chip computer called the Microcomp.

The prices and data given in the article apply to the add-on section and that's why it may seem simple and low-priced.

The "running system' is the Microcomp and its construction and operation is covered in other articles.

The Microcomp kit comes complete with an EPROM that is filled with simple projects and ideas that show how programs are created and this will provide you with the background needed to understand the working of this program. When you build the Microcomp, the "add-ons" will be a low-priced extra and provide a lot of features for very little cost.

If you are starting from scratch, this is what you will need:

1 - Microcomp kit and PC board
1 - Noughts and Crosses kit and PC
1 - 9v 300mA plug pack
1 - case

Boards, parts and EPROMs are available separately, for those who have some of the parts.

This can be called 'levels of play' and the computer has been programmed to recognise the first two but not the third.

When you advance to the third, the challenge is to see how many variations can be created and how many ways you can win.

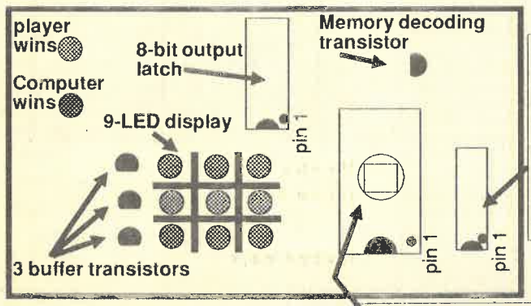
0's & X's IS AN 'ADD-ON'

Our Noughts and Crosses project is an 'add-on' for the Microcomp. The board contains programmed ROM, output latch, 'nibble' memory, 9-LED display, 3 driver transistors, memory decoder transistor and the WIN and LOSE LEDs.

The output latch drives the 3x3 matrix using 6 of its output lines with the remaining two powering the WIN and LOSE LEDs. When both these LEDs flash, the game is a 'stale mate'.

The program is the heart of the project and it is assumes the Microcomp has already been constructed and is working correctly.

When you buy the Microcomp kit, it comes with a programmed EPROM containing a number of simple programs covering input-output instructions, incrementing, displaying, counting and a few games. But not the Noughts and Crosses game. This is a separate chip.



The "O's & X's" PC board

EPROM containing the Noughts & Crosses program

After you study the simple programs, you should be confident enough to tackle this project. We have further 'add-ons' planned and they will all be using the Microcomp as the basic running system.

All of parts for the O's & X's are available in full kit-form or short-form. If you have most of the parts, the PC board and programmed EPROM is available. Otherwise the full kit is suggested.

On the other hand, if you want to burn the EPROM yourself, you can get the kit minus the EPROM, and by using the listing on the follow pages, you can type the program yourself.

The program can also be placed in a non-volatile RAM and run on the 'comp.' This is a most successful way of doing things and is the secret of how the program was developed in the first place.

THE PROGRAM

A description of the program will help you understand how the computer 'thinks' and how to improve its performance, if required.

The program uses 4-bit RAM memory to hold the data for the game. With 4 bits we can store values from 0-F and this is sufficient as we only need values from 0 - 4.

A replica of the board is created in memory during the start-up routine. Nine

Computer value: 04
Player value: 01
Cursor value: 02

memory cells. from 0801 to 0809 are filled with zero to represent a blank board.

The program then goes to a SCAN routine. This routine is separated into two parts. The first turns ON all the LEDs (which are in play) and the second part turns ON only the computer's LEDs.

The result is the computer's LEDs appear to be illuminated all the time while the player's pieces flash. The program detects the difference between the two by the value in memory.

Computer pieces are given the value 04, while the player's pieces are 01. Another feature that must also be detected is the piece currently being placed on the board by the player. It is identified by the value '02'.

The value "04" has been chosen for the computer so that it is possible to know the combination of the pieces in any row, column or diagonal.

Since a win for the player is represented by 03, the next value we can select is 04. The value for the cursor does not come into the count as it is turned into a "player piece" before row-tallying is commenced.

To place a piece on the board, button A is pressed. This introduces a flashing piece and the program then goes back to the SCAN routine.

On each subsequent press of button A, the piece advances across the board, jumping any of the pieces already in play. When it gets to the end, it re-appears at the start.

The computer must recognise this

0801	0802	0803
0804	0805	0806
0807	0808	0809

The positions on the board corresponding to the memory locations

piece and flash it at the same rate as a player's piece. But it must also know where it is located on the board, otherwise it won't be able to locate it.

As soon as you are satisfied with the location of the new piece, button B is pressed. This causes it to remain permanently in position.

A Noughts and Crosses program is a very simple requirement for a micro controller chip. It only requires a few pages of program and a small area to store the data corresponding to the moves. This data is temporally stored in an area called RAM (Random Access Memory).

Since we only need about 9 pieces of information, we can use the smallest memory chip available. That's a 2114 - even though they are no longer made.

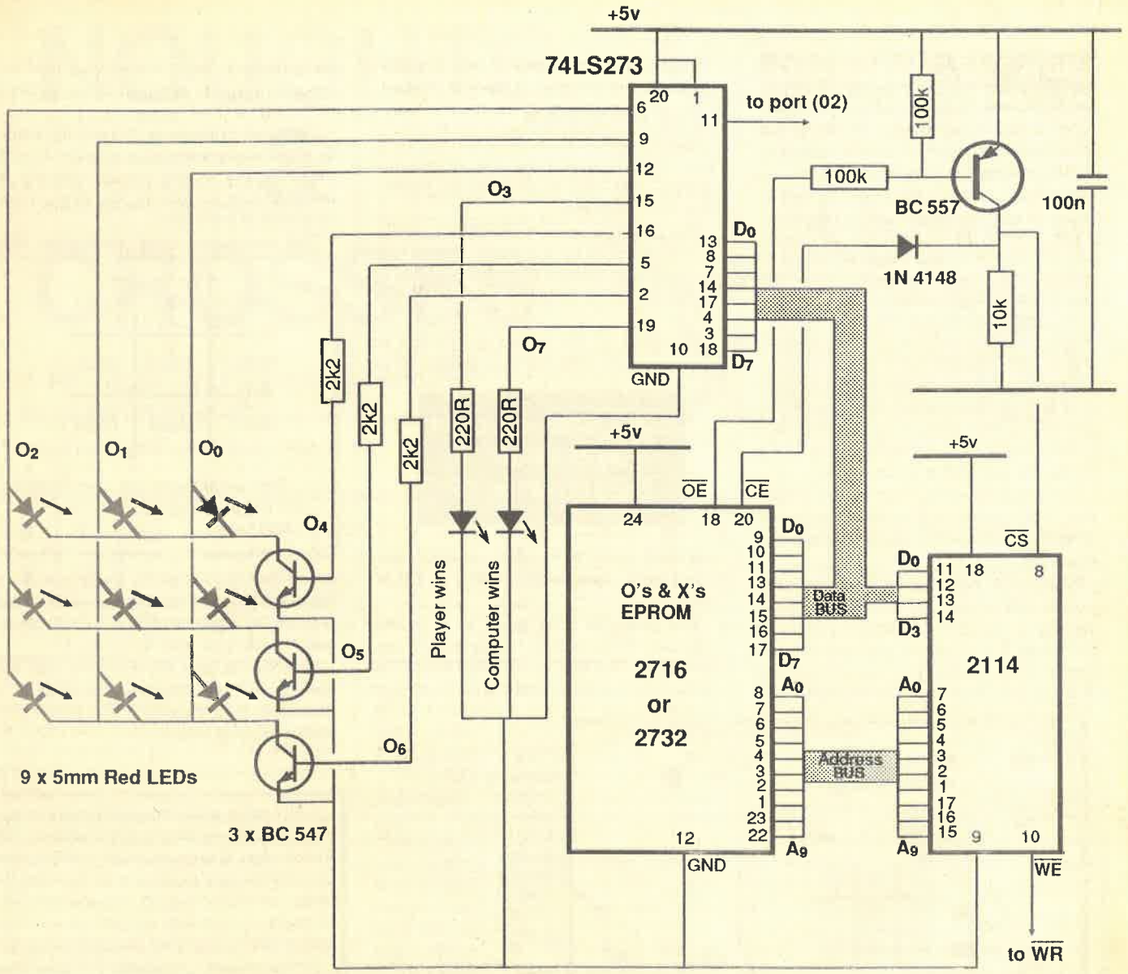
As you have seen, the basic Microcomp project runs without any RAM and if RAM is not available, the program cannot use commands such as PUSH and POP since a stack is not available.

For the program to have a stack, the Z-80 must have 8-bit memory. A 2114 provides only 4-bit memory and allows you to store values from 0 to 15. (0 - F).

Although the Noughts and Crosses program does not follow conventional programming rules, it shows how to get around the inconvenience of not having a stack.

In place of a stack, the program uses registers such as IX, IY and HL to store return-address values.

The main purpose of this project is to show how a program is put together and how each of the steps of a game is converted to a set of instructions, and how these instructions are executed.



NOUGHTS AND CROSSES CIRCUIT

The Noughts and Crosses project fits on the Microcomp board with the O's and X's EPROM in the 2716/2732 socket.

The program starts at page zero and accesses a 74LS273 output latch and a 2114 memory chip.

The latch drives 3 columns of LEDs and the bases of 3 sinking transistors to create a 3x3 LED display. A "PLAYER WIN" LED and "COMPUTER WIN" LED are also driven from this latch.

The latch is accessed at port (02).

The Memory chip has 4 pages of nibbles, starting at 0800 to 0BFF and is used to store the current values on the Noughts and Crosses board.

The locations used in the memory chip are shown on the following page. Only the first few nibbles are used as we requires nine nibbles for the board and two locations to hold data for the scanning of the display.

The memory chip is accessed at 0800 and at this address A11 goes HIGH.

For instance, for an address of 0801, lines A0 and A11 will be HIGH. This turns on the chip via A11 as described below and accesses memory location 01 at the same time.

When A_{11} turns ON, the base of the BC 557 goes high and this turns the transistor off. When memory request MREQ (from the Z80) goes LOW, pin 20 of the EPROM goes low and thus the diode on this line does not supply a voltage to the memory chip pin 8. This allows the CS pin of the 2114 (pin 8) to be pulled low via the 10k resistor and the 2114 is selected.

The WE pin (10) is selected at about the same instant and when it is at a logic 0 level, the in/out lines act as inputs. This enables data to be written into the RAM.

To read data from the 2114, the address lines (above 0800) are activated and MREQ goes low. This causes the chip to be selected as before but this time the WE input is held at a logic 1 level.

The in/out lines then output the data stored at the selected address.

Only a half-byte of data is provided by the RAM with the high nibble coming through the bus as zero due to the pull-down resistors on the Microcomp board.

The 3x3 display is multiplexed in the program to provide any combination of LEDs to be displayed.

Full details of the operation of the program are given in this article including notes to help you produce your own program to run on the same display.

HOW THE CURSOR LED WORKS

The cursor LED starts at 0800 but since this location is not picked up by the SCAN routine, the cursor is not seen.

At 00F3, memory location is looked at by the IX register and is zero at the start of play. When this location is compared with 02, at 00F6, the result is not zero and so the program jumps to 010E where it is compared to 00 and since it is zero, 0800 is loaded with 02.

On the next push of the button, 0800 contains 02 and the program compares it with 02 at 00F6 and since the result is zero, it loads A with 00 and places 00 in memory location 0800, increments IX to 0801, checks to see that the cursor counter is not zero (at 0101, 0105 and 0108) then loads the value at the address looked at by the IX register into the accumulator.

If the square is empty (at 010E) it loads the accumulator with 02 and fills the square (at 0114).

The IX register keeps track of the cursor and by zeroing the location looked at by the IX register, incrementing IX and loading the new location with 02, the cursor moves across the display.

THE ALGORITHM

The program goes through a number of short routines that look for a particular condition and when it is found, the program stores the result and returns to SCAN.

PARTS LIST

2 - 220R 1/4 watt resistors
3 - 2k2
1 - 10k
2 - 100k
1 - 100n monoblock capacitor
3 - BC 547 transistors
1 - BC 557 transistor
1 - 1N 4148 diode

1 - 18 pin IC socket
1 - 20 pin IC socket
1 - 24 pins to produce wire-wrap "stand-off" socket
2 - 24 pin IC sockets

9 - 5mm red LEDs
1 - 3mm red LED
1 - 3mm green LED

1 - Programmed ROM (0's & X's)
1 - 2114 RAM
1 - 74LS273 Latch chip (from Micro-Comp)
2 - lengths hook-up flex
2 - female matrix sockets
10cm tinned copper wire
4cm heatshrink tubing 2.6mm dia

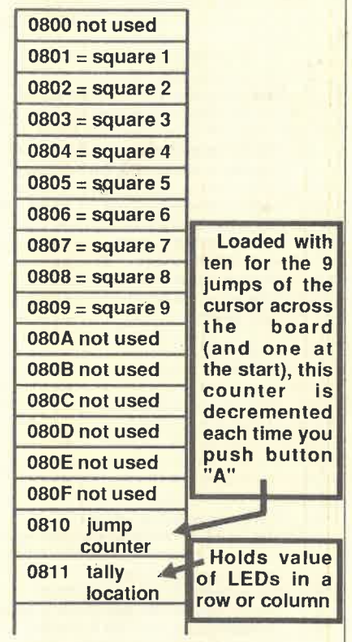
1 - Noughts & Crosses PC board

The order in which the 'conditions' are placed in the program is very important and when these are dealt with very quickly, the computer appears to have intelligence.

These routines are called ALGORITHMS. An algorithm is defined as a process that solves a problem in a finite number of steps.

You will notice that some of the algorithms in the program solve a problem (such as forcing a stalemate) even though there may be more than 30 different combinations of data. An algorithm is simply presented with data and comes up with an answer.

The first algorithm in the program appears at 011F. It looks for a player WIN.



Inside the 2114

This is detected by any row, column or diagonal containing the value 03.

If so, a bit is set in the 'C' register which prevents the switches being accessed and at the same time allows a value to be added to the output latch to turn on the WIN LED.

Next the program looks to see if it can win. Again, another algorithm. It does this by loading a sub-routine with 08 and looks to see if any column, row or diagonal contains 08. The flashing 02 piece will be converted to 01 by this time. The value 08 will represent 2 computer pieces in a row. When a particular row or column is found, the DE register pair points to a table containing the value of the corresponding 3 memory locations.

The program looks at each location in turn and if it is blank, puts 04 into memory and returns to SCAN. Next the program looks for a stale mate.

This is a sub-routine located at the end of the main program. It looks to see if all

1	2	3
4	5	6
7	8	9

The numbering of the squares on the board

locations are filled. If they are, the result must be a stale-mate as a "player win" or "computer win" will have already been passed in the program.

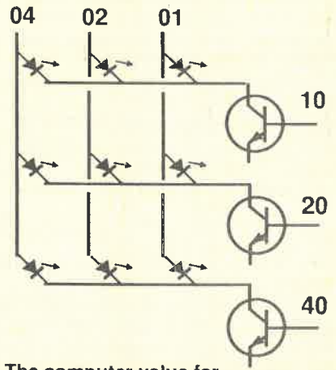
Next the program looks for a 'stopper' This is detected by looking for 02 in a row, column or diagonal and represents 2 player pieces. It responds by inserting a computer move into the empty location and returning to SCAN.

Next the program looks to see if the board is empty. This will mean the computer is making the first move and it will respond by placing a value in one of the four corners.

If any of the squares is filled, it will mean the game has already started and the next condition is to look for an empty centre square. It does this by firstly loading the centre square memory location into the accumulator and comparing it with zero. If it is empty, it loads 04 into the location and returns to SCAN.

The first piece of strategy (another word for algorithm) to be introduced into the program is to look for 06 on the board. This is a 'trap' situation where the player is not exerting any pressure at the moment but producing a 'fork' for a two-pronged attack in the next move.

The program must detect if the player



The computer value for each column and row.

NOUGHTS AND CROSSES PROGRAM

```
LD A,00     0000  3E 00
LD D,0A     0002  16 0A
LD HL,0800  0004  21 00 08
LD (HL),A   0007  77
INC HL      0008  23
DEC D       0009  15
JR NZ,0007  000A  20 FB
LD C,00     000C  0E 00
LD A,0A     000E  3E 0A
LD (0810),A 0010  32 10 08
LD IX,0800  0013  DD 21 00 08
LD E,08     0017  1E 08
LD IY, 0801 0019  FD 21 01 08
LD HL,0023  001D  21 23 00
JP 0077     0020  C3 77 00
ADD A,10    0023  C6 10
OUT (02),A  0025  D3 02
LD IY, 0804 0027  FD 21 04 08
LD HL,0031  002B  21 31 00
JP 0077     002E  C3 77 00
ADD A,20    0031  C6 20
OUT (02),A  0033  D3 02
LD IY, 0807 0035  FD 21 07 08
LD HL,003F  0039  21 3F 00
JP 0077     003C  C3 77 00
ADD A,40    003F  C6 40
OUT (02),A  0041  D3 02
DEC E       0043  1D
JR NZ,0019  0044  20 D3
LD E,08     0046  1E 08
LD IY,0801  0048  FD 21 01 08
LD HL,0052  004C  21 52 00
JP 00B3     004F  C3 B3 00
ADD A,10    0052  C6 10
OUT (02),A  0054  D3 02
LD IY,0804  0056  FD 21 04 08
LD HL,0060  005A  21 60 00
JP 00B3     005D  C3 B3 00
ADD A,20    0060  C6 20
OUT (02),A  0062  D3 02
LD IY,0807  0064  FD 21 07 08
LD HL,006E  0068  21 6E 00
JP 00B3     006B  C3 B3 00
ADD A,40    006E  C6 40
OUT (02),A  0070  D3 02
DEC E       0072  1D
JR NZ,0048  0073  20 D3
JR 0017     0075  18 A0

LD D,00     0077  16 00
LD A,(IY+00) 0079 FD 7E 00
CP 00       007C  FE 00
JR Z,0082   007E  28 02
LD D,04     0080  16 04
LD A,(IY+01) 0082 FD 7E 01
CP 00       0085  FE 00
JR Z,008B   0087  28 02
INC D       0089  14
INC D       008A  14
LD A,(IY+02) 008B FD 7E 02
CP 00       008E  FE 00
JR Z,0093   0090  28 01
INC D       0092  14
IN A,(01)   0093  DB 01
BIT 0,C     0095  CB 41
JR NZ,00AB  0097  20 12
BIT 1,C     0099  CB 49
JR NZ,00AB  009B  20 0E
BIT 2,C     009D  CB 51
JR NZ,00AB  009F  20 0A
BIT 6,A     00A1  CB 77
JR NZ,011A  00A3  20 75
BIT 7,A     00A5  CB 7F
JR NZ,00F1  00A7  20 48
LD A,D      00A9  7A
JP (HL)     00AA  E9
BIT 7,A     00AB  CB 7F
JR NZ,00A9  00AD  20 FA
RES 0,C     00AF  CB 81
JR 00A9     00B1  18 F6
```

C

A B X

The accumulator is zeroed.
Register D is loaded with 0A for to make 11 loops of the routine. HL is loaded with the starting address of memory and zero is loaded into each location. D is DECremented and if the result is not zero, the program loops.
Register C is our TEST REGISTER.
Location 0810 keeps track of the flashing cursor.
IX will look at memory, starting at 0800
E is loaded for 8 loops of the SCAN program. IY is loaded with the first memory location and HL is the return address after jumping to a sub-routine at 0077. The value for turning ON the LEDs in a row is generated in D and loaded into 'A.' The value 10 is added to A to turn ON the appropriate row.
'20' is added to A to turn on the second row.
The third set of bytes is looked at by IY and HL is loaded with the return address.
'40' is added to A to turn on the bottom row of LEDs.
The loop counter E is decremented and if it is not zero, the program loops.

0046 is the second half of the scan program in which only the computer pieces (04) are recognised. It is the same as above with the IY register sitting at the top of three bytes in the table. Register E is the loop register for the 8 sets of bytes. The program jumps to the sub-routine at 00B3 for the row values containing the computer pieces.

D is going to tell us which LEDs to turn on. IY sits the top of 3 memory locations. If the particular location is zero, the program jumps.
It is not zero, D is loaded with 04.
The IY register reaches to the second memory location to see if it is zero. If not, the D register is incremented twice.

IY reaches to the third memory location. If it is not zero, D is incremented

The input port is read.
Bit 0, 1 and 2 of the C register is tested to see if the input port is to be jumped over.

Button B is read

Button A is read

The tally register D is loaded into 'A' and the sub-routine returns.

Bit 0,C prevents switch bounce (see text)

This routine puts "00' into ten memory locations, starting at 0800

This routine provides a memory location for the flashing cursor, so we can keep track of it.

SCAN ROUTINE FOR COMPUTER AND PLAYER PIECES at 0017

The board is scanned 8 times in the routine from 0017 to 0045. It turns on all the LEDs that are in play. The routine then goes to a sub-routine at 0077 that produces the output value for each row. To this value is added 10, 20 or 40 to turn on either the first, second or bottom row as shown in the diagram at the end of the program.

SCAN ROUTINE FOR ONLY COMPUTER PIECES at 0146

Computer pieces are recognised by the value 04 and only the computer pieces are turned on in this routine. The sub-routine for detecting the computer pieces is at 00B3. Player pieces are not turned on during this part of the program and thus produce a flashing effect. This routine is looped 8 times before jumping to 0017.

TURN ON ALL THE LEDS IN A ROW IN PLAY at 0077.

Register D is the tally register and it creates a value from 00 to 07.
It starts with zero and the contents of the first location in memory is loaded into the accumulator and compared with zero. If it is zero, the zero flag is set and thus a jump to 0082 will be performed.
If the contents are 01, 02 or 04, the tally register will be loaded with 04 (to set-up the left row of the display) The next location, if filled, will increment D twice to hold 06, If the final location is filled the tally register will be incremented and will hold 07.
Bit 1,C is checked for switch debounce, bit 1,C for Player win and bit 2,C for Computer win, to see if the input port is to be jumped over.

Buttons B and A are detected in this routine. This routine is frequency scanned and therefore the buttons will be detected as soon as they are pressed.

A B X

```
LD D,00     00B3  16 00
LD A,(IY+00) 00B5 FD 7E 00
CP 04       00B8  FE 04
JR NZ,00BE  00BA  20 02
LD D,04     00BC  16 04
LD A,(IY+01) 00BE FD 7E 01
CP 04       00C1  FE 04
JR NZ,00C7  00C3  20 02
INC D       00C5  14
INC D       00C6  14
LD A,(IY+02) 00C7 FD 7E 02
CP 04       00CA  FE 04
JR NZ,00CF  00CC  20 01
INC D       00CE  14
XOR A       00CF  AF
BIT 0,C     00D0  CB 41
JR NZ,00EB  00D2  20 17
BIT 1,C     00D4  CB 49
JR NZ,00E9  00D6  20 11
BIT 2,C     00D8  CB 51
JR NZ,00ED  00DA  20 11
IN A,(01)   00DC  DB 01
BIT 6,A     00DE  CB 77
JR NZ,011A  00E0  20 38
BIT 7,A     00E2  CB 7F
JR NZ,00F1  00E4  20 0B
XOR A       00E6  AF
JR 00EB     00E7  18 02
ADD A,08    00E9  C6 08
ADD A,D     00EB  82
JP (HL)     00EC  E9
LD A,80     00ED  3E 80
JR 00EB     00EF  18 FA
SET 0,C     00F1  CB C1
LD A,(IX+00) 00F3 DD 7E 00
CP 02       00F6  FE 02
JR NZ,010E  00F8  20 14
LD A,00     00FA  3E 00
LD (IX+00),A 00FC DD 77 00
INC IX      00FF  DD 23
LD A,(0810) 0101  3A 10 08
DEC A       0104  3D
JP Z,000E   0105  CA 0E 00
LD (0810),A 0108  32 10 08
LD A,(IX+00) 010B DD 7E 00
CP 00       010E  FE 00
JR NZ,00FF  0110  20 ED
LD A,02     0112  3E 02
LD (IX+00),A 0114 DD 77 00
JP 0017     0117  C3 17 00

LD A,01     011A  3E 01
LD (IX+00),A 011C DD 77 00

LD B,03     011F  06 03
LD IX,012C  0121  DD 21 2C 01
LD IY,0137  0125  FD 21 37 01
JP 01EC     0129  C3 EC 01
SET 1,C     012C  CB C9
IN A,(01)   012E  DB 01
BIT 6,A     0130  CB 77
JR NZ,012E  0132  20 FA
JP 000E     0134  C3 0E 00

LD B,08     0137  0608
LD IX,0144  0139  DD 21 44 01
LD IY,0248  013D  FD 21 48 02
JP 01EC     0141  C3 EC 01
SET 2,C     0144  CB D1
JR 0155     0146  18 0D

LD B,02     0148  06 02
LD IX,0155  014A  DD 21 55 01
LD IY,0168  014E  FD 21 68 01
JP 01EC     0152  C3 EC 01
LD HL,0801  0155  21 01 08
LD A,(DE)   0158  1A
LD L,A      0159  6F
LD A,(HL)   015A  7E
CP 00       015B  FE 00
```

C

C

Y Z

The TALLY register D is zeroed.
A is loaded from the IY register and compared with 04 to see if it a computer piece.
If it is, the result is zero as a compare operation is similar to a subtraction (without altering the contents of A) and the program does not jump but loads D with 4 and looks at the next location (IY + 01).
Again the accumulator is compared with 4 and if a computer piece is found, the D register is incremented twice (at 00C5 and 00C6). The third location is (IY + 02) and it is compared with 04 and if a computer piece is present the D register is incremented (at 00CE).
A is then zeroed (at 00CF) and bit 0 of register C is tested to see if the input port is to be jumped over.
Bit testing extends from 00D4 to 00E4.

If bit 1,C is SET, the program turns ON "Player Win" LED by adding 08 to the accumulator (at 00E9).

If bit 2,C is SET, the program turns on "Computer Win" LED (at 00ED) by loading the accumulator with 80 and jumping to 00EB where it adds the other computer pieces then jumps back to the SCAN routine.

At 00F3 the program looks at the location of the cursor and if present, it will contain 02. This causes the program to increment to 00FA where A is loaded with zero and this value is loaded into the location looked at by the IX register to turn off the flashing cursor. The Cursor Counter register 0810 is looked at by A to see if it is zero (whereby the cursor will jump back to the start of the display). A is decremented and loaded back into the counter location (at 0108). At 010B the next location of the cursor is looked at by the IX register and its value loaded into the accumulator. It is compared with zero and if it is not empty, it is loaded with 02 to create the new cursor location. The program then jumps to 0017.

The flashing cursor is replaced with a player piece.

The program is looking for a player WIN.
IX is the "Found" register
IY is the "NOT found" register
Jump to the sub-routine
"Player WIN' has been found - SET bit 1,C
Look at the input port
Loop here if button B is being kept pushed

Return to SCAN

Look for a Computer WIN.
IX contain the "Found" address for the jump
IY holds "NOT Found" address for the jump
Jump to the sub-routine
"Computer WIN" found! SET Bit 2,C

Look for a "Stopper."
Stopper Found. Jump to 0155
Stopper NOT found. Jump to 0168
Jump to sub-routine.
This routine places the third computer piece in the appropriate row or column to create a WIN. DE holds the address of the first byte in the row, column or diagonal. This is then converted to a memory location

Sub-routine for row values containing computer pieces.

Bit 0,C is switch bounce
Bit 1,C is player WIN
Bit 2,C is computer WIN

Bit 6,A detects button 'B' and bit 7,A detects 'A.'
Add A,08 turns on player win LED and Add A,80 turns on Computer win LED.
JP(HL) is equivalent to a return statement. HL has been pre-loaded with an address in readiness for the completion of the sub-routine.

BUTTON 'A' ROUTINE at 00F1

When button A is pressed, the cursor LED address is contained in register IX

The value of this address is compared with 02, confirming it presence. The location is then zeroed and IX is incremented. The jump counter (address 0810) is decremented and if it is not zero, the next board location is compared with zero to see if it is empty. The location is then filled with 02 and the micro jumps to 0017.

BUTTON 'B' DETECTED at 011A

The cursor LED is replaced with a player piece.

LOOKS for a PLAYER WIN at 011F

It is detected by any row, column or diagonal containing the value 3. The sub-routine at 01EC performs the task and comes back with a 'found' value in IX or a 'not found' value in IY. Register pair DE will point to the first byte in a table

LOOKS for a COMPUTER WIN at 0137

This is detected by a row, column or diagonal containing the value 08.

Looks for a STOPPER at 0148

Any row, column or diagonal containing a value of 2 must be stopped.
If 02 is found, the micro jumps to 0155, if not it goes to 0168.
The sub-routine at 01EC does all the detection. If found, DE contains the first of the 3 bytes. If it is 04 we know the fourth square is involved.

```
JR Z,0162   015D  28 03
INC DE      015F  13
JR 0155     0160  18 F3
LD A,04     0162  3E 04
LD (HL),A   0164  77
JP 012E     0165  C3 2E 01

LD D,09     0168  16 09
LD HL,0801  016A  21 01 08
LD A,(HL)   016D  7E
CP 00       016E  FE 00
JR Z,0174   0170  28 02
JR 0196     0172  18 22
INC HL      0174  23
DEC D       0175  15
JR NZ,016D  0176  20 F5
LD A,R      0178  ED 5F
AND 07      017A  E6 07
LD HL,0238  017C  21 38 02
ADD A,L     017F  85
LD L,A      0180  6F
LD A,(HL)   0181  7E
LD H,08     0182  26 08
LD L,A      0184  6F
LD A,(HL)   0185  7E
CP 00       0186  FE 00
JR Z,0190   0188  28 06
LD A,R      018A  ED 5F
AND 0F      018C  E6 0F
JR 017C     018E  18 EC

LD A,04     0190  3E 04
LD (HL),A   0192  77
JP 012E     0193  C3 2E 01
LD A,(0805) 0196  3A 05 08
CP 00       0199  FE 00
JR NZ,01A4  019B  20 07
LD A,04     019D  3E 04
LD (0805),A 019F  32 05 08
JR 012E     01A2  18 8A

LD B,06     01A4  06 06
LD IX,01B1  01A6  DD 21 B1 01
LD IY,01E2  01AA  FD 21 E2 01
JP 01EC     01AE  C3 EC 01
LD A,(0805) 01B1  3A 05 08
CP 04       01B4  FE 04
JR NZ,01C1  01B6  20 09
LD A,R      01B8  ED 5F
AND 07      01BA  E6 07
LD HL,0240  01BC  21 40 02
JR 017F     01BF  18 BE
LD B,04     01C1  06 04
LD IX,01CE  01C3  DD 21 CE 01
LD IY,01E2  01C7  FD 21 E2 01
JP 01EC     01CB  C3 EC 01
LD HL,0801  01CE  21 01 08
LD A,(DE)   01D1  1A
LD L,A      01D2  6F
LD A,(HL)   01D3  7E
CP 00       01D4  FE 00
JR Z,01DC   01D6  28 04
INC DE      01D8  13
INC DE      01D9  13
JR 01D1     01DA  18 F5
LD A,04     01DC  3E 04
LD (HL),A   01DE  77
JP 012E     01DF  C3 2E 01

LD B,04     01 E2  06 04
LD IX,0155  01 E4  DD 21 55 01
LD IY,018A  01E8   FD 21 8A 01
```

Y Z not first move empty W

and compared with 00 (at 015B0). If it is zero, the program jumps to 0162 where the accumulator is loaded with 04 and this value is placed into the memory location. If not zero, the program looks at the other two squares.

D is the "Count register."
HL is loaded with the start of the board
The contents of the location is loaded into A and compared with zero to see if it is empty.
If it is empty, the program jumps to 0174
The location is NOT empty
The pointer register is INCremented.
The count register is DECremented
Jump back to the start of the routine of not the end of the board. Load A with a random number from the refresh register. Use only the 3 lowest bits. Load HL with "Corner table."
ADD 38 to the accumulator and place the result back in L to look down the table for a random number.
Load H with 08 for start of memory.
Load the random number into L
Load the value looked at by HL, into A.
See if the square is empty.
If it is, jump to 0190.
If not, get another random number

Jump to 017C and try again.

When an empty square is found, load A with 04 and place it in the location looked at by HL. Task is complete, jump to 012E.
Load A with the value in the centre square.
Compare it with 00.
If it is not empty, jump to 01A4.
If it is empty, load A with 04
Load 04 into the centre square.
Task complete. Jump to 012E

Routine will look for '6.'
IX contain the "FOUND' return address.
IY contains the NOT FOUND return address
Jump to the sub-routine
Look to see if the computer is in the centre square by comparing the contents of 0805 with 04. If it is not 04, jump to 01C1.
Load a random number into A
keep only the 3 lowest bits.
Load HL with the start of the "centre squares" table and jump to 017F.
Load B with 4 to look for the row containing the computer piece. IX contain the "Found" address and IY holds "NOT found"
Jump to the sub-routine.
Load HL with the start of memory.
DE holds the address of the row with the computer piece. Load the address into L via A and look at the first byte to see if it is zero by comparing it with 00.
Jump if it is zero
Otherwise increment DE twice

and jump to 01D1
The location is zero, so load A with 4 and place it in the location looked at by the HL register pair. Jump to 012E

Look for a computer piece.
IX contains the "Found" return address
IY contains the NOT FOUND return address.

TABLES:

at 0220:

01	04	07	01	02	03	01	03
02	05	08	04	05	06	05	05
03	06	09	07	08	09	09	07

at 0238:

01	01	07
07	03	09
03	09	

at 0240:

02	04	08
08	06	04
06	02	

This is turned into a memory location at 0158, 0159 & 015A. The value in location 0804 is then compared with 00 and if it empty, it is loaded with 04 and the program returns to SCAN with "Computer WIN" LED.

Random corner if first move at 0168

Computer looks to see if all 9 locations are empty. If they are, it places a piece

Note that this part of the routine is used by a lower section. This reduces the length of the program and saves duplication. Loading H with 08 sets the HL register to 0801 etc.

Looks for a centre square at 0196.

Computer looks for a centre square to

Looks for a '6' at 01A4

This is a 'trick' situation where the player is forming a 'fork.' Computer must play a side square.
This is done by loading the accumulator with the centre square (at 01B1) and detecting the occupier. If it is the computer, a random side is played.
If the computer is not in the centre, the program looks for a row containing only a computer piece and adds to it. This "throws the player off" and aims for a stalemate.

Looks for a '4' at 01E2

W

```
LD HL,0801  01EC  21 01 08
LD DE,0220  01EF  11 20 02
LD A,08     01F2  3E 08
LD I,A      01F4  ED 47
LD C,03     01F6  0E 03
XOR A       01F8  AF
LD (0811),A 01F9  32 11 08
LD A,(DE)   01FC  1A
LD L,A      01FD  6F
LD A,(0811) 01FE  3A 11 08
ADD A,(HL)  0201  86
LD (0811),A 0202  32 11 08
INC DE      0205  13
DEC C       0206  0D
JR NZ,01FC  0207  20 F3
LD A,(0811) 0209  3A 11 08
AND 0F      020C  E6 0F
CP B        020E  B8
JR NZ,0216  020F  20 05
DEC DE      0211  1B
DEC DE      0212  1B
DEC DE      0213  1B
JP (IX)     0214  DD E9
LD A,I      0216  ED 57
DEC A       0218  3D
LD I,A      0219  ED 47
JR NZ,01F6  021B  20 D9
JP (IY)     021D  FD E9

From 013D:
LD D,09     0248  16 09
LD HL,0801  024A  21 01 08
LD A,(HL)   024D  7E
CP 00       024E  FE 00
JP Z,0148   0250  CA 48 01
INC HL      0253  23
DEC D       0254  15
JR NZ,024D  0255  20 F6
LD D,08     0257  16 08
LD B,00     0259  06 00
LD A,88     025B  3E 88
OUT (02),A  025D  D3 02
DJNZ 025F   025F  10 FE
XOR A       0261  AF
OUT (02),A  0262  D3 02
DJNZ 0264   0264  10 FE
DEC D       0266  15
JR NZ,025B  0267  20 F2
JP 0017     0269  C3 17 00
```

found not found

HL is loaded with the first address of the display
DE is loaded with the start of the table
A is loaded with 8 for 8 loops of the program and put into reg I (I cannot be loaded directly)
C is the byte counter
Zero A at the start
Zero the TALLY location
Load the first byte of the table into A
Load this value into L
Load the value in the TALLY register into A
ADD the value looked at by the HL register pair
Load the result into the TALLY register
Look at the next byte in the table
Decrement the three byte counter
If it is NOT zero, jump to 01FC. If zero, go to next address (0209. Load the TALLY register into A and keep only the 3 lower bits.
Compare the result with register B
Jump to 0216 if the result is NOT zero.
DEC DE 3 times so that it looks the top of the 3 byte table.

Jump (IX) as a value has been found.
Load the loop counter into A and decrement it to see if the 8 loops are up.

Jump to beginning of routine
Jump (IY) as a value has NOT been found.

Register D is the loop counter
Load HL with start of display memory
Load the first byte into A
Compare with zero.
If the result is zero, jump to 0148
Look at next byte
DECrement the loop counter
Jump if the loop counter is not zero.
Create an 8 loop display
Load B with zero for longest delay
88 will turn on both Player WIN and Computer WIN LEDs.
Dec register B and jump to 025F 256 times.
Turn of both LEDs
Output zero to the LEDs
Dec register B and jump to 025F 256 times.
Decrement the loop register
Jump to 0267 if D is not zero.
Jump to 0017

Sub-routine for scanning rows, columns and diagonals to detect a particular condition at 01EC

The program jumps (IX) if the condition is found or (IY) if the condition is NOT found.
Register pair DE will point to the start location of the row, column etc. in the table at 0220.
There are 8 blocks of 3 bytes to test. DE is loaded with the start of the 24 byte table. The first byte is loaded into A (at 01FC) and this is converted to a location on the board via 01FD. Memory location 0811 holds the value of the TALLY and is zeroed at the start (01F9). The present value of 0811 is loaded into the accumulator and the value at the location on the 3x3 display is added to it. The result is loaded back into memory location 0811. The byte table is incremented and C decremented to the 3 locations we are adding together. The routine is looped 3 times and 0811 will contain the TALLY.

Looks for a 'STALEMATE' at 0248

The board is checked to see if any square is empty. If all are full, both WIN and LOSE LEDs flash 8 times.
This is an assumed condition as a Player WIN, or Computer WIN has already been tested.
A short 8 loop routine is executed to turn on both LEDs and the program returns to SCAN.
Bit 0,C has not been set so that button can be pushed the second time to view the stalemate condition.

After playing against the computer a number of times you may want to improve its strategy. Its decision-making can be tightened up completely or partially, by removing its options and random choices.

Firstly you can limit its decision for a random move during its first move to one of four corners by changing 01E8 to FD 21 78 01.

Another gap can be closed up by looking for a player in square 6 (second row, right) and playing location 9 (bottom, right)
The program for this is:

at 01E2:

```
LD B,04     01E2  06 04
LD IY,026C  01E4  DD 21 6C 02
LD IY,0178  01E6  FD 21 78 01
```

Look for a computer piece.
IX contains the "Found" return address
IY contains the NOT FOUND return address.

at 026C:

```
LD A,(0809) 026C  3A 09 08
CP00        026F  FE 00
JP NZ 0155  0271  C2 55 01
LD A, (0806) 0274 3a 06 08
CP 01       0277  FE 01
JR Z,0283   0279  28 08
LD A,(0808) 027B  3A 08 08
CP 01       027E  FE 01
JP NZ,0155  0280  C2 55 01
LD A,04     0283  3E 04
LD (0809),A 0285  32 09 08
JP 012E     0288  C3 2E 01
```

Load A with 9th square
Is is empty?
Jump to 0155 if it is not empty
Load the 6th square into A
Is it a player piece?
Jump to 0283 if it is
Load A with the 8th square
Is it a player piece?
Jump to 0155 if it is not.
The 9th square is empty (as found out at 0271) so load it with a computer piece (04) and jump to 012E

This will leave only one or two ways of winning. We won't remove these as that'll remove all the fun.

Here are some of the original games played by the author with the staff as he worked out the strategy for winning. For over a week the staff were engaged in "playing games" so that each "Loop-hole" in the game could be closed up.

With fun like this, it's no wonder the staff have never had a sick day off. But that's the fun of electronics - you never get sick of it and its different every day. This program has taken hundreds of hours to produce and we have enjoyed every minute of it.

or computer is in the centre square. Depending on the outcome, it will respond with a corner if the computer is in the middle, or look for a row containing just a computer value, if the player is in the centre.

Finally the computer looks for a row or column containing just a computer move. This condition will only be looked at after all the other statements are not met and will apply when the game is part way through. If the condition is found, it places a computer value in the first available square. If not found, a random square is filled.

These tests are sufficient to produce an intelligent reply to all the situations the computer will face. The strength of play is not extremely high and allows for a number of ways in which the player can achieve a win. If the game were to be tightened up, the response would become more predictable and much of the fun would be lost.

HOW THE PROGRAM RUNS

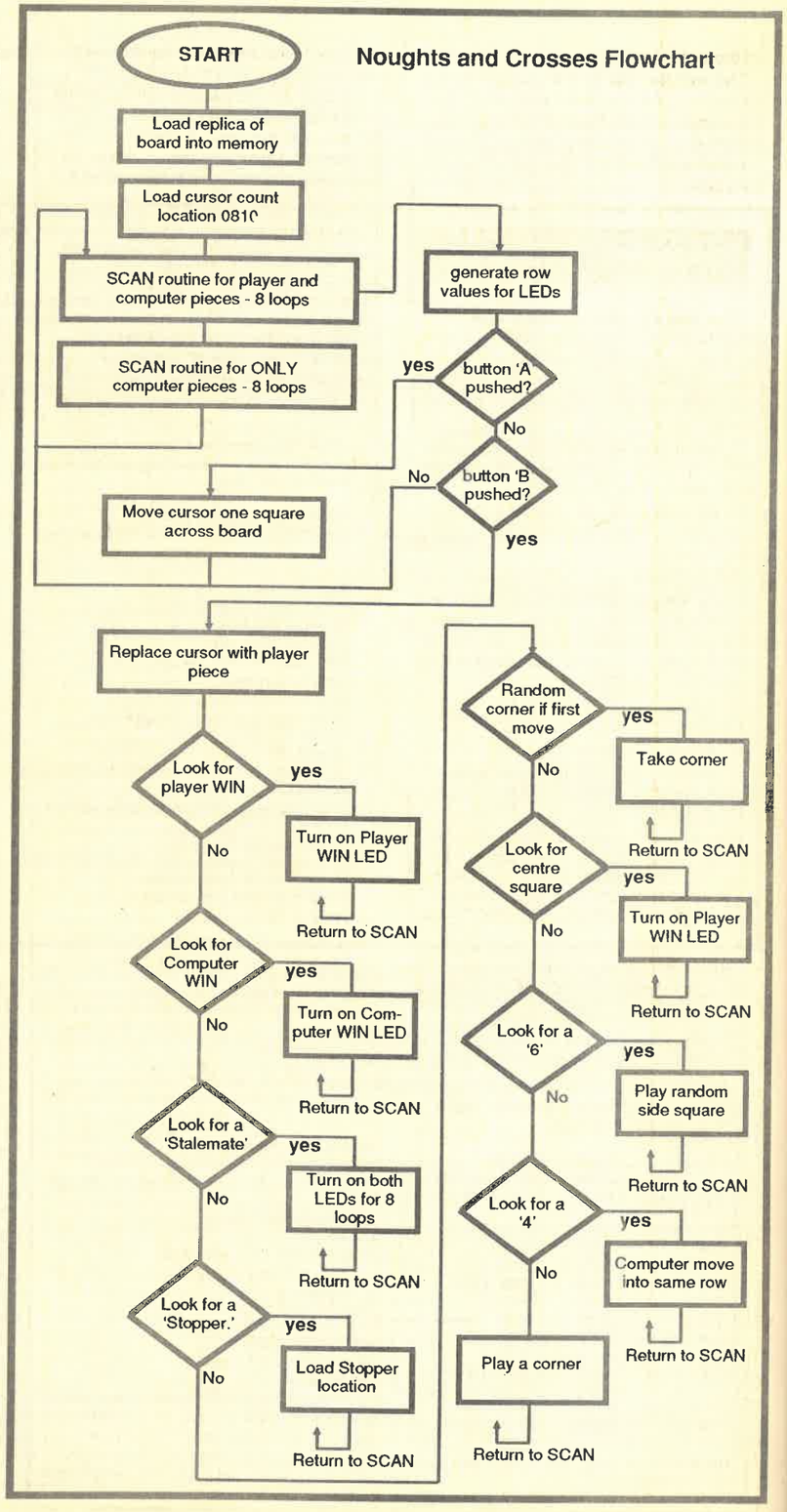
The program is running all the time. Most of the time it is executing the SCAN routine which lights up the LEDs in the display. The top row of LEDs are the first to be illuminated. The routine determines if the first LED is to be lit by looking at memory location 0801. If this location is not zero, 04 is added to a tally register (so that the left-hand LED will be lit - it has nothing to do with the computer value of 04).

Then the next memory location is looked at, and if it contains a value, the tally register is incremented twice. Finally the third location for the row is looked at and if it contains a value, the tally register is incremented once.

This means the tally register will contain a value from zero to 7 and when an output statement is executed, will turn ON the appropriate LEDs. The display is arranged as a 3x3 matrix and this requires 6 output lines. The two remaining lines of the latch go to the Player WIN and Computer WIN LEDs.

If each led in the display were to be accessed separately, it would require 9 lines, but this is beyond the capacity of a single latch. To turn on the top row of LEDs, output O_4 (the 5th line) must be turned ON and also any of outputs O_0, O_1 & O_2.

This means the top row of LEDs turn on TOGETHER. Each is individually sourced via an output line and sunk via a single transistor. This gives the LEDs maximum brightness when being multiplexed. Creating the values to turn on a row of LEDs takes a lot of clock cycles and while the computer is creating the values for the middle row, the top row remains illuminated. That's the purpose



of the output latch.

The middle row is the second to be turned on and then the bottom row. This sequence is repeated for 8 loops then the program jumps to an almost identical routine which detects only the computer locations.

By jumping back and forth between the two routines, a blinking effect is generated on the players LEDs, while the computer LEDs appear to be ON all the time. If the speed control on the Microcomp board is turned down, the scan process can be detected and all the LEDs will flicker!

During each of the row-generating routines, the program passes an instruction which looks at the value of the input port.

When a switch is pushed, the 'IN' instruction (see address 00DC and 012E

SCANNING THE DISPLAY

After playing a few games you will see how the display turns some LEDs on fully and allows others to flash. This may lead you to think that some LEDs are constantly illuminated - but you will be mistaken!

If you turn the speed down, all the LEDs begin to flash! Some flash repeatedly while others turn on and off slowly.

But it's not until you reduce the speed even further that you realise the miracle of scanning.

As we have said in the body of the article, we cannot access all the LEDs individually as this would need 11 lines. We only have an 8 line latch and so we must resort to the trick of multiplexing.

MULTIPLEXING is SCANNING and to see the scan in action you will need to place a 22n capacitor across the 1n in the oscillator, near the edge of the board. You can do this under the board or by poking the leads into the holes on the top of the board.

This will reduce the scan-rate so that you will be able to see the top row comes on, then the second row and then the bottom row. This is really amazing, only ONE ROW is on at a time and the scanning direction is down.

Armed with this information, the next thing you need to do is work out how the scanning information is sent to the display. To do this you need to make a list of the data lines on the EPROM, for D_0 to D_7 (see data table 1), then list the connections to the 74LS273 for D_0 to D_7. Each input line of the latch chip has a corresponding output line and this gives 8 outputs, O_0 to O_7.

If you look at the way the lines have been arranged you can see that one line goes to column 3, one goes to column 2, one goes to column 1 and one goes directly to the "Player WINS" LED. These 4 lines are half a byte and are called a NIBBLE, so that the lower nibble controls the columns. The upper nibble controls the rows via a set of 3 transistors and this sets the direction of scan as either up or down, according to the program.

(One of the bits from each nibble is connected directly to the 'Player wins' LED and one to the 'Computer wins' LED but this will not concern us at the moment.)

Refer to the second table to see how the values in the program (called data values) such as 01, 02, 04, 08 etc are outputted to produce a HIGH on the 8 lines.

To turn on the left-hand LED in the top row, bit 2 (output line O_2) and bit 4 (output line O_4) of the output latch must be turned ON.

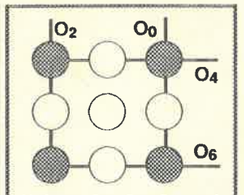
The transistor inverts bit 4 to sink the LED while bit 2 sources it. Bit 2 has the hex value '04' (see data at address 0240) and bit 4 has the hex value '10' and thus the value required to turn on the first LED is 04 + 10 = 14.

If you want to turn on the three LEDs in the top row, the value will be 04 + 02 + 01 + 10 = 17.

If you want to turn on the first LED in the top row and the last in the bottom row, you can't!

Let's look at the reason why.

To turn on the top left-hand LED, the values are 04 and 10. To turn on the bottom right-hand LED the values are 01 and 40. Putting these values together we get



If lines O_2, O_0, O_4 and O_6 are activated at the same time, the 4 corner LEDs are lit, not just the diagonals.

Hex value		Binary value		Effect on display
Bit 0	01	0000	0001	drives col 3
Bit 1	02	0000	0010	drives col 2
Bit 2	04	0000	0100	drives col 1
Bit 3	08	0000	1000	"Player wins" LED
Bit 4	10	0001	0000	drives row 1 trans.
Bit 5	20	0010	0000	" row 2 transistor
Bit 6	40	0100	0000	" row 3 transistor
Bit 7	80	1000	0000	Computer wins LED

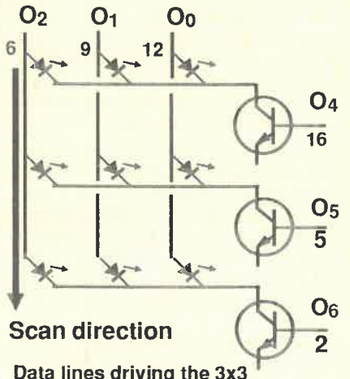
Table 2. The connection between each bit, its hex value, binary value and how it drives the display.

55. If you look at the circuit diagram carefully you will see that these 4 lines will illuminate the four corners as shown in the diagram below, and not just the two LEDs as required. That's why we have to scan the display.

We have to turn on the top row first, turn it off, then go to the bottom row and turn on the required LED, then off again.

PIN DATA:			
EPROM		74LS273 input	74LS273 output
D_0	9	13	12
D_1	10	8	9
D_2	11	7	6
D_3	13	14	15 player wins
D_4	14	17	16
D_5	15	4	5
D_6	16	3	2
D_7	17	18	19 computer wins

Table 1. Pin numbering for EPROM and latch.
The output pins of the latch are shown in the diagram below:

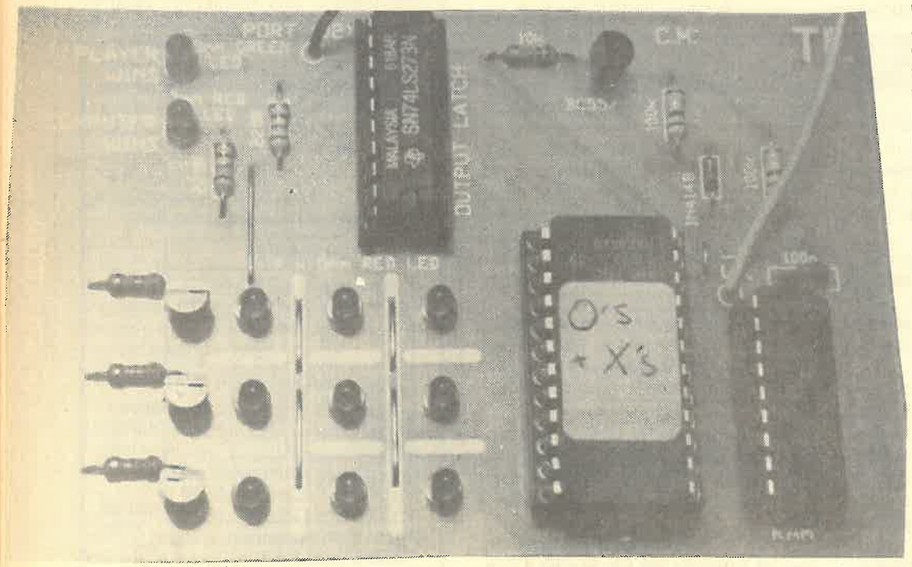


Data lines driving the 3x3 display

If this is repeated very quickly we achieve the illusion that the two corner LEDs are illuminated. It would be so easy if the LEDs could be individually accessed, but the reason we decided to scan the display is to explain the concept of scanning.

All the values for the display are generated by the program. The tally register 'D' and the instructions ADD A,10 ADD A,20 and ADD A,40 produce the necessary values.

in the program) places a value in the accumulator. A particular bit is tested to see if button A has been pressed and if it has, the program branches to a sub-routine to bring a cursor LED onto the screen. This LED can be moved into each of the 9 locations. At the same time a bit is set in the 'C' register which allows only one increment across the screen for each press of the button.



This is a debounce bit and is only reset when the program detects the button is not pressed. The cursor LED advances across the display and will jump over any LEDs which are displaying.

When button B is pressed, the cursor LED changes to a fixed player LED and the program advances through a number of tests which have already been described. Debounce for switch B is a single loop which is executed if the button remains pressed after the strategy section has been run.

SWITCH DEBOUNCE

One of the important elements of a program is the input detection routine. It is the interface between the outside world and the computer. Quite often this interface is a switch or push button or key from a keyboard.

All these devices are mechanical and although we may think that a switch opens and closes without any problems, a computer sees a switch differently.

To a computer, a switch takes a long time to open and close and in the process of doing so, the contacts "bounce." We can draw a parallel with an electric light switch. Every time we turn a light on, we hear the switch arc and spark. This is due to the contacts opening and closing very quickly as the switch operates.

If you connect this type of switch to a computer program, you will get many counts for each of its operations as the computer is strobing the switch many times per second and it will record each of the arc and sparks.

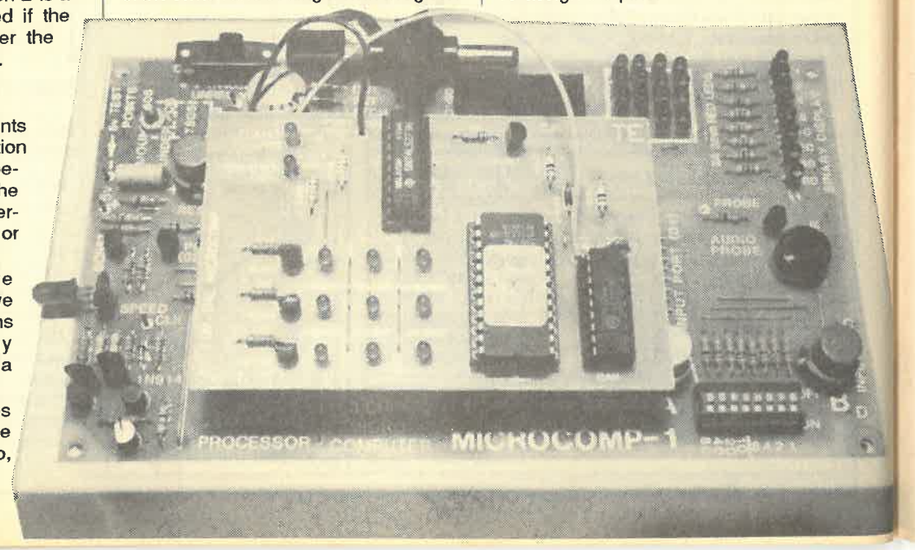
This is obviously not acceptable as it will produce a false count and the only way to overcome it is to produce a key scan program that will create a delay between opening and closing of the switch.

It does this by picking up the first close of the switch and setting a bit in a register and then waiting a while to see if the switch has been released.

Our program includes a simple switch debounce. At 000C, register C is loaded with zero. At 0093 the input port is strobed and button A is detected if it is pressed. (Button A is bit 7.) At 0095, bit 0,C is still zero and so are bits 1 and 2 so the program goes to 00A5 where bit 7 of register A is found to be NOT ZERO and so the program jumps to 00F1 where bit 0 of register C is SET. The program then goes through various other routines, strobing the input port for an update of button A and passes 0095 where it finds bit 0,C is NOT ZERO and jumps to 00AB where it checks the current condition of button A. If it has been released, bit 7, A will be zero and so bit 0,C will be reset at 00AF. The program carries out some hundreds of instructions between the time when it detects a button-push and when it is released and this provides the debounce. If the clock frequency were higher, we may need some additional debounce timing such as a "count register" so that the time between each strobe of the input is long enough to prevent false counting.

WIRING THE CHIPS

If you look at the pin numbering for the 74LS273 on the main board and compare it with the '273 on the add-on board, you will find they are different. This is ok as you can call any data line D_0 and this will make its corresponding output line O_0. For instance, if you make pin 8 D_0, pin 9 will be O_0. Try not to do this however if the layout of the board cannot allow for standard pin assigning, you can re-assign the pins.



CONSTRUCTION

The PC board is designed to fit onto the EPROM socket of the Microcomp and carries all the components required for the game. A wire-wrap socket and component carrier are soldered together to create a stand-off so that the Noughts and Crosses board is about 2cm above the mother board.

Before the 24 pin sockets are soldered together, the wire wrap must be fitted onto the board and soldered into position. The overlay indicates the position for each component and 4 links are required to connect the LEDs into the display.

These links form part of the Noughts and Crosses framework and may be hard to see at first. Three other links are required to complete the rest of the circuitry.

It is advisable to use IC sockets for the memory and latch as these chips can then be used in later add-ons. That's the overall picture of construction. Now down to specifics:

The first item to be fitted to the board is the wire-wrap socket. This is pressed firmly onto the board so that the pins extend FULLY from the under side. Solder each pin carefully and strongly as the board will be pulled in and out of an IC socket many times during its life- time and stress and strain will be placed on these pins.

A component header (or component carrier) must be soldered to these pins as the wire-wrap pins are too thick to fit into the IC socket. If a component header is not available, an IC socket can be used; provided you can get to the 'grippers' and solder to them, without totally damaging the socket.

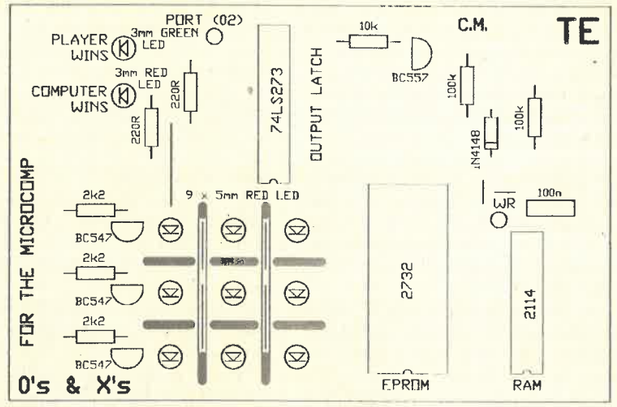
Using an IC socket like this is not as good as a component header as it is not as strong, But it is much cheaper and an acceptable alternative when headers are out of stock.

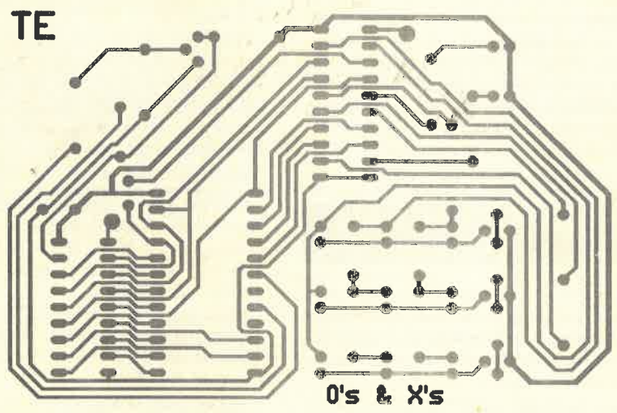
Next fit the jumpers. Four of these are on the display, one is above it and one is on the other side of the EPROM.

Next fit the 8 resistors and signal diode, making sure they touch the board both before and after soldering. Next fit the 3 BC 547 transistors and one BC 557, fit the IC socket for the latch and memory chip and then the eleven 5mm red LEDs.

Make sure the cathode lead goes down the hole marked on the board and if you are not sure which is the cathode, use a test LED and resistor to identify it.

This is done by connecting the pair to a 3v battery and seeing the LED turn ON. You cannot look into the LED or tell by the length of the leads as some LEDs are around the wrong way. You can use LEDs other than red if you wish or maybe select orange and green for the win and lose.





Full-size artwork for Noughts and Crosses

Two flying leads are required to connect the board to the rest of the computer. These go to PORT (02) and WR. They are clearly marked on both the Microcomp and sub-board.

Use a short length of hook-up flex for each and connect a female matrix pin connector to the end and cover it with heatshrink tubing to make the ends rigid. Then you will have no difficulty in fitting and removing the leads.

Insert the three chips, fit the board into the EPROM socket on the Micro-comp, attach the port lead and memory lead and the project is ready for testing.

IF IT DOESN'T WORK

If the board doesn't work, you have to decide if the fault is a hardware or software problem.

You must make sure the Microcomp board works perfectly by trying the programs contained in the "Microcomp-ROM."

Make sure the input switches are off as the 7th and 8th switches are connected to the buttons and will prevent the O's and X's program from working.

To see if the O's and X's EPROM is causing the fault, replace it with the Microcomp-ROM (in the O's and X's board) and switch ON the first switch (switch 1) at the input port.

The Microcomp EPROM will not make any sense on the 3x3 display however some of the LEDs will illuminate when the programs are executed.

For instance, when switch "1" is turned on, the "Computer Wins" LED will illuminate.

At the same time bit A in the left-hand display (of the 7-segment display) will illuminate and a tone will be emitted from the speaker.

This proves the pins on the socket and the stand off on the 'add-on' board are soldered correctly and the two jumper leads are connected to the mother board. It also shows the latch chip is working.

If you run some of the other programs such as "00-99 counter" or "Running Letter Routine," you will see the LEDs in the display come on. If they don't, the fault will lie with the sinking transistors or some of the 9 LEDs being around the wrong way.

You can check the O's and X's ROM to see if it has a program by placing it in the Microcomp socket. The right hand display will come on to show that the program is scanning the display and if you turn the speed down, the segments will flicker.

This is about as far as we can go with simple tests and you will now require test equipment such as a logic probe, continuity tester and multimeter.

We have already described the first two in our Talking Electronics magazine and you can purchase the kits from us.

They are invaluable pieces of equipment for troubleshooting digital circuits and especially a computer project like this, where many signals are flowing around the circuit at the same time.

The first piece of equipment to use is the continuity tester. Turn off the power to the Micro-Comp and remove the 'add-on' board. Test each line for continuity, from the IC socket to the end of its run. Then test each line against all others, for shorts. This will involve hundreds of tests but the continuity Tester makes it very quick as you can probe around very quickly, listening for a beep to indicate a short or continuity.

You should then fit the board to the Microcomp and test from the Z-80 to the EPROM (on the O's and X's board) to make sure the wiring goes through the plugs and sockets.

If you have not found the fault by this time I think you are going to have problems.

The only thing you can do with the Logic Probe is detect the presence of a signal on various lines, when the project is turned on.

Even though the Microcomp is the simplest computer you can get, the signals flowing on the lines are very complex and the timing for the signals is very involved and exacting, even the clock frequency is very low.

Even detecting when the RAM is activated is a difficult task. So, don't expect too much assistance from the Logic Probe. The only thing it's going to do is tell you if a signal is on a particular line.

If you cannot get it to work, you can send it in to us for checking. There is a small charge for this but it's better than giving up.

Frankly, I will be quite surprised if it doesn't work first go as I have had mine stored away for 5 years and when I got it out to finish the article, it worked first go.

Let's hope yours works, and now it's time to play.

HOW TO PLAY

Press RESET to start the program. Press button "A" and on the second press, the cursor LED will appear on the screen. Press "A" until the LED is in the desired location, then press "B" to 'fix' the LED. The computer will then make its move.

Press button "A" again for your second move and continue until either "Player wins" or Computer Wins" LED is activated.

ACTIVATING THE MEMORY

Noughts and Crosses requires a small amount of memory to hold a replica of the playing board, plus a few extra locations to hold temporary values during the course of play.

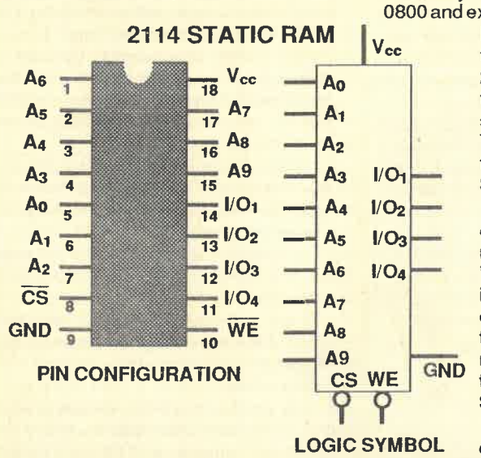
A 2114 memory chip has been used for this and is capable of holding a lot more information than we need. They are very old technology devices and are no longer manufactured but can be obtained from old computer boards.

A 2114 is an N-Channel Silicon-Gate MOS static RAM and is structured to accept 1024 4-bit words. A 4-bit word is called a NIBBLE and so a 2114 will accept 1024 nibbles.

1024 can be converted to 'pages' where one page is equal to 256 nibbles. Thus we can think of the 2114 as holding 4 pages of nibbles.

As one page is equal to FF nibbles, this is the best way to think about memory when you are writing programs, so that you will be aware of the beginning and end of a page.

Memory in this project is decoded at 0800 and extends for 4 pages. This gives RAM at 0800 - 0BFF. 0800 is the first address above the 2716 EPROM (or the particular half of a 2732 as selected on the Microcomp). The 2114 is accessed via two lines. These are $\overline{CS}$ Chip Select and $\overline{WE}$ Write Enable.



The 2114 is a 4096-bit static Random Access Memory organised as 1024 (4-pages) of 4-bit words.

Firstly the chip is placed in either the READ or WRITE mode and then selected or 'activated'. The READ mode is when information is taken out of the chip and WRITE is the operation of placing information into it. The chip is then turned ON via the Chip Select line.

Exactly when and how the chip is placed in one of its active states requires a knowledge of signal levels and timing. The Write Enable is indicated by $\overline{WR}$ and this indicates the chip can be written into when the line is LOW. Chip Select ($\overline{CE}$) indicates the chip is active (turned ON) when the line is LOW.

Sometimes different symbols are used for these lines but the result is the same.

The chip is addressed at 0800. At this address (and for the next 2k of memory - if 2k is fitted to the Z80), line A11 goes HIGH. Thus we can use this line in association with the Memory Request to select memory. And this is what we have done. By simple combining of these two lines we have created an "OR" situation which will select the chip whenever a 'load into memory' or 'read memory' statement is encountered and providing the location falls within RAM space.

A11 and Memory Request are combined via a transistor because A11 goes HIGH and Memory Request goes LOW when the program statement is encountered. From these two lines we require an output which goes LOW and this is what the memory decoder transistor produces.

The other line to the memory chip is the Write line. This goes LOW when the program statement is encountered and thus it can be connected directly to the chip. The other two chips on the board are the EPROM and latch. The EPROM holds the program and is only partly filled in this project. The program is short enough for you to type in yourself and full details are given in this article.

MORE PROGRAMMING IDEAS

There is basically two ways of producing any type of intelligent game. One is to use lots of logic gates such as AND, OR, NAND etc and other chips to drive a display.

The other is to design the display and write a program so that a microprocessor does all the work.

I know the choice I prefer.

The microprocessor program is much easier to create and much more effective in its appearance. Once you have designed the hardware, a program gives you complete freedom to create any type of effect you like and you can include levels of skill and strategy to make the program appear intelligent.

But before you start to write the program you must sit down and work out what you want it to do and how it will appear on the display.

Each section of the program has to be written separately and viewed on the screen to see how long the effect takes, in real time. For instance, the flashing of the LEDs must be adjusted so that the flash rate is correct.

You may need to adjust the length of the routine and check the clock speed to get it exactly correct.

Quite often you need to include a delay value in the routine to slow-down the loop-time so that the flash rate appears at the right speed.

In our case we have a very low frequency oscillator (the transistor oscillator on the main board) and we had to keep the routines as short as possible to prevent screen-flicker. (Slow down the clock to see what we mean.)

With a computer program you can modify it so easily to suit any situation. For instance, the scan could be designed to run from the bottom of the display to the top, or the program could be designed to lose every time, without having to change any of the hardware.

You can also produce "computer thinking time" by including a "do-nothing" loop that simply wastes time. This way it will appear that the computer is thinking.

The advantage of using your skills to produce a program is obvious. Everyone knows how to play a game and can appreciate the skill needed to make it run, and win!

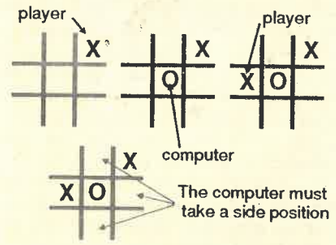
Programs are kept as short as possible by creating sub-routines that can be called by the main program. Once the sub-routine is executed, any results that are generated are stored in a register or in RAM.

Because a program consists of lots of jumps and calls, it is often very difficult to see how it is being executed. To make it easier, we have identified most of the jumps in the O's and X's routine and you should go through it ONE INSTRUCTION AT A TIME and follow the jumps to see how they are executed.

If you were to remove all the sub-routines and loops and stretch the program out to its full length, it would occupy more than 20 pages! So, it's essential to create lots of sub-routines.

Once you understand the running of a program, the next phase is to learn how to convert a decision or problem into machine language (machine code) so that the processor is able to understand it and act on it.

Even the most difficult decision can be broken-down into one or more very simple statements or tasks. Take the problem of solving the "fork" situation in the game. The diagrams below show how this comes about:



The program is required to detect a fork and play a side square.

This is one of many "set-ups" that the program must detect and so the data can be prepared and put into a common routine (an algorithm) to generate a result.

Although there is no pressure in a fork situation, a move to the top corner by the player will win the game, so the computer must look for a "6." (player=1, computer=4, player=1) and at 01A4 the "look register" (register B) is loaded with 06 and the 8 blocks of 3 bytes are checked to see if any row, column or diagonal adds up to 06.

This is done at 01EC where HL is loaded with the first memory location for the board. DE is loaded with the address of the first byte in the table at 0220 and A is loaded for the 8 loops of the program (this is then stored in the I register as it cannot be loaded directly with a value).

C is loaded with 3 for the number of bytes we will be working with and "A" is zeroed at 01F8 and loaded into the TALLY location (0811) to start the tally at zero.

This sets up the routine and at 01FC the value at 0220 is loaded into the accumulator (on the next pass, the value at 0221 will be loaded etc).

This value is then loaded into L so that HL will point to memory location 0801 and at 01FE, the accumulator is loaded with the present value of the TALLY register (zero).

To the accumulator is added the value looked at by the HL register (square 1) and the answer is put back into the TALLY register 0811 (at 0202). The next byte of the table is looked at (at 0205), the 3-loop counter is DECremented and if it is not zero, the routine is looped.

After 3 loops (at 0209) the TALLY register is loaded into the accumulator and only the 4 lower bits are kept (at 020C). The result is compared with B (06) and if it is 6, the "table-looking" register DE is DECremented 3 times to point to the top of the particular row, column or diagonal (at 0211, 0212 and 0213) and the program jumps (IX).

If the TALLY register is NOT equal to 6, the program loops 7 more times and comes out JP (IY) at 021D with a "NOT FOUND" condition.

If "FOUND," the program jumps to 01B1 and looks at the centre square. If it contains a computer piece (01B4), the refresh register is loaded into the accumulator (we are using the refresh register as a random number generator) and ANDed with 7 to get a value from 0-7. The "corner table" (at 0240) is loaded into HL (at 01BC) and the program jumps to 017F where the value of L (40) is added to the accumulator then loaded back into L so that HL will look at one of the bytes in the table at 0240. H is then loaded with 08 (at 0182) and L is loaded with the value found in the table so that when A is loaded with the value looked at by HL in memory, (it will look at 0802, 0804 or 0806) and compared with 00 (at 0186) it will find the square empty.

The program then jumps to 0190 where it loads the computer piece into the square and jumps to 012E.

This seems to be a lot of instructions for such a simple operation but you have to get each routine ready for looping, counting, etc and work within the capabilities of the Z80.

Once you set up a general routine (an algorithm) it can be used to find other conditions and values, so the overall length of the program will be as short as possible.

There are many other ways of writing the program. One simple approach is to allocate 24 byte of memory to the 8 rows, columns and diagonals. At the end of each trio you provide a tally. You can then scan the memory for a particular requirement.

THE FUTURE OF PROGRAMMING

The future of programming is enormous. Programming is very similar to writing a book or drawing a cartoon. In all three you start with a blank sheet and prepare a story that follows through a series of chapters or cells.

It's generally a story that has never been written before and is an expression of your own ability.

The better you become at thinking and expressing yourself, the better you can present your ideas.

As you know, new cartoons are coming along all the time and so are bigger and better novels.

So too with programming. With the advent of smaller, cheaper and more-powerful microprocessor chips and microcontrollers, the possibility for new products is enormous.

Although the programming language we are describing in this series of articles is one of the older formats, we have decided to present it as it is the cheapest way to get into program writing.

Even though some of the newer microcontrollers are smaller and more powerful than the Z80, they are quite expensive to get into as far as programming is concerned.

But firstly let me explain the difference between MICROPROCESSOR and MICROCONTROLLER.

The Z-80 is a microprocessor. It reads instructions from a chip that holds the program. This can be either an EPROM or a one-time programmable ROM. It also requires another chip to hold values that will be changing during the execution of the program. This chip is called a RAM (Random Access Memory).

Then there's other chips such as a clock, input and output latches, and a few more to keep everything in synchronisation.

But imagine if all these chips were in the one package. Imagine how small the project would become and how convenient to put into a piece of equipment. Due to the enormous popularity of the microprocessor, this is now a reality. A single chip that holds its own program, executes it, has an area of memory to hold variables and is capable of accepting information from the outside world and displaying the result on a display or delivering it to an output device.

A complete package like this is called a microcontroller and although early versions were very simple, we now have a range that equals and surpasses the capability of the Z80.

It has been proven that the only way to go in this technological age is with the microcontroller. It will be the heart of all future designs.

Some of them come in a window version (EPROM version) but most are designed as one-time programmable devices for high-volume production runs.

Because of this, the EPROM versions are very expensive as very few are produced while the single-burn versions are extremely low-cost.

Even though they are very inexpensive, the catch is in the set-up cost.

A development system can cost as much as $1,000 to $5,000 for the assembler, disassembler, computer and multi-programmer. This is why very few individuals have ventured into the newer devices but those that have spent the money on development systems are enormously busy and successful.

We see microcontrollers in almost every new electronic appliance that comes on the market, from washing machines, alarms, ovens, VCR's, to talking clocks and keychains.

Even "GI Joe" has a has a controller on a chip the size of a match-head so that when you push the button on his backpack, he blares out "I've got you covered!" or something to that effect.

Even a $14 book with electronic sounds has a simple microcontroller looking after the sequence of sounds and keypad - there's no simpler way to do it.

If I can think of a new design, I can think of a hundred. A microcontroller could be used for speech in a dialling alarm, or for those who have difficulty in talking, in car safety, in medical devices intended to be carried on the person, in security and in lots of areas that have never been thought of.

If we can get a talking alarm clock for $30 and a 30-second talking keychain for $5, there's no limit to the cheapness for microcontroller products.

All it takes is for someone to get to it and think of an idea. Why not let it be you. Do you realise that a millionaire is produced every 20 minutes in the USA? There are a million millionaires in America and many of them became a millionaire by putting together a simple invention or producing a simple device.

Who knows how popular an idea can become. Look at silly putty or the super ball, or the whistling keychain, or the talking pen watch. All these ideas made a millionaire out of the inventor.

So, look at what could happen with the power of a processor in your hands.

At the moment it's an expensive door to open but the first step is to appreciate the enormous scope that programming offers, and try your hand at producing something simple.

There is no better place to start than with our modular system as it offers all the capabilities without the high cost. After all, one of the Nintendo games designers started with our TEC computer and once he mastered it, he showed his capability to the big boys and the rest is history. He has now been commissioned to produce 2 new Nintendo games a year!

Obviously the method of producing a video game is entirely different to hand-assembling a simple Noughts and Crosses game but it is the concept of programming we are attempting to get across.

If you have the capability of putting ideas into words, then programming is the best way of putting your electronic thoughts into reality.

Admittedly, you must take things slowly at the start and that's what we intend to do. In the previous pages we have shown you the simple stages of putting a program together so that you can decide if this is the field for you.

The best way to find out is to see if you can improve some of the programs we have presented. Try writing them in an entirely different way and when you can say "I can do better," you are on the right path.

Firstly you have to see what we have done, then rewrite the program so that it can be executed with fewer instructions.

Some of the instructions for the Z80 are extremely powerful and as you improve your programming skills you will use more of them.

This will bring you up to the generation of programming where you will use compilers and assemblers and produce structured programs for more-complex tasks.

Then you may be able to move into the area of controllers, called microcontrollers where a very small instruction set takes the place of over 700 instructions.

This family of chips is called RISC chips, for Reduced Instruction Set Computers.

We are currently investigating one such microcontroller and hope to bring out a development system that is as low cost as the Z80.

Until this eventuates, I suggest you try your hand at writing and designing programs so you can be ready for the real thing.