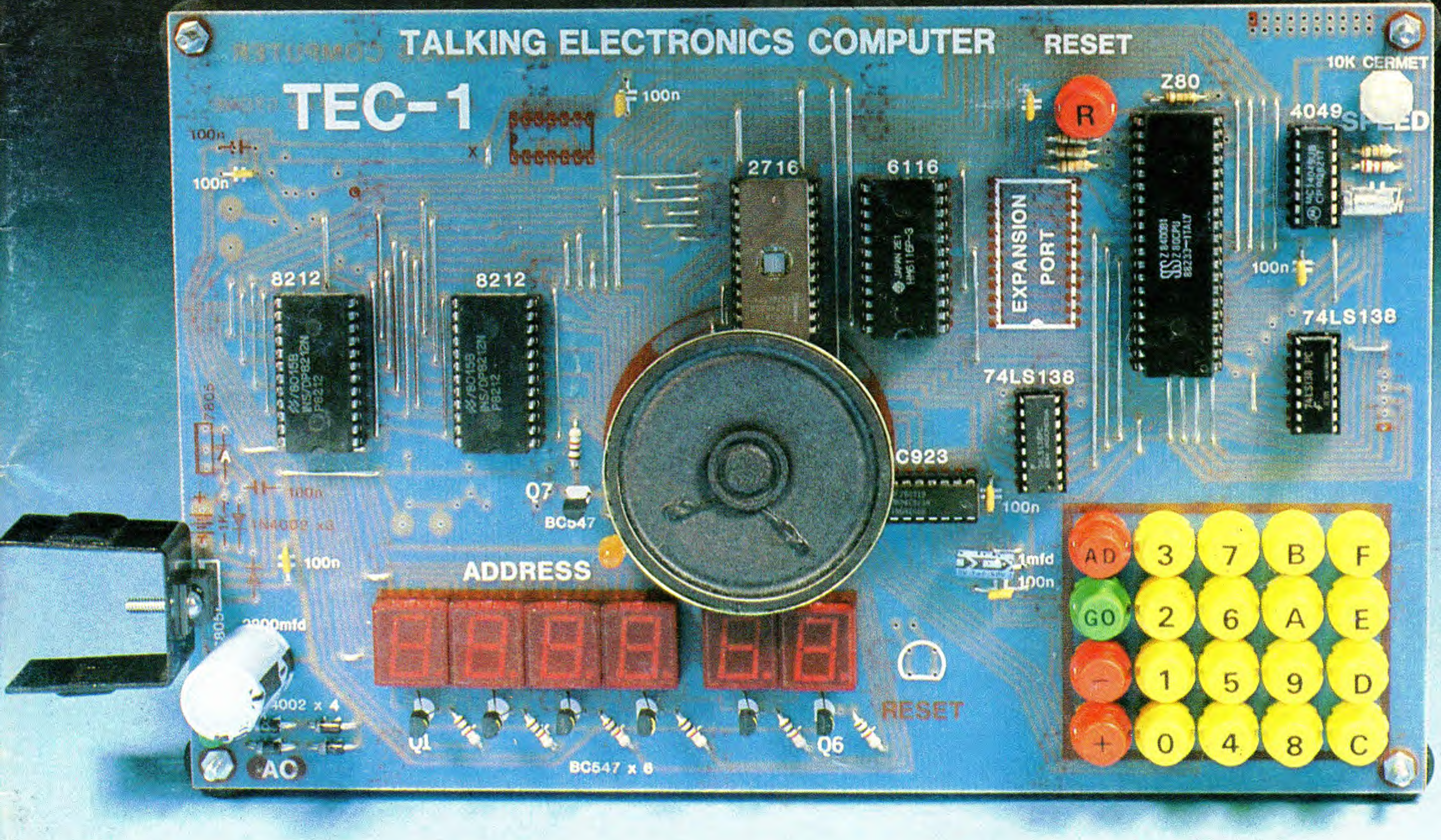
TALKING ELECTRONICS®

$1.80 ★

$2.40NZ

Issue No 10.

TEC-1 TALKING ELECTRONICS COMPUTER



LOGIC PROBE

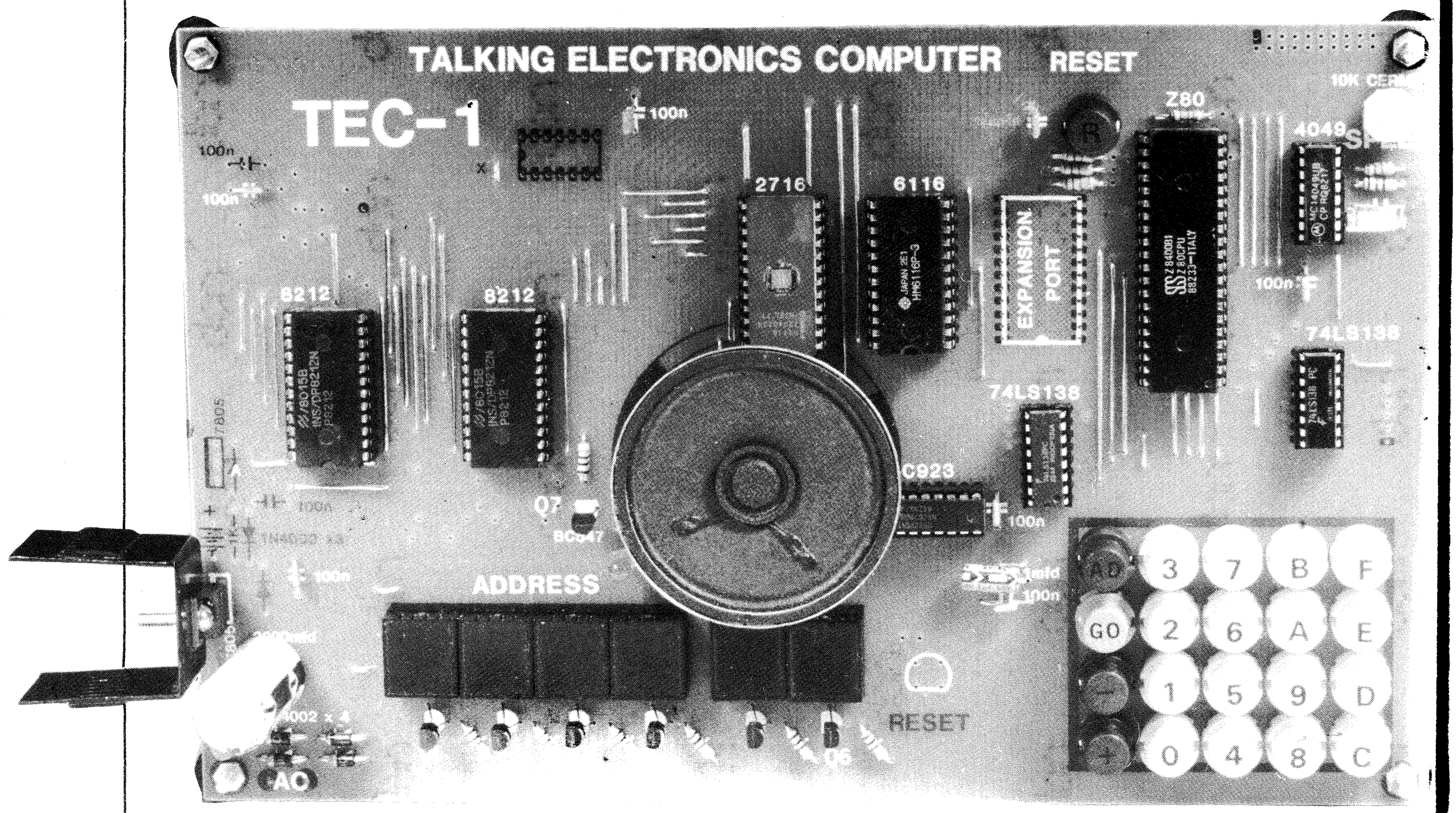
ROULED

COUNTER MODULE - ONE SHOT

TALKING ELECTRONICS COMPUTER

$87.30

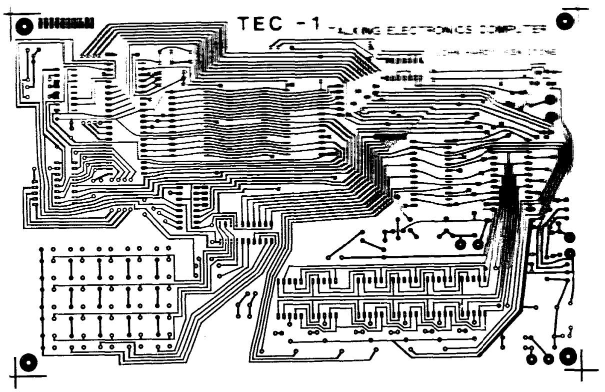
**Including all parts & PC board.
Post & Pack $4.50**



**A single-board Z80-based computer with enormous potential. You can provide controlling and sequencing of up to 8 different devices.
The whole circuit is accessible for expansion and adding control lines, making this the most universal programmable project on the market for under $100.**

PC Board $19.00 (post $2.50)
2716 - TEC-1 Programmed EPROM **$12.00**(Post $1.50)
Kit of Parts (including EPROM)**$68.30** (Post $2.50
All Parts & PC Board $87.30 (Post $4.50)
You will also need a 6v lantern battery (from your local hardware shop) or a 2155 transformer ($5.90) and a power lead or a 9v AC or DC Plug Pack rated at 500mA ($14.50)

Complete TEC-1 with transformer $93.20 (Post $5.50)
Complete TEC-1 with Plug Pack $101.80 (Post 5.50)



TALKING ELECTRONICS

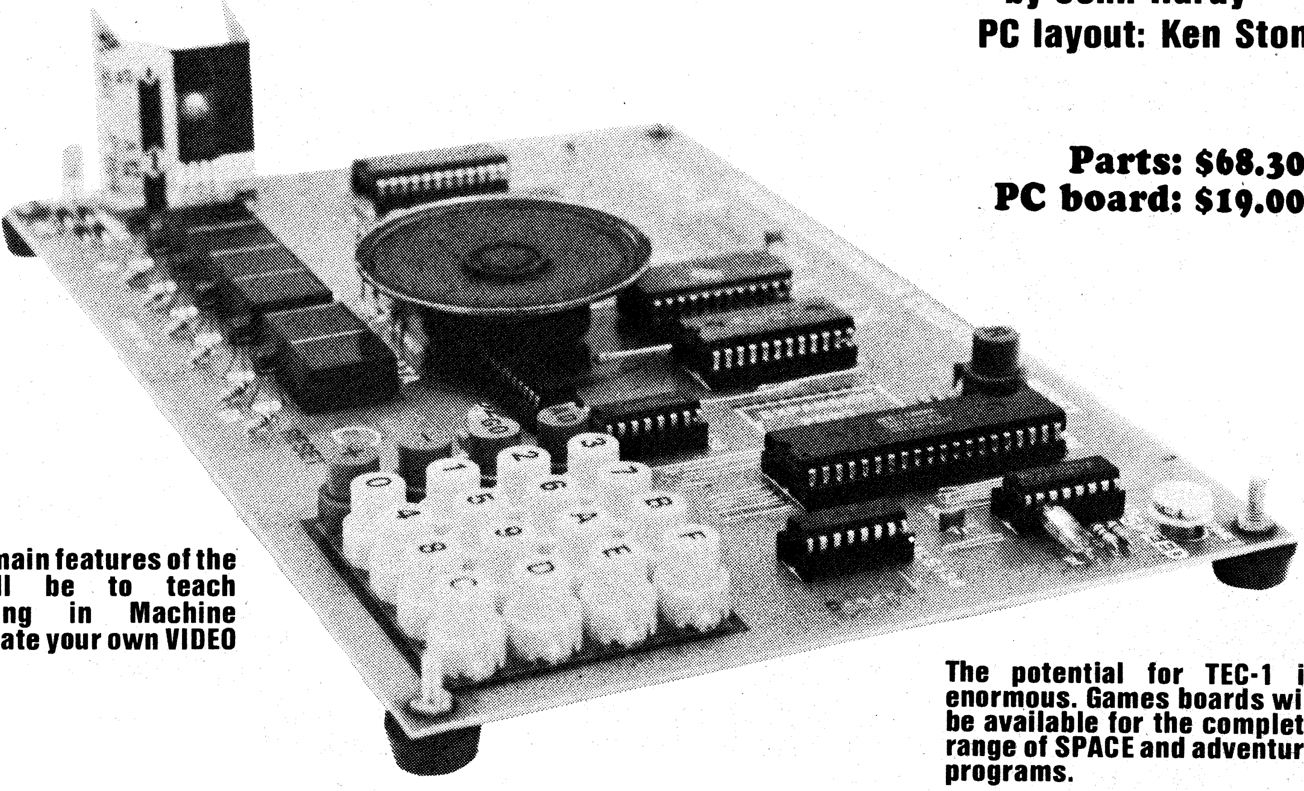
35 Rosewarne Ave Cheltenham 3192   584 2386

TEC-1

TALKING ELECTRONICS COMPUTER

—by John Hardy
PC layout: Ken Stone.

Parts: $68.30
PC board: $19.00



One of the main features of the TEC-1 will be to teach programming in Machine Code to create your own VIDEO GAMES.

The potential for TEC-1 is enormous. Games boards will be available for the complete range of SPACE and adventure programs.

If you think TALKING ELECTRONICS Magazine is a good place to start learning about electronics, you will find our TEC-1 computer absolutely fantastic.

We have spent many hours looking into the type of computers on the market and also computer kits.

Nothing has come up to the capabilities of the unit we are about to describe. And more important, you will learn the facts and operations of programming from ground level. We will assume you know nothing and thus place special attention to covering the meaning of every term and feature as it comes up.

The only requests we make are the following:

You must have already constructed at least 6 projects from Talking Electronics or equivalent magazines and it would be nice for you to have built the DIGI CHASER and say a couple of equally difficult projects such as the LOTTO SELECTOR and CLOCK.

This means you will be accustomed to soldering fine connections and know how to prevent making bridges between lands.

Fortunately the computer board has a solder resist mask and this means only the individual solder lands are exposed and they are already pretinned for easy soldering.

However some of the lands are close to one-another and a small low-wattage soldering iron is required for the project.

We have built 4 final designs and they all work perfectly. On one board we accidently created a solder bridge and this needed a little trouble-shooting, but we finally found it. So, for this reason, each kit includes a length of de-solder wick to mop up the surplus solder.

If you don't have a small soldering iron, fine solder and desolder wick, they will have to be obtained before constructing the kit.

BUYING THE KIT

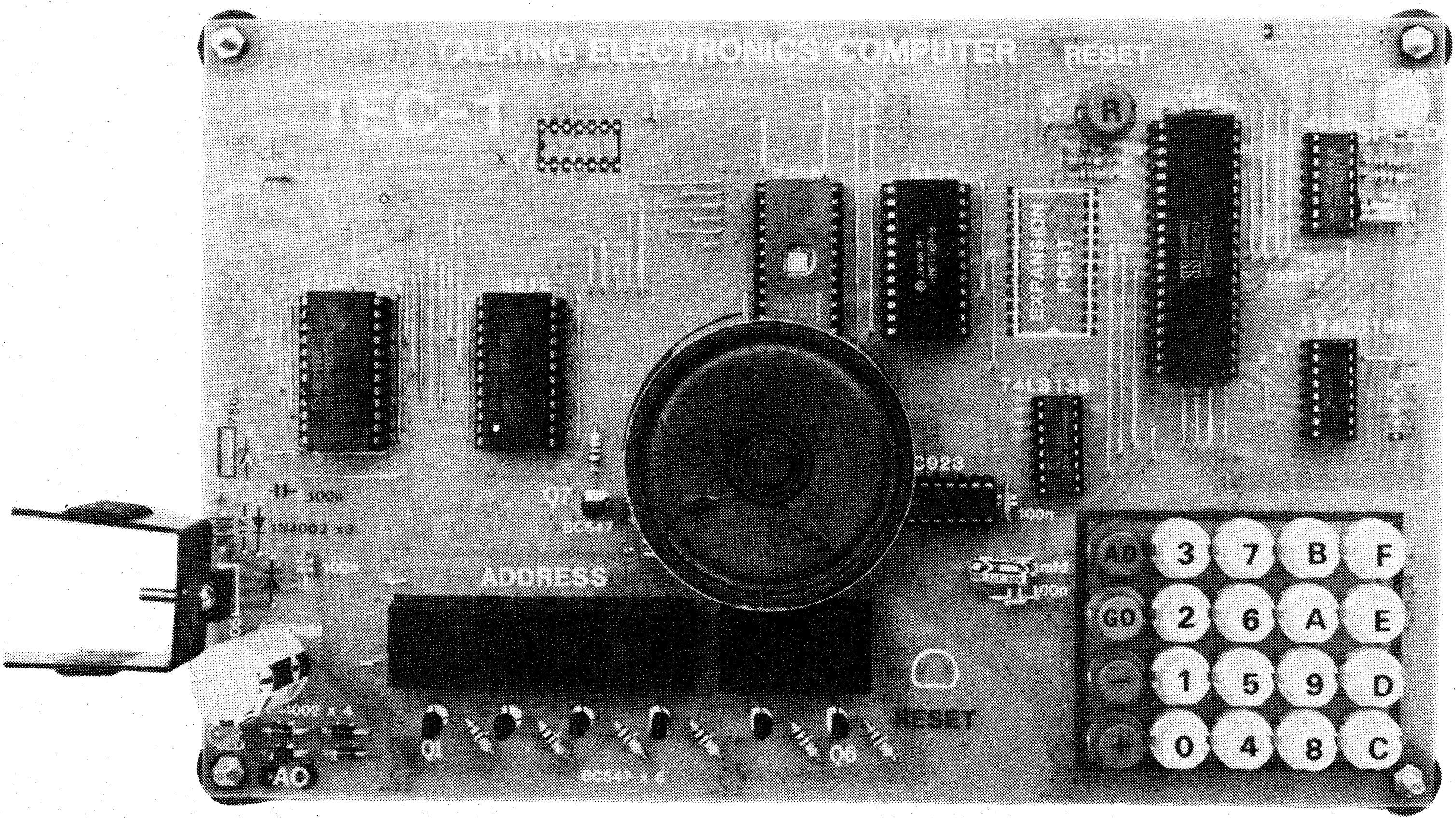
One thing you may not be aware of, is the need for one special chip.

Every computer requires a specially programmed chip so that it will start up and execute the correct operations. This chip can be likened to the BOSS in a work establishment. The chip we are referring to is the 2716 EPROM. You can buy it quite cheaply at any electronics store but unfortunately it is BLANK. And obviously it won't do a thing if you put it into a computer. To be of any use you will have to program it or write a program for it yourself.

Obviously this is way out of the question and so you have to buy one which is pre-programmed, from us.

For this service you have to pay a programmer's fee. A lot of time has

cont.. P 60....



TEC-1 IS A SINGLE-BOARD COMPUTER

AN OVERVIEW:

The TEC-1 is a single-board computer with readouts in the form of 7-segment displays. The complete unit is shown in the photograph. It contains its own on-board regulated power supply which needs only an AC input for the computer to be fully operational.

The key pad is constructed from individual switches inscribed with hexadecimal numbers 0 to F and 4 switches labelled AD for address, GO, + for incrementing the address and — for decrementing the address.

The computer will play a number of games as well as present the alphabet and all this is contained in the 2716 EPROM which is directly above the speaker. The TEC-1 can also be connected to 8 output devices and they can be turned on and off in any combination as determined by the program you write. This program is stored in the 6116 RAM and any information in this chip is lost when the computer is turned off.

The reset button above the empty expansion port socket will reset the computer to the first address location (0800) and by pushing the GO button TWICE, any program you have entered into the computer, will run.

The computer contains 2k of RAM and this is programmed in machine code. Machine Code is very memory efficient and has a fast execution rate, making it possible to create high-speed programmes for video games and multi-function controlling.

Extra memory can be added via the expansion port and this is added to a daughter board directly above the main board via a dip header plugging into the expansion port socket. This will increase the capabilities of the computer to 12k plus 2k of memory-mapped in/out ports.

The speaker has two functions. It gives an audible beep every time a key is is pressed and becomes the output when music or tones are being played.

All the names of the chips are written on the overlay of the board and in simple terms they provide the following functions:

8212 - drives each digit for the display via buffer transistors.
8212 - drives the segments A - G and the decimal points for the display.
2716 - EPROM (Erasable Programmable Read-Only Memory). This has been programmed by John Hardy and contains the brains of the TEC-1.
6116 - The RAM (Random Access Memory) into which you put your own program. The Z80 also uses it during the operation of some of the programs.
Z80 - The heart of the computer.
4049 - The oscillator or CLOCK for the TEC-1.
74LS138 - selects between ROM (2716) and RAM (6116).
74LS138 - Selects between keyboard and display.
The photograph has been illuminated from the rear to show the tracks on the underside of the board. Normally these tracks are hardly visible as they are hidden under the solder mask.
Notice how neat everything is presented. You can credit the superb layout to Ken Stone who recognises the importance of making a project look appealing. Note especially the few resistors and capacitors required for a fully digital project.

The 20k cermet pot has been specially chosen as it has a cover which is connected to the wiper contact so that the pot can be turned with your fingers. This controls the speed of the operation of the computer and you will be using this control quite a lot.

The output pitch of the notes will vary according to the setting of the speed control as will the difficulty of the games and the scrolling of the screen when the letter sequence is addressed.

All chips are mounted in sockets for a mumber of reasons:
1. It looks professional.
2. It makes construction easy,
3. It makes testing and replacement easy, and
4. You can test other chips in the sockets.

The 100n capacitors are miniature solid dielectric types, about the size of a match-head, and they are specially suited to removing any spikes generated by the chips or from the power supply.

The TEC-1 will operate from a 6v battery such as a 509 lantern battery or from the mains via a transformer. The 7805 regulator keeps the operating voltage at 5v which is absolutely necessary for the chips we are using.

The battery back-up arrangement means you can have a battery sitting beside the computer in case the power fails or if you wish to change the computer from one room to another. When the battery back-up is operating the complete TEC-1 is operating as it is not posssible to power-down the Z80 without it affecting the contents of the RAM.

There are two empty IC sockets as well as a number of rows of holes on the board. These are for later expansion and not used at this stage.

The RESET key can be positioned near the display is desired. It is connected via 2 jumper leads to this lower position.

Finally you will be pleased to know the TEC-1 doesn't need any TV monitors, additional keyboards or bulky power supplies. It is self-contained on the single PC board.

THE EXPANSION PORT

The expansion port socket can be used in two different ways.

1. It can be used to increase the on-board memory of the computer to 4k RAM by inserting a 6116 RAM with IC socket, directly into the vacant space.

2. Alternatively, the expansion port can be used to increase the memory on steps of 2k by adding a daughter board above the main computer board. This will take a row of 5, 6116 chips and a bank of latches. Each 6116 will provide 2k of RAM and this is one of the add-ons which will be described in the next issue.

Each of the chips on the daughter board is selected by a line from the 74LS138 (near the clock oscillator). It is known as an address decoder and the first decoded output selects the EPROM. The second output selects the on-board 6116, the third selects the expansion port socket. If a duaghter board is used, the first chip on the board is selected and so on until 7 lines are used. 5 individual wires must be taken to the daughter board to provide this selection feature. They are taken from the 5 unused holes near the 74LS138.

To give an indication of the amount of memory you may require, here is a simple guide:

Each 6116 will accept 2048 bytes of information. A normal program contains between 1 and 4 bytes of data per instruction and this means one 6116 will accept about 600 instructions! To hand-assemble a program of this length wuld take months. We have only 3/4 filled the 2716 and you will be amazed at the capabilities of its contents.

So you can see, 2k will be quite adequate for most purposes.

The main use for the expansion is when the microcomputer is colecting and storing its own data for later retrieval. In this mode the computer can use up an enormous amount of memory, very quickly.

Take an example of a music sequencer. 2k of memory will last about 10 to 20 seconds. Or an echo unit. This will last less than 1 second!

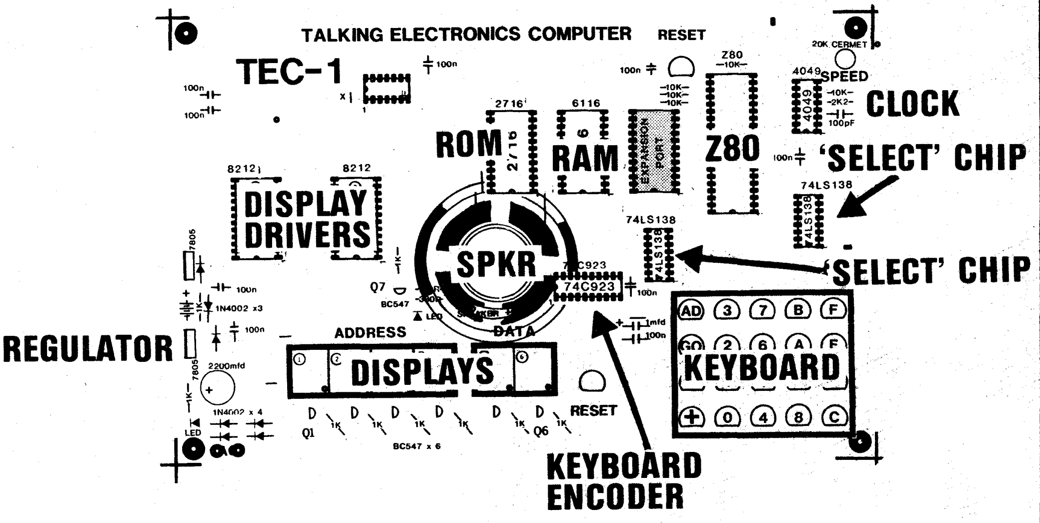
BATTERY BACK-UP

To use battery back-up, diode A must be installed. Connect the battery via a switch so that you can move from one location to another. Switch the battery OFF when the computer is using the mains power.

MARKING THE KEY-TOPS

The key tops can be lettered using LETTRASET. 16pt letters and numbers are used for 0 - F and 12pt letters for the AD and GO keys.
A coat of nail varnish will stop the lettering from wearing away.

THE FUNCTION OF EACH CHIP



TEC-1 is a complete microcomputer on a single PC board.

The function of each chip will become clearer after reading the text.

The most important concept is to understand how each chip is controlled by the Z80.

The above diagram shows where the ROM, RAM, Z80 etc are postioned on the board along with the other chips and devices.

been spent to get a set of instructions into the 2716 EPROM and each is individually filled from a master and verified, before it is added to a kit. This takes time and royalties are due to the designer, like the sale of a book or record. This accounts for its high cost.

We have called the EPROM a 2716 but actually it is a 2716-MON-1 EPROM, indicating it contains a program.

This is the only expensive chip. All the others have been chosen for their low price and availability. This is the way we approached the design of the computer. We looked at the price of each component and arranged the design around the low priced items.

The only components special to the TEC-1 are the EPROM and the printed circuit board. All the other components can be purchased at major electronics shops. The only advantage with buying a kit is the saving in time and frustration.

It would be very rare indeed for you to be able to buy all the components at one electronics shop. And so you will have to spend time and money in the hope of saving money.

In addition, there are a couple of pitfalls for the inexperienced. For instance, 4049 chips made by Fairchild should be avoided. They do not work in our situation. Also the push buttons should be the type suggested as one of the links inside the switch is used to create the wiring for the matrix.

The price structure for each kit is broken up as follows:

1. The Printed Circuit Board.
2. The 2716 MON-1 EPROM.
3. The kit of components.

The only other parts you will need to purchase are a 2155 transformer and power lead or a 9v DC plug pack rated at 500mA. You can, of course, use a lantern battery. This will be sufficient to operate the computer for about 5 to 8 hours, but will prove to be a very expensive way of running the TEC-1.

WHAT THE COMPUTER WILL DO

When you are going to spend a lot of money on a project and a considerable number of hours in its assembly, it's nice to know what the project will do.

Here is a summary of the first stage of the capabilities of the computer. Apart from the obvious experience gained in assembling a computer, the TEC-1 will make you aware of the chip-types required to create a complete system.

You execute some simple programs which use pre-programmed information from the EPROM and display it on the screen. If we take the letter program for instance, you create a single static letter,then add another letter and enable them to run across the display. Finally you create your own words and sentences which can be made to pass across the display at a rate determined by the setting of the SPEED control.

But most important you learn some of the instructions necessary to write your own programmes.

You also learn to increment and decrement the memory address to look at the contents of each location and possibly alter it if required.

You carry out the same procedure with a set of tones and these can be combined to produce a tune. You can also access two tunes in the ROM and this will give you an indication of what can be achieved. Your own tune can be added to the end of the RUNNING WORD DISPLAY and create a wide variety of possibilities.

There are also three games in the EPROM and these can be played in-between the educational programming.

The first game is NIM. Everyone knows this game as 23 matches. The address location for this game is 03E0 and the computer starts with 23 on the display. The object of the game is to try and leave the computer with last match. You can take 1, 2 or 3 matches during your turn. Believe me, it isn't easy.

The second game is LUNA LANDER. Its address is 0490. The numbers on the screen represent velocity and height. You are required to land on the surface of the moon at zero velocity without running out of fuel. The full details of this game are on the last page of this article.

The third game is INVADERS. A number is set up on the left hand end of the display and invaders approach from the right. See the article on how to exterminate them on P 74.
Difficulty is set by the speed control and your score apprears at the end of your turn.

This is only the beginning. In the next issue we will expand the TEC-1 and interface it with the outside world.

PRICES:

Here are the prices for the TEC-1.

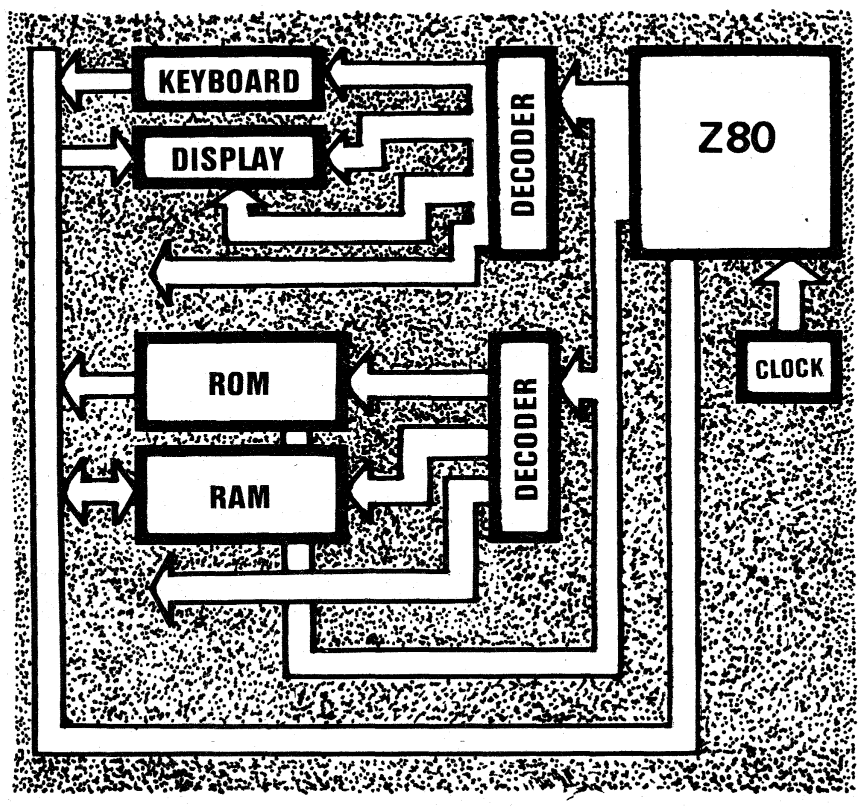
Depending on how much you already have in stock and how you intend to construct the project, so the price will vary. Don't spoil the ship for a ha'penneth o' tar. Use only the best components.

Complete kits are available at our larger outlets and the PC board will be available with pre-programmed EPROM from some of the other outlets. As a back-up service, the complete kit will also be available through the magazine as listed on the kit pages. Remember, the board is double screened and solder masked to give a classy finish to the project. If you have never made your own PC boards before, don't start with this board. The work involved in making a board of this complexity is enormous and the result will be nothing like the cover photo.

The only two outlays you have to accept are the PC board and the pre-programmed EPROM. All the rest can be obtained from your local supplier.

PC Board $19.00 (post $2.50)
2716 - MON - 1 Programmed EPROM **$12.00**(Post $1.50)
Kit of Parts (including EPROM)**$68.30** (Post $2.50
All Parts & PC Board $87.30 (Post $4.50)
You will also need a 6v lantern battery (from your local hardware shop) or a 2155 transformer ($5.90) and a power lead or a 9v AC or DC Plug Pack rated at 500mA ($14.50)

Complete TEC-1 with transformer $93.20 (Post $5.50)
Complete TEC-1 with Plug Pack $101.80 (Post 5.50)



TEC-1 BLOCK DIAGRAM

This simplified BLOCK DIAGRAM shows how each of the chips are inter-connected.

The Z80 Central Processing Unit is the overseer of the whole system and it selects which device it wishes to access via one of the 74LS138 decoder chips. Each will select one-of-eight output lines. These decoder chips are not fully utilized in this project and this leaves room for further expansion.

If we take the key-board as an example, we see it passes its information to the Z80 via the DATA BUS.

This bus consists of 8 lines and carries binary information. This will allow any number from zero to 255 to be sent.

The ADDRESS BUS is a 16 line path which is only a one-way street. Information only emerges from the Z80 on this bus. The Data bus is a two-way street of 8 lines. Information can be passed into the Z80 on this path as well as emerge from it.

Each block in the diagram represents a chip and the only two chips missing are the display drivers.

OUR BORDER

Z80 computer terms have been added to the top and bottom of the pages, for this project. Some of the more common instructions are contained in this string.

Here are the meanings of these terms:

ADD Add the contents of a CPU register to the accumulator.

AND This is a logical AND operation in which two binary numbers are compared. If the first digit in each number is a 1, the answer is a 1. If only one number is 1, the answer is 0. If both numbers are 0, the answer is zero.

BIT This is an instruction to test the status of a bit in a register.

CALL This is a call instruction which will be executed if a particular condition is satisfied. If the condition is not satisfied, then the call instruction is ignored and the program execution continues.

DEC This is an instruction to decrement the value of a CPU register by one.

EX This is an exchange command. The contents of any register can be exchanged with any others.

IN Input to a CPU register from an input port.

INC The increments the value of a CPU register by one.

JP This is a jump instruction.

LD This is a load instruction.

NEG This instruction negates the contents of the accumulator. The result is the same as subtracting the contents from zero.

NOP This is the NO OPERATION instruction.

OR This is a logic instruction which compares two numbers. If either of the first digits is a 1, the answer is a 1. The same applies to the second and third digits. etc.

OUT An output instruction from a specified CPU register.

POP This is an instruction to POP from the stack into a register pair.

PUSH This is an instruction to PUSH an index register onto the stack.

RES This is an instruction to clear the status of a single bit in a CPU register to the logic zero state.

RET This is a conditional return instruction.

RL This rotates the contents of a CPU register to the left through the carry bit.

RST A restart subroutine directive

SBC A double subtraction instruction

SET Sets the status of a single bit in a CPU register to the logic ONE state.

SLA This instruction shifts the contents of a memory location to the left

SRL This shifts the contents of a CPU register to the right

SUB This instruction subtracts the contents of a CPU register from the accumulator

XOR This is a logic instruction which compares each bit of two numbers and gives an answer of 1 if either bit is one. But if both are 1, the answer is zero.

LOOKING INTO THE TEC-1

There are two chips in our computer which provide the major amount of processing.

To make it easy to understand, we will call them the BOSS and WORKER. The boss is the 2716 which is the specially programmed Read Only Memory (ROM) and contains all the information to get the computer started and keep it operating.

The worker is the Z80. It is the arms and legs to which all instructions are sent and it provides the ability (muscles) to carry out the requests of the ROM.

The Z80 can also be thought of as an octopus, extending out its tentacles to all parts of the computer to keep everything in very strict control.

These two chips are the most important items in the computer and it will almost run without any other devices. But you would not be able to push any buttons or see the results of the operations. So we need more chips.

The first of these are the display chips. Because each has only 8 outputs, we need two. The display is multiplexed (see issue 2. P.5.) and this type of design uses the least amount of wiring and the least number of input leads. For a 6 digit display with decimal points, we require 8 inputs for the segment drive and 6 inputs for the digit drive. One 8212 is used for each of these with the digits being driven via driver transistors.

This leaves two spare outputs and one of these is used to drive the speaker via a buffer transistor (Q7).

The Z80 (the microprocessor) is constantly feeding information into the display via the pair of 8212's. When you understand the operation of multiplexing a display you know it is constantly being scanned to create the figures.

To prove this feature, turn the speed control down to minimum and shake the board. You will be able to detect the strobing of the display.

The keyboard is also an interesting feature. It is also being constantly scanned by the 74c923 (the chip near the speaker), waiting for one of the keys to be pressed. The scanning commences at the first row, which is the bottom row and it reads from each of the columns to see if any of the buttons have been pressed. Next it progresses to the second bottom row and again checks the columns. If a button is detected, it sends a debounced signal to the Z80 and interrupts it. The Z80 drops whatever it is doing and accepts a 5 bit binary number from the 74c923, which corresponds to the key being pressed. An example of a 5-bit binary number is 10011 and the following table gives the value for each key on the pad.

KEY	Binary No.
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
A	01010
B	01011
C	01100
D	01101
E	01110
F	01111
+	10000
-	10001
GO	10010
AD	10011

The 2716 ROM tells the Z80 how to interpret the 5-bit binary instruction and what to do with it.

This means we could change the position of all the keys, re-program, the 2716 and the system will be operational again. In other words it is a SOFTWARE programmed set of instructions.

The 74LS138 below the EXPANSION PORT selects between the keyboard and display. Take this example: Button 5 is pressed. The Z80 has all its attention directed to scanning the display via the pair of 8212's. The 74LS138 is allowing the 8212's to function and at the same time prevents an output from the 74c923 to be passed to the Z80.

When button 5 is pressed, the 74c923 sends and interupt signal to the Z80. The Z80 stops scanning the display, requests the 74LS138 to shut down the display and open up the information from the 74c923 from the keyboard. Once the keyboard is read, the Z80 reverts to scanning the display.

This happens so fast that you cannot see the display flicker. The blanking you may see on your model is the result of the time taken to beep the speaker. The longer the beep, the longer the displays are blanked.

The RAM is like a black board. It holds temporary information which is being constantly modified by the microprocessor. When the power is switched off the contents of the 6116 is lost.

The 2716 has a set of instructions which allows the user to access the RAM. Without these instructions you would never be able to get into its memory.

The 74LS138 near the speed control selects between the ROM and the RAM and also any extra memory added to the expansion port.

The 4049 is simply an oscillator or clock which produces clock pulses for the Z80.

Because the Z80 is a dynamic device, its registers need to be constantly cycled to retain their contents. There is a minimum clock rate for this and if the rate is reduced, the computer will crash.

SUMMARY OF EACH CHIP

8212 - Display driver. One chip supplies 8 lines to the segments and the other drives the digits via a driver transistor.

2716 - ROM. The central library of the computer. It tells the computer how to operate.

74c923 - Keyboard control device which interfaces the keyboard with the Z80 chip.

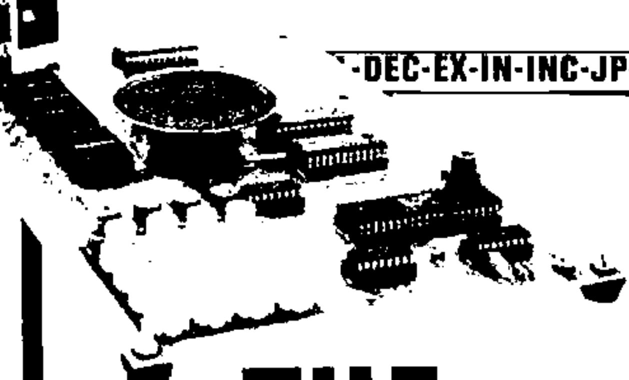
6116 - RAM. Temporary storage for data and instructions. Storage for your own programmes.

74LS138 - Selects the display or keyboard as required by the Z80 chip.

Z80 - The core of the computer. Contains all the necessary logic for executing programmes and manipulating numeric data.

4049 - Wired as an oscillator and fed to the Z80 to determine the operating speed of the system.

74LS138 - Selects between the ROM and RAM as instructed by the Z80.



THE Z80 CPU

The heart of the TEC-1 is a Z80 CPU. This is the largest chip on the board, having 40 pins, and is the central item around which all the other chips operate. The 40 pins are all used to advantage as they are needed to send and receive data as well as send out address locations. On top of this, 8 pins are required for controlling the functions of the Z80.

If we consider the Z80 to be a worker, the BOSS will have to be the 2716 EPROM.

When the computer is first turned on, the Z80 has just enough intelligence to output an address to the EPROM to locate the CPU's first instruction. The EPROM returns this instruction via the DATA BUS and the two start communicating. The EPROM tells the Z80 what to do, how to do it and where it must be put. In less than a second, the start-up procedure has been completed and the whole system comse alive with the START ADDRESS appearing on the display.

For the moment, the Z80 is the chip we wish to investigate.

Most of its pins are ADDRESS and DATA lines. Eight of these are grouped together to become the DATA BUS and 16 are grouped together to become the ADDRESS BUS.

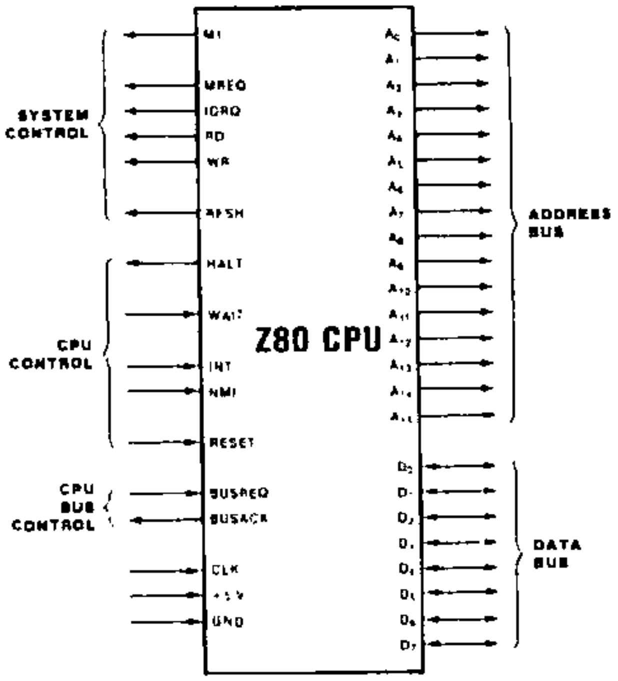
The Z80 uses the ADDRESS BUS to locate the data it wants. This may be in the ROM (the 2716) or in the RAM (6116). It uses the ROM/RAM select chip 74LS138 in this process. Or the information may be from the keyboard. In this case it uses the display/keyboard select chip, another 74LS138.

The Z80 can only do one thing at a time and it is only because the system is operating at between 250kHz and 2MHz (as determined by the spaed control) that you think everything is happening at once.

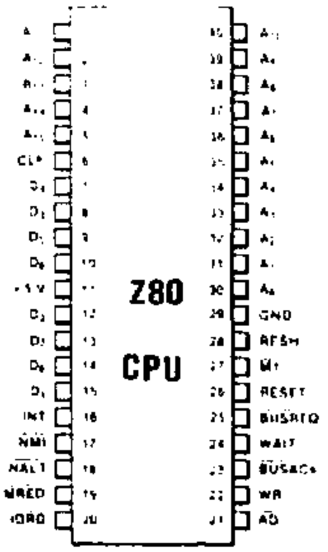
THE Z80 SERIES

The Z80 series (without an 'A' after the 80) was the first to be developed and has a maximum operating speed of 2.5MHz. The Z80A series operates at 4MHz.

There is a whole family of Z80 chips and you must be careful to read the letters which follow the Z80 name, to identify the actual function of the chip.



Z80 LOGIC FUNCTIONS



Z80 PIN OUTS

The Z80 microprocessor is the central element of a microprocessor (computer) and is called CPU. This stands for Central Processing Unit.

Five other chips provide support for the CPU in complex computer systems. We have not used any of these in our simple system but it is handy to know of their existence.

They are: The PIO (Parallel Input/Output). This can be wired to interface peripheral devices such as printers and extra keyboards etc.

The CTC (Counter/Timer Circuit). This chip features 4 programmable 8-bit counter/timers each of which has an 8-bit prescaler. Each channel will operate in either counter or timer mode.

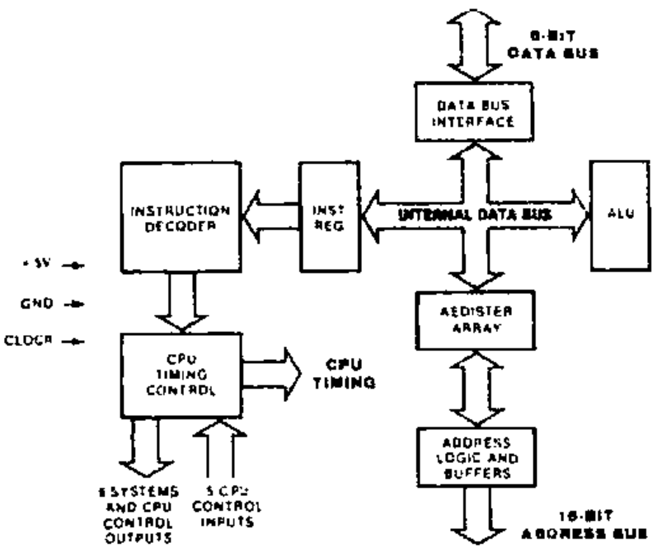
The DMA (Direct Memory Access). This controller provides dual port data operations.

The SIO (Serial Input/Output). This controller offers two channels. It is capable of operating in a variety of programmable modes for both synchronous and asynchronous communication.

The DART (Dual Asynchronous Receiver/Transmitter). This chip provides low-cost asynchronous serial communication. It has two channels and a full modem control interface.

The Z80 series of chips have completely different operations from each other and the letters on the chip are VERY important.

It's difficult to realize, but the Z80 is classified as a "dumb worker". It may be dumb but it is very quick. It is capable of carrying out instructions at the rate of about 10,000 to 200,000 operations per second, depending on the type of instruction. Each of these takes a particular number of cycles to execute and these features are contained inside the Z80 architecture and cannot be altered.



Z80 CPU BLOCK DIAGRAM

Each operation for the Z80 has a machine code instruction such as 1E, 06, 0E, 85, E6 dd, 06, E9, 8, 81, ED 47, 00 etc and these will be discussed in a later article. For the moment, we want the computer to seem a reality.

There are lots and lots of sections inside the Z80 chip and most of them are very difficult to explain in simple terms.

One area which can be explained via a simple comparison is the bank of registers. These perform most of the operations in the Z80.

They are given the names B, C, D, E, H and L, with an accumulator register A. These registers can ba likened to a car space in a parking lot. Each register represents one car space. The car represents one WORD of information and this work consists of 8 bits or one BYTE. A bit is a signal on a single line which can be either HIGH or LOW and 8 lines entar the Z80 in the form of a DATA BUS.

This data bus is the same as the road into a parking lot and the car is one word. The 8 bits are 8 seats and aach car can have up to 8 people. Depending on where they sit and tha number of people, the size of tha byte is determinad. We can say that byte and word are the sama for our system as the Z80 is an 8-bit microprocessor. If it were a 16-bit microprocessor, a word would be 16 bits.

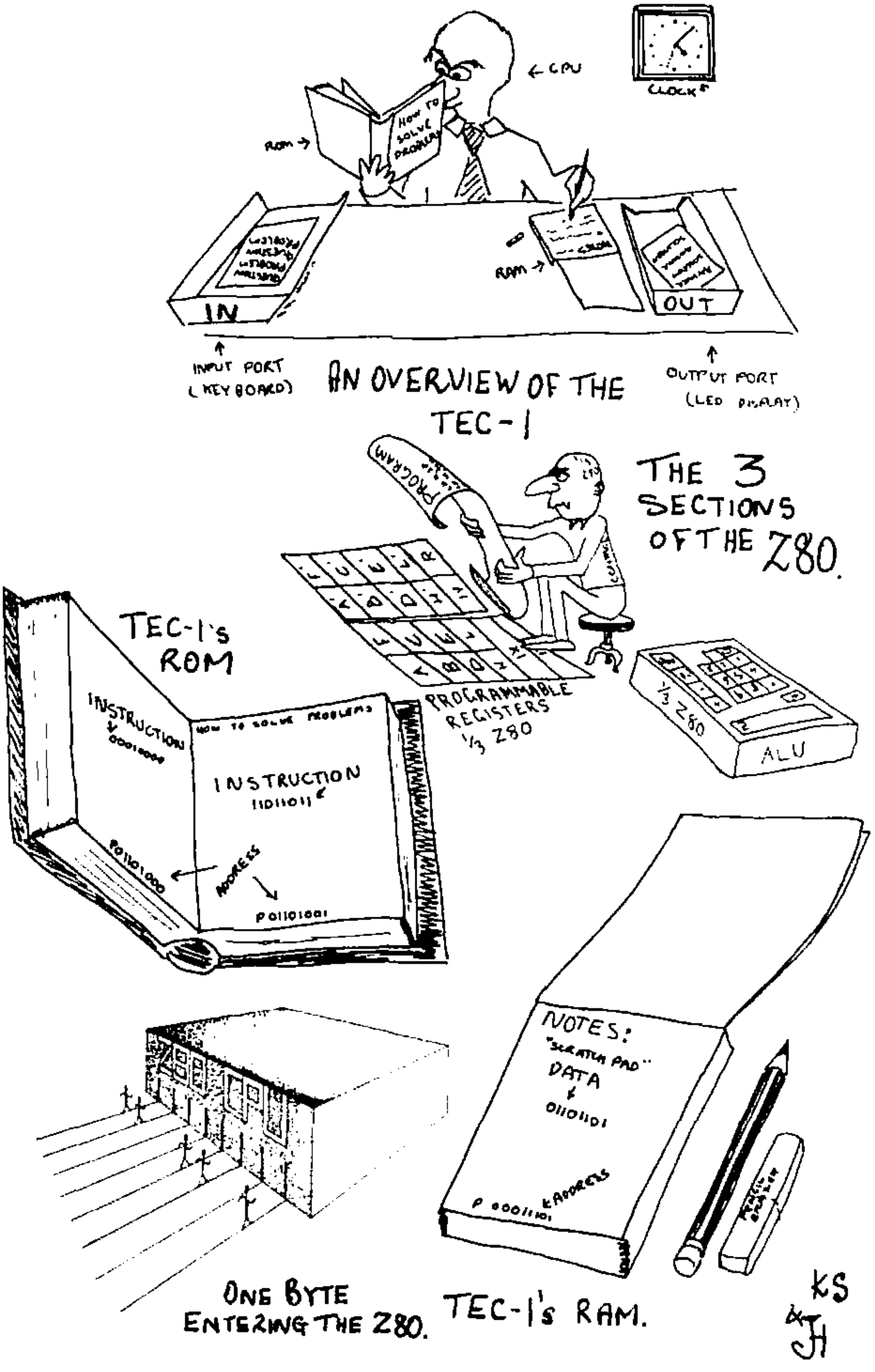
Normally the car parks in space A (register A) and this is also callad the accumulator register as the answer for any addition instruction, for instance, will appear in register A.

Tha car can represant a number from 00 to 255 and the ragister will accept any of these numbers. This is all the register will hold. . .just one number from zero to 255.

There is one important fact that we have omitted to mention. Before tha byte can be put into ragister A we must send an instruction to the Z80 so that it will know where tha number is to ba put.

This instruction happens to be 3E for register A. If you wanted to load register B, the operation code would be 06 and to load register C it would be 0E. These codes are called "MACHINE CODES" or MACHINE CODE LANGUAGE and they are intarpreted by tha Z80 to perform one of over 245 different shuffling or arithmetic operations.

If you want to add a number to the number above, it will have to be firstly loaded into register B, then the two registers can be added. This will teke a number of operations with the result always appearing in register A.



You can transfar the contents of ragister A to tha RAM (6116) via a furthar instruction so thet the result is not lost when the next number is sant to registar A. Otherwise the previous contents of register A are writtan over.

The Z80 contains an equivalent bank of emergency registers which can be accessed via a spacial instruction. Thesa are called A'(A-prime) B', C', D', E', F', H', and L'. It also contains a numbar of 16-bit registers (like a spaca for car and trailer) and numerous building blocks which are needed to kaap tha Z80 operating. These can be likened to the workers needed to keep a parking lot neat and with a smooth flow of traffic.

CONSTRUCTING THE TEC-1

Constructing the TEC-1 is no more complex than building any of the cover-projects in Talking Electronics . . it only takes longer.

The most important aspect of this project is NEATNESS. We have gone to a lot of trouble to create a printed circuit board that looks really neat, with a layout that is very pleasing. Don't upset the aesthetics of the board with poor-quality layout or incorrect components. If you intend to buy the components individually at your local electronics store, look at the photos in this article for the type of components we have used, and purchase the same styles.

Don't use anything old or dirty and make sure the tinned copper wire for the jumpers is CLEAN, thick and absolutely straight. We will tell you how to do this in the notes.

The TEC-1 is not designed to be fitted into a case. It is too beautiful to hide. Like all our projects, it is designed to be viewed. This keeps you alert to the construction, contents and arrangement of the chips and components. You must constantly remind yourself of the name of each chip and its function. It's an indoctrination process which can only be of benefit in the long term.

Before commencing construction we suggest you get everything organised on the workbench.

We can't stress strongly enough, the need for a good soldering iron.

We have a range of soldering irons in our assembly area including: a 10watt, 12v pencil iron, a 15watt 240v Micron, a 60watt Constant Temperature Scope iron and a Weller Soldering Station. We also have a plummer's soldering iron and two instant-heat solderings as well as a miniature instant-heat iron. Which one would you choose?

If you chose an instant-heat iron, we don't want to know you. They will lift the lands off the board and create more problems than you can imagine. Also construction time will be considerably longer as you have to wait for them to heat up for every connection. Our choice is the 10watt 12v type. It it light-weight, and enables you to produce a speedy connection. This is important when constructing a large project like this.

Other important tools and aids are: fine solder, sharp side cutters, and a pair of long-nosed pliers. You will also need a soldering-iron stand and a solder tray to accept the dead solder left on the iron after making each joint.

Get everything ready on a clean part of the workbench and have all the components available for insertion.

The first part of construction is the most laborious. It is the fitting of the 55 links. You must take great care when fitting these links as they must be absolutely straight, with their ends bent to a sharp 90°. The whole link must touch the board.

Start at one end of the board. Cut a length of copper wire about 10cm long, which will be sufficient for about 5 links. With a pair of pliers at each end of the length of wire, pull the two pliers apart until the bends and kinks are removed and the wire is perfectly straight. Now you can use the wire. Bend one end with the pliers and insert it into one hole in the board. Solder this end and snip the excess wire from the joint.

Feed the other end down an appropriate hole and pull it through with the pliers until the link becomes straight. Keep this part pressed against the board while soldering it. Cut the surplus from the connnection and inspect the first addition to the board. Continue with each link as you come to it and take your time. It will take the best part of an hour and no link should be loose enough to touch any other, even if it is pushed slightly.

The next components to add to the board are the resistors. There are 15 of these and they should also touch the board. Check the value of each resistor before inserting it. They are hard to remove if you make a mistake.

Next are the IC sockets. The reason for inserting them at this stage will be quite obvious. As each socket is inserted, it becomes the highest component on the board and this means the board can be turned over and the socket will rest on the workbench while the pins are being soldered.

You almost cannot make a mistake with these sockets as the number of pins corresponds to the holes in the board. The only point to remember is the identification notch at one end of the socket. These should cover the dot so that when the chips are inserted, the notch on the chip aligns with the dot on the board.

Next add the 6 FND 500 displays. Once again the board can be turned over and rested on the display to keep it pressed against the display while the pins are being soldered.

The next order of insertion is not critical and would consist of inserting the power diodes, 2 LEDs, 7 100n

cont. P. 68

PARTS LIST

1	-	100R
1	-	330R
8	-	1k
1	-	2k2
5	-	10k
1	-	20k cermet
1	-	100pf
7	-	100n 100v
1	-	1mf 16v
1	-	2200mfd 25v
4	-	1N 4002 diodes
7	-	BC 547 transistors
1	-	5mm red LED
1	-	5mm green LED
6	-	FND 500 or 560 displays
1	-	7805 regulator
2	-	8212
1	-	2716 TEC-1 Monitor
1	-	6116
1	-	74c923
2	-	74LS138
1	-	Z80 CPU
1	-	4049 NOT Fairchild)
3	-	16 pin IC sockets
1	-	20 pin IC socket
4	-	24 pin IC sockets
1	-	40 pin IC socket
21	-	PC mount push switches
1	-	8R speaker
1	-	heat fin for 7805
4	-	rubber feet
5	-	nuts and bolts
60cm tinned copper wire		
3m fine solder		
10cm desolder wick		

Substitutes:
2200mfd electrolytic can be replaced by 1000mfd in the TEC-1.
Rubber feet can be stick on feet.

HOW THE CIRCUIT WORKS

The TEC-1 circuit looks very simple and, in fact, it is very simple.

This is because many of the chips in a computer circuit are connected to a parallel wiring system called a BUS. There are 2 main buses in a computer and they are called ADDRESS and DATA.

Normally the ADDRESS bus is 16 lines wide but our computer is only a baby design. We have used 11 lines in the address bus with line 12, 13 and 14 going to a decoder chip to select between display, keyboard, memory and expansion.

The data bus is like a highway with data passing to and from the Z80 and the other chips. The data line will carry a binary number between 00000000 and 11111111, which is 0 - 255.

The 8212's are latches which drive the displays. One controls the segments a to g and the decimal points while the other drives the digits via a set of buffer transistors.

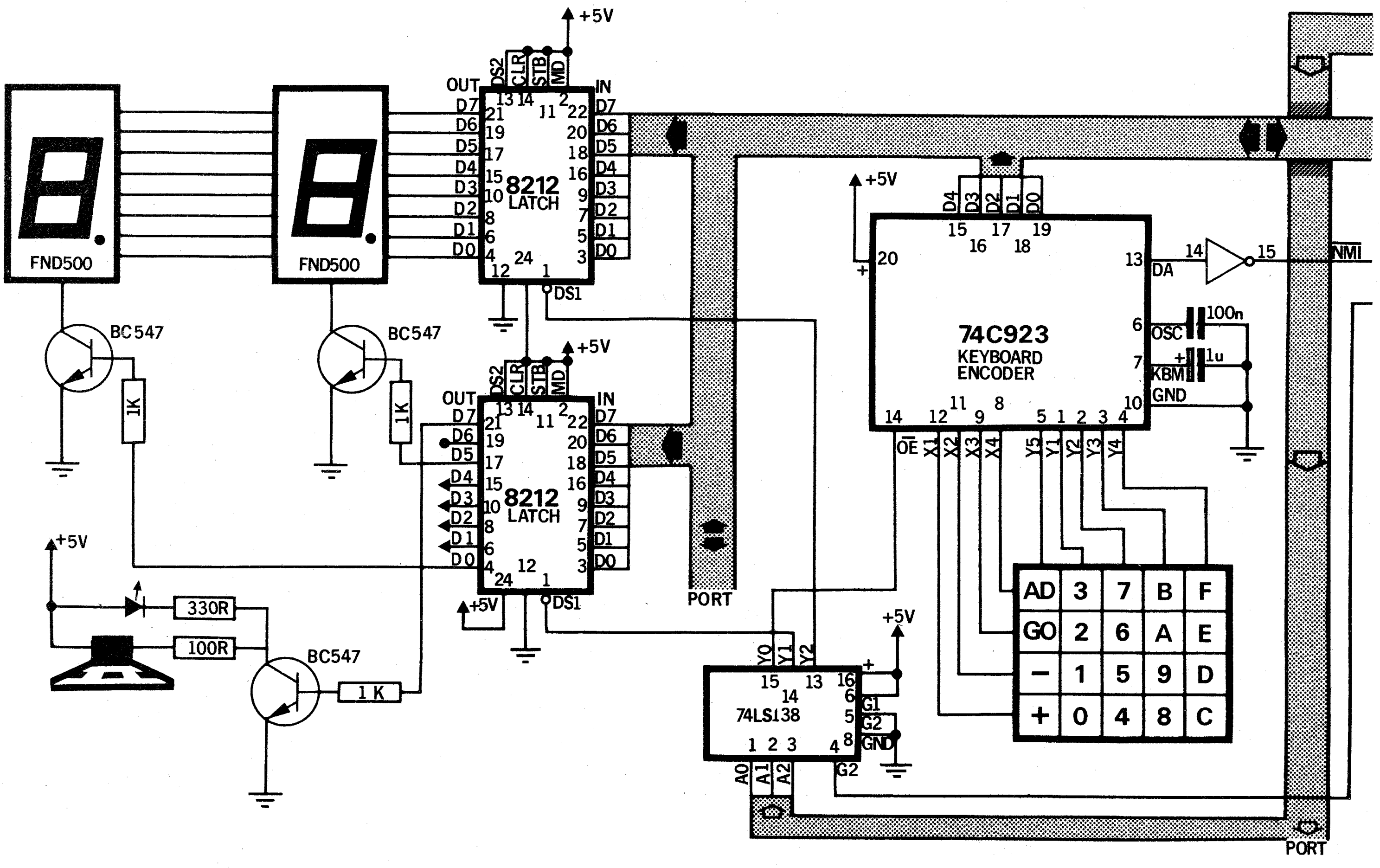
Data and program are stored in the memory whch comprises the 2716 EPROM and 6116 RAM.

The Z80 addresses a particular location in the memory by sending a binary number down the address bus. It determines the condition of SENDING or RECEIVING by the state of the R/W line (pin 22). When this line is LOW, the Z80 is sending data to the RAM and when HIGH, it is receiving data from the memory.

Data is sent or received via the data bus and only one data transfer can occur at a time.

When the reset button is pressed, the computer sets the condition for initial data entry. These include entering (or loading) the address pointer to 0800, setting the dots on the data displays, setting the stack to the highest point in the 6116 RAM and calling a routine to produce the two-tone 'ready' beep.

The Z80 then goes into a scan routine to display 0800. This information is

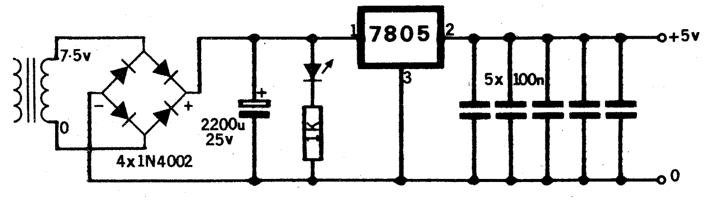


POWER SUPPLY

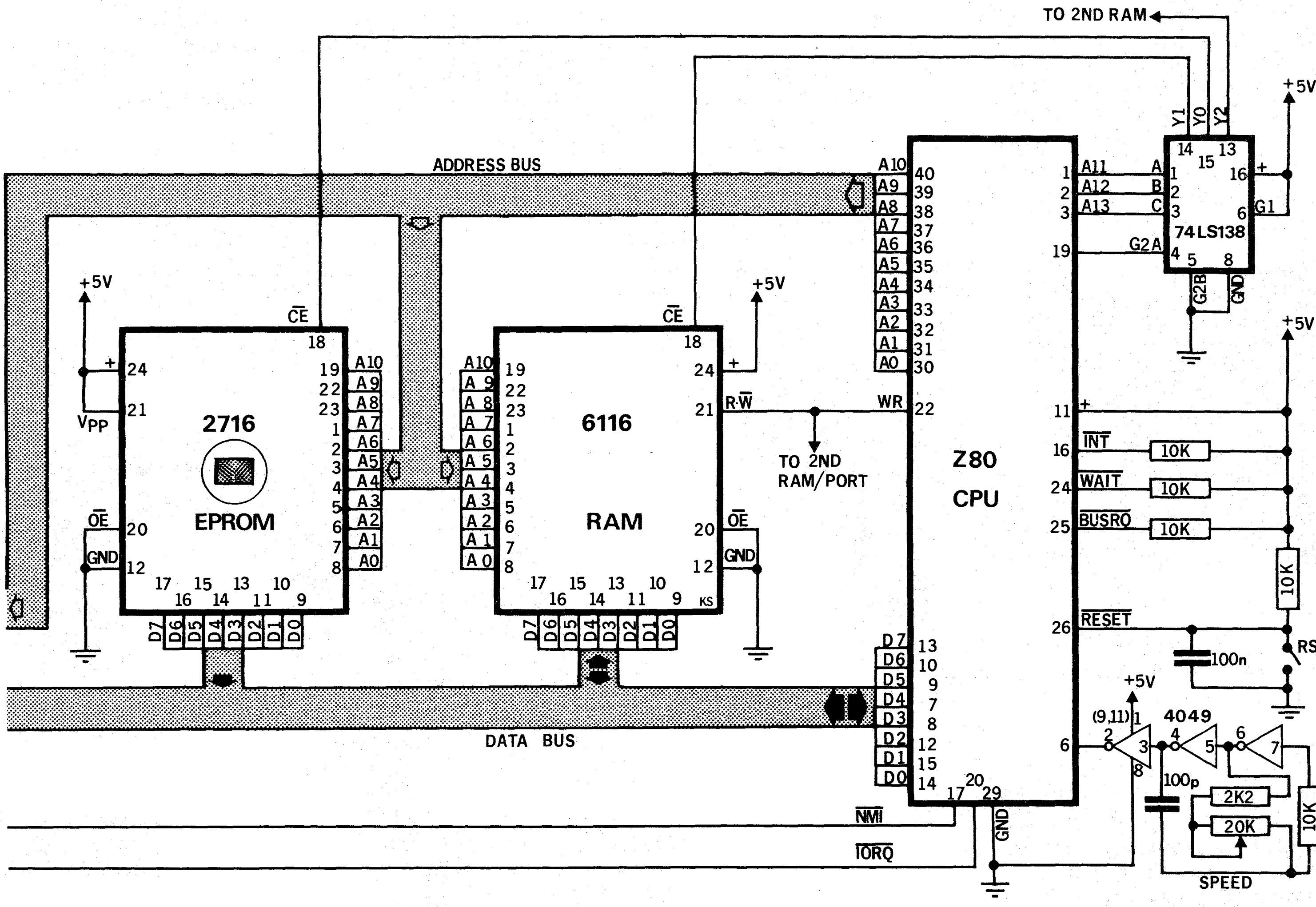
The 7805 regulator is included on the PC board. When using a 2155 transformer to power the TEC-1, use the 7.5v tapping. This will produce about 9.5v DC into the regulator which is ideal to gain full voltage and current from the power supply without overheating the regulator.

The TEC-1 will accommodate a DC Plug Pack. Use a 9v type capable of delivering 500mA.

The five 100n capacitors on the output line are spike suppression capacitors. They are placed near each of the chips to prevent noise from one chip upsetting the funtion of the computer. We suggest low-impedance mono-block types for this application.



POWER SUPPLY



taken from the highest bytes in the RAM, which have been deposited in the set-up routine.

The Z80 will continue to scan the displays until it is interrupted by the 74c923, via the inverter of the 4049 chip. This is a Non Maskable Interrupt line. The Z80 immediately branches to a routine at 0066H which inputsthe binary code of the key being pressed and stores it in a register in the Z80.

It firstly checks if the key is a function key or a numeric key 0 - F. It does this by checking bit '4' of the 5-bit binary number. Bit 4 is the 1 in: 10000. The first bit is called bit 'zero'.

If the display is in the address mode, the function key will simply put it into the data mode. This is indicated by the dots moving to the data displays.

If the computer is in the data mode, a function key will perform the function intended. A + will increment the address pointer , a — will decrease the address pointer AD will set is to the address mode and GO will execute the program starting at the address shown in the display.

If a numeric key is pressed, the data displays are cleared and the digit is shiftedin from the right. A second key will produce a 2-digit number.

When the + key is pressed when the displays are in the data mode, the address will increment.

While this seems a simple operation, an enormous amount of data is flowing from the Z80/ROM/RAM combination. The address pointer, which is temporarily stored in RAM, is loaded into the Z80 HL register (two 8-bit registers connected to become a 16 bit register). This register increments by instruction 23 (inc HL) and HL is then stored back into the same location in RAM.

The display is then changed to reflect its new address and contents of RAM (in data displays). The Z80 then reverts to its scan routine waiting for another key to be pressed. All this happens in a few milliseconds!

The 74LS138's are simple DEVICE SELECTING chips. The have 3 binary input lines and this enables them to select any one of 8 devices. These can be ROM, RAM, keyboard, display, speaker, or external chips, even additional memory or video displays.

The speaker is simply an amplified version of a 'BIT'. A BIT is a HIGH or LOW state and constant rapid changing from HIGH to LOW will produce a tone.

The quality of the sound can be altered by the ratio of the HIGH to the LOW. White noise is a result of random bit production. Correct programming enables the production of music and sound effects.

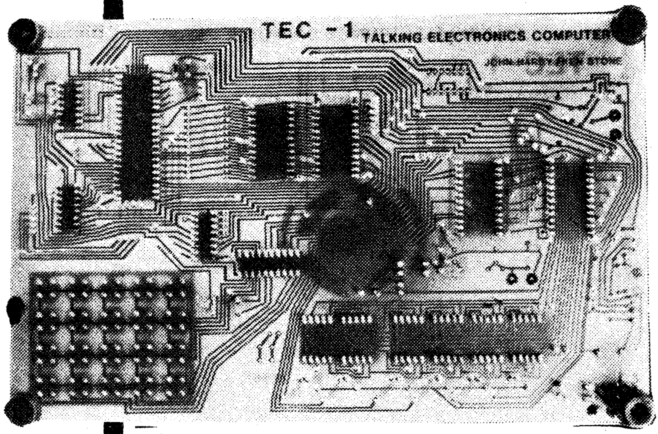
The power supply is a simple 7805 voltage regulator arrangement. Provided the input voltage is only about 3v above the output voltage, the regulator will not need a large heat-fin. The 2200mfd electrolytic can be replaced with a 1000mfd electrolytic as the computer consumes only about 500mA.

The speed of the TEC-1 is controlled by the 4049 clock oscillator. This is only a simple 2-inverter oscillator which can be adjusted via a speed control to vary the speed of the information passing the displays. A crystal controlled clock can be added at a later stage.

capacitors, 7 transistors, 1- 100pf capacitor, 20k cermet pot and 1 - 1mfd electrolytic.

Attach the flag heat-sink to the 7805 and insert the regulator into the holes nearest the 2200mfd electrolytic.

The other 7805 powers the expansion board and will be covered at a later stage. Insert the 2200mfd electrolytic and solder the leads.



The underside of the TEC-1 showing the layout of the copper tracks.

The keyboard switches are individually inserted and soldered as shown on the overlay. The flat on the switch runs across the bottom of the switch so that the jumper link inside the switch completes the wiring of the matrix.

Attach the speaker to the board via a piece of double-sided sticky tape and connect the voice coil to the circuit via short lengths of tinned copper wire.

Four rubber feet are attached to the board with nuts and bolts to prevent the underside of the board from scuffing the workbench.

The final, and most important items to add, are the IC's. These are pushed into the sockets so that pin 1 on each chip is facing towards the display. The 74c923 faces towards the left and you double check each chip before AND after it is inserted. If the rows of pins are too wide, they can be pressed closer by pressing the edge of the chip on the PC board and then the pins will be easier to insert into the socket.

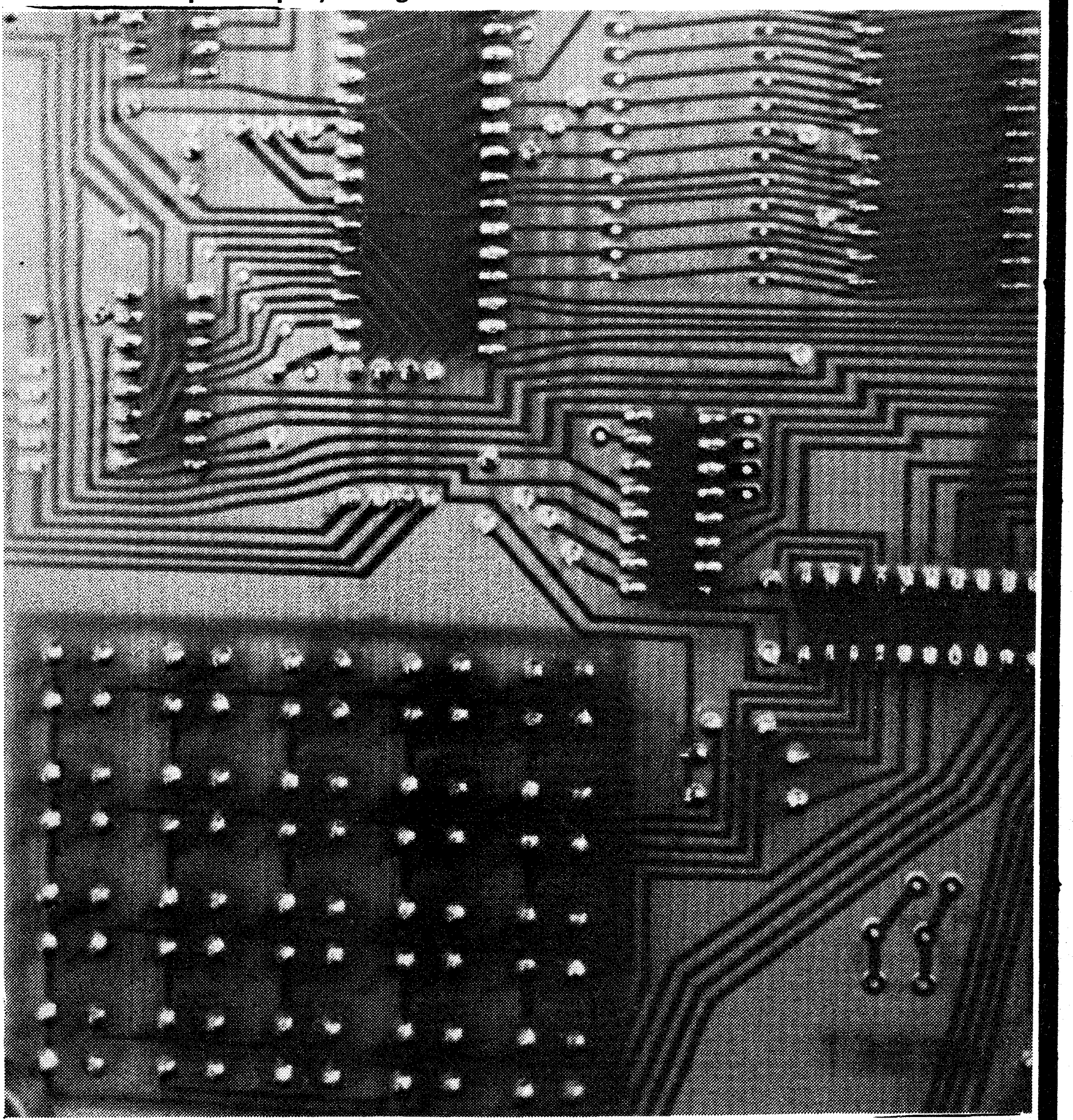
Connect an AC supply to the board and the TEC-1 is ready for operation.

You can use either a 2155 transformer or a 9v plug pack capable of delivering 500mA. In either case the incoming voltage should not be more than 8v to 9v to prevent the regulator getting too hot.

Switch on the power and note the display lights up with 0800. This is the first available address and indicates the computer is ready for action.

If you are well-versed in Machine Code language, you can begin immediately with preparing your own programmes. You will find the TEC-1 is very versatile in its applications and will allow a wide variety of expansions to be accommodated.

Treat the computer as a basis for experimenting and learning. Later we will provide add-ons for a crystal oscillator, output displays for games, and control devices for up to 8 different items at the same time.



An enlargement of the Key-board section showing the soldering.

We will also welcome any programmes you write for the computer and it doesn't matter what subject they are witten about.

The tape interface to be added in the next article will allow you to save programmes and re-use them later.

If you are new to programming, you will appreciate the introduction presented on P. 71. It starts at the beginning and shows you how to key a short program and activate a readout in the form of a visual display as well as a musical score.

Three games on P. 74 will intrigue everyone. The level of skill can be adjusted by turning the speed control. This increases the rate of operation of the whole computer.

The are also other programmes in the EPROM and these will be discussed in the next article.

If you are having trouble getting the TEC-1 to operate, see the article on P. 70. It will solve most of the simple construction faults.

We all wish you the best with your new acquisition.

Don't under-estimate the capabilities of the TEC-1. It is a very powerful machine.

IF THE TEC-1 DOESN'T WORK:

If you are faced with the situation where the TEC-1 fails to operate properly, or if it doesn't work at all. . . don't worry. This will be a blessing in disguise.

You learn a lot more about electronics and computers by fixing the TEC-1, than just building and running it.

As requested in the introduction to this project, you should already have a certain amount of background in building projects. This is when all these skills will come together.

The first point to remember with the TEC-1 is this: The TEC-1 SHOULD operate perfectly the first time it is turned on. This is because it is built with NEW components which are first-quality items and the PC board has been thoroughly checked. If you are unfortunate enogh to produce a dud, you must firstly realise that there is a 99% possibility that the fault is in the construction.

You should go over the entire project again, checking every component, connection and the value of each part. The best way to do this is to ask someone ELSE to do the checking. This is because you cannot check your own work. Most of the projects that come to us for repair are simple faults, overlooked by the constructor. Faults like 1k instead of 1M, 22k instead of 3k3 etc. This type of fault can very easily creep in. This is because humans think positively. Most constructors are CERTAIN all the values are correct! How could they make a simple mistake like THAT?

After passing the TEC-1 over to a government checker (anyone impartial) you can begin the TEST procedure. This will need test equipment.

This is where the LOGIC PROBE will come in handy. That's why we presented it in this issue. You will find it invaluable, as most of the lines on a computer are PULSE lines and these are constantly changing accoding to the clock rate or as requested by the Z80.

The first test is a RESISTANCE TEST.

To carry this out successfully, you should remove all the chips. This is to prevent any false readings.

We will be looking for solder bridges between one or more of the pins. These can be very difficult to see as they are sometimes as fine as a human hair or even merely a microscopic splash of solder.

That's why you must never tap the soldering iron on or near the board, as excess solder will fly off the tip and land on some unknown part of the board. This will cause a bridge which will take hours to locate. You must only tap the iron in a solder tray and this must be done after every joint to prevent dropping solder and creating a problem, like now.

Set the multimeter to LOW OHMS RANGE. Make sure you adjust the ohms control so that the needle travels to the far right hand end of the scale to indicate very LOW resistances.

When all the chips are removed, most of the wiring on the underside of the board consists of individual conductors and this means almost no adjoining pins are connected. This makes it ideal for testing via a resistance measurement.

The first place to check is the RAM/ROM section where each of the pins has a conductor running between them. Turn the board over and measure the resistance between each solder connection. The multimeter needle should not move at all. If all the readings are HIGH. progress to the Z80, and then each of the other chips. If the pointer deflects at any stage, trace through the wiring to see if a resistor or push button is in the path. You will also get some low readings when testing near the display. So don't treat these as faults.

USE THE LOGIC PROBE AS DESCRIBED IN THE FIRST PROJECT, TO TEST THE TEC-1.

Another very important check you can make is the continuity of all the printed wiring on the underside of the board. Sometimes one of these tracks can become eaten away in the etching process, resulting in a break.

Place one of the probes on one end of a conductor and visually trace it through to the end. Place the other probe at this point and prove that it is conducting. You can also make sure the jumpers are connecting by checking the ends of each run.

If all these checks fail to locate any problem, you will have to carry out tests with the computer operating. This will mean replacing the chips and connecting the power.

Start by placing the probe on pin 6 of the Z80. This is the clock input pin and without any signal here, the whole computer will not operate. The three LEDs on our LOGIC PROBE will illuminate and you will hear a frequency from the mini speaker in the probe. As you adjust the speed control, the sound will change pitch, If the 3 LEDs don't flash, change the 4049. Some chips are very critical in this circuit and others don't work at all. If a chip fails to oscillate, you can reduce the 10k to 2k2 and this will give you a broader range. Some chips may tend to drop out at the low end. You should buy a CD 4049 as soon as possible.

Once you have a clock pulse entering the Z80, you can check some of the other sections of the computer.

The computer can be placed in a WAIT situation by tying pin 24 to earth. This pin is connected to the 10k which is the closest to the reset switch. It has an empty hole to which you can solder a test wire and use a jumper lead to create the wait situation. Press RESET and probe pin 15. If it is LOW, everything is OK. If it is not LOW, you may have a dry joint or short-circuit on pins 1 - 6 of the 74LS138. While the computer is in the WAIT condition, test the operation of the key-board by probing pin 15 of the 4049. This is the output of the 74C923 key-board encoder, after it has passed through an inverter to the non-maskable interrupt of the Z80.

This output line is normally HIGH and goes LOW when a key is pressed. The only other pin of the 74C923 which can be checked in a simple manner is pin 7. It is normally LOW and goes HIGH for the duration a key is pressed.

The 74C138 select chip below the expansion port selects between the key-board and the two 8212 driver chips. When in the wait mode, pins 13, 14 and 15 are HIGH. Under running conditions, pin 15 pulses LOW when a key is pressed.

The two display driver chips, (8212) are difficult to test under static conditions as the display is multiplexed.

When in the wait mode, one of the displays may illuminate and you will be able to detect the HIGH's entering the 8212's.

The advantage of the sockets becomes apparent when you have to remove or change any of the chips.

If the computer still fails to operate correctly, try replacing the set of chips. This will only be feasible if you know someone with a TEC-1. Within your own board you can exchange the two 8212's and 74LS138's. Don't forget, the 2716 must be programmed. A blank one will not get the computer started.

Make sure no pins are bent under the sockets or broken off at the chip. Be sure the chips are around the correct way and most of all, make sure the chips are in the correct positions.

If all this fails, write to us. We have a repair service available. If sending the project by post, pack the board between two thick pieces of foam or stiff card-board. Use a large jiffy bag and mark it fragile. Certify the parcel in case anything is damaged. This way it will get to us in one piece. We will have a look at your project and let you know what it will cost to fix. Usually it doesn't cost much and this will take a load of your mind.

I hope it never gets to this stage, but at least you know the service is available.

EXPERIMENTS FOR THE TEC-1

The computer should now be fully assembled and ready to go. All you have to do is learn how to operate it.

The following set of experiments will give you the experience necessary to recall some of its routines and produce a simple sequence of your own.

To introduce you to the TEC-1 we have programmed a welcome message into the EPROM. This can be located at 02D1.

To call up this program, press the following sequence of keys:

RESET, D, 1, + 0, 2, ADdress, 0, 2, 7, 0, GO, GO.

Adjust the speed control and see what John has written. If that's not a clever way of personalizing a piece of equipment!!

If you don't know what to do, don't worry. Follow through these experiments and come back to the WELCOME mesage later.

Now to the learning section:

Experiment 1.

AIM: To examine the increment of the address.

Apparatus:TEC-1.

Procedure: Turn the TEC-1 on and look at the address display. The address is the first four digits. When the TEC-1 is turned on, the first available address is 0800. This can be incremented by pressing the '+' key and the address will increase to the next available location. Carry out this procedure by pressing the '+' key and watch the display: 01, 02, 03, 04, 05, 06, 07, 08, 09. The next location is 0A and this is where the computer departs from the reading you would expect. The TEC-1 is programmed in Hexadecimal in which 4 binary lines are grouped together to form a hex number. In this way we can write binary numbers from 0000 to 1111 and this means we can go higher than 9 as nine is only 1001 in binary. The next hex number is A, then B, C, D, E, and finally F. The following table shows the binary equivalent for 0 - F.

Decimal:	Hex:	Binary:
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

A comparison between decimal numbers (based on the power 10) hexadecimal numbers (based on the power 16) and binary numbers (based on the power 2). The data for TEC-1 is entered in hex on the key-board. Examples of hex are: 3F, 4C, 5B, FE, C4, DD,
The max hex for 2 digits is FF and this corresponds to 255.

Press the + and watch the display increment,
Press the − key and watch the display decrement.

Experiment 2
STUDYING HEX

Aim: To study hex notation and count in Hex.

Equipment: TEC-1.

Theory: Each byte of data for a program must be given an address. The computer automatically advances one address location on pressing the + key. However we must be able to read and write hex values to be able to prepare a program.

Procedure: Study the Hex notation in expt 1. and answer the following set of problems:
Use the TEC-1 to verify your answers.
Problem 1: A program starts at 0800. What are the next 21 addresses?

0801 ★ ★ 0804 ★ ★ ★
0808 ★ 080A ★ ★ ★ ★
★ ★ 0811 ★ ★ ★ 0815.
To verify your answer, press RESET then keep pressing + + + + + etc. Don't worry about the values in the data displays.

Problem 2: A program starts at 0A00. Complete the following set of addresses:
0A00 ★ ★ ★ ★ ★ 0A06
★ ★ ★ ★ 0A0B ★ ★ ★
★ ★ ★ 0A12.
To locate address 0A00: Press RESET, press AD (the dots will appear on the address displays indicating the address can be changed).

Press 0, A, 0, 0. Press +. This becomes the first address location. Press + + + + + etc to increment the display.

Problem 3: A program of 50 addresses finishes at 091E. What are the previous 35 addresses?
091E ★ 091C ★ ★ ★ ★
★ ★ etc to 08Fb.

Problem 4: (a) Add 4 address locations to 0209.
(b) Add 8 address locations to 1FFF.
(c) Add 4 address locations to 0BFD.
(d) Decrement the address 7 locations from 0800.

Work out all the above answers on paper before checking with the TEC-1

ANS: 4(a) 020D, (b) 2007 (c) 0C01 (d) 07F9

Experiment 3:

CREATING A BEEP

AIM: To create a tone or beep on the TEC-1.

Theory: The 2716 has been pre-programmed with a loop to give a pulse to the speaker. Depending on the speed of the system, the tone of the pulse will be varied.

Procedure: We can address the beginning of this routine by pressing the following keys:
RESET, ADdress, 0, 1, 8, E, GO, GO.

You will hear two beeps, and by turning the speed control down, they will become separated. The first beep is the one you have programmed. In the next experiment you will will change the frequency of the beep.

Notes:
When the TEC-1 is reset, the decimal points appear in the DATA readouts. This indicates the data can be changed by pressing the keys 0 - F.

By pressing the ADdress key, the dots will appear in the ADDRESS readouts. This can now be changes by pressing the keys 0 - F.

Experiment 4:

Creating a Tone or Note.

AIM: To create a tone.

Theory: It is possible to produce a tone from the speaker which is the result of a routine in the EPROM.

Procedure: To create a single note or tone, press the following:

RESET, 2 + 8 + **1** *ADdress, 1, B, 0, GO, GO.*

Only the first note to be heard in the speaker is the product of our programming. The other beep or beeps come from other routines.

The number **1** in the routine above selects the particular note. It can have one of 24 different values and thus we can create different effects as shown in a later experiment.

AIM: To create a single note with a period of silence.

The instruction for silence is 00.

Run the following program:

2 + 8 + 0,3 + 00 + 00 + 00 + 00 + 0,3 + 00 + 00 ADdress 1, B, 0, GO, GO.

You will notice that pressing 00 is the same as pressing 0. The DATA entry is self-adjusting. Thus 03 is the same as 3 on the data display.
Turn the TEC-1 off between experiments so that the 6116 RAM has its contents destroyed. Otherwise some of the previous programs will come through the speaker.

To create a scale:

Program this sequence:

2 + 8 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + A + B + C + D + E + F + 10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 0 + 0 + 0 + 0 Address 1, B, 0, GO, GO.

The speaker will produce the scale, then silence.
To hear the sequence again: Press RESET, ADdress, 1, B, 0, GO, GO.

AIM: To produce a repeat function.

The note, notes or sequence can be repeated by adding the instruction 1E to the end of the list.

Try this routine:

2 + 8 + 1 + 2 + 3 + 4 + 5 + 6 + 5 + 4 + 3 + 2 + 1 + 1,E Address 1, B, 0, Go, Go.

To produce a repeat function with a pause or silence, try this routine:

2 + 8 + 1 + 2 + 3 + 4 + 5 + 0 + 0 + 0 + 0 + 5 + 4 + 3 + 2 + 1 + 1,E Address 1, B, 0, GO, GO.

Experiment 5:

To Create A Tune.

By using the note table on P.73, any tune or melody can be produced. Try this sequence, then write your own tune.
2 + 8 + 0A + 08 + 06 + 08 + 0A + 0F + 0A + 0D + 0F + 06 + 06 + 0A + 0D + 06 + 0D + 0A + 0D + 12 + 16 + 14 + 12 + Of + 11 + 12 + 0F + 0D + 0D + 0D + 0A + 12 + 0F + 0D + 0A + 08 + 06 + 08 + 0A + 06 + 06 + 00 + 1,E, Address 1, B, 0, GO, GO.

Question: How do you recall this sequence?

SOUNDS AND TUNES

The TEC-1 has a number of musical routines programmed into the 2716 EPROM.

These are accessible via the keyboard. The first of these is an Irish Jig. This is called by the sequence:

RESET, E, F, GO,

With all tunes the pitch of the notes is dependent upon the speed of the computer. This is determined by the speed control.

You can experiment with adjusting the 20k cermet for each of these tunes, to get different effects.

In order to access some of the other tunes, you will need to follow this key sequence:

RESET 3 0 + 5 address 1 B 0 GO, GO.

Experiment 6:
Creating a Running Letter

AIM: To produce a running A.

Procedure: Press the following sequence of keys:

RESET, 2 + 8 + 1 + 0 + 0 + 0 + 1,E Address 2, 7, 0, GO, GO.

The letter A is being shifted one place to the left by the routine at 0270.

The letter can be made to travel the full length of the display by adding further zeros to the program.

Press this sequence of keys:

RESET 2 + 8 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 1,E address 2, 7, 0, GO,GO.

To produce a running sentence:

Press this sequence of keys:

RESET 2 + 8 + 07 + 0E + 0E + 04 + 0 + 9 + 04 + 05 + 01 + 1A + 0 + 0 + 0 + 0 + 0 + 1,E address 2, 7, 0, GO, GO.

Experiment 7:
Combining Words With A Tune

AIM: To combine a tune and running words in one sequence.

Procedure: The programme we will be writing in this experiment consists of a set of instructions and included in this are two CALL statements (call 1B0 and call 270) followed by a PITCH table and a LETTER table.

The result of your programming will be a short tune followed by three letters running across the screen and this will be repeated.

We will firstly describe the program for experiment 7 in **WORDS.** Refer to the program below to see what we are talking about.

The program starts at 0800 and in the first two bytes (800 will accept a byte of data and 801 will accept a byte of data) we will store the address of the pitch table (which starts at 0900) . Later the computer will store the letter table and this will be repeated over and over again as the program contains a jump or repeat instruction.

First of all we will discuss each instruction at each address so that you will be able to understand the program you will be keying into the TEC-1.

At address 802 the instruction 3E tells the accumulator to load the immediate byte, which is 00.
This takes up address 802 and 803.

At address 804 the instruction 32 tells the computer to load the contents of the accumulator into the address given by the following two bytes. The lowest byte is always presented first, then the highest-order byte. Thus 00 is loaded first then 08.

The next available address is 807.

The instruction 3E tells the accumulator to load with the immediate byte which is 09.

At address 809 the instruction 32 tells the computer to load the contents of the accumulator into the address given by the following two bytes.

Address 80C. This is a CALL instruction which calls the routine located at 1B0. This is the address of the music programme.

At 80F the instruction is to load the accumulator with the contents of the immediate byte which contains 0A. This 0A is the most significant byte of the address for the letter table. As 800 already contains the byte 00 (the least significant byte of the address for the letter table) we do not have to load it again.

At address 811 the instruction is to load the address 801 with the contents of the accumulator (800 is already loaded correctly).

At address 814, the instruction is to call the letter printing routine located at 270. This routine contains an instruction to look at location 800 and 801 and see where the look-up table is located. In our case it is at 0A00.

The final address 817 is an instruction to JUMP to address 802. This is used as a repeat function.

THE PROGRAM:

```
800  Push + + to get 802.
802  LD A,00       3E 00
804  LD (800), A   32 00 08
807  LD A, 09      3e 09
809  LD (801), A   32 01 08
80C  CALL 1B0      CD B0 01
80F  LD A, 0A      3E 0A
811  LD (801), A   32 01 08
814  CALL 270      CD 70 02
817  JP 802        C3 02 08
```

The first column is the address of the memory location in RAM.

The centre column is the assembly language in mnemonics.

The third column is the Machine Code listing.

PITCH TABLE:

```
0900:

          01
          00
          01
          00
          02
          03
          04
          05
          04
          05
          1F - means to return to
                    line LD A, 0A.
```

LETTER TABLE:

```
0A00:

          01
          02
          03
          00
          00
          00
          00
          00
          00
          1F - means to return to
                    line JP 802.
```

To run the program: Press: Reset, +, +, GO, GO.

This is how the sequence should be keyed:

Press RESET to get the first location. Press: *00 + 09 + 3E + 00 + 32 + 00 + 08 + 3E + 09 + 32 + 01 + 08 + CD + B0 + 01 + 3E + 0A + 32 + 01 + 08 + CD + 70 + 02 + C3 + 02 + 08 ADdress 0900 + 01 + 00 + 01 + 00 + 02 + 03 + 04 + 05 + 04 + 05 + 1F ADdress 0A00 + 01 + 02 + 03 + 00 + 00 + 00 + 00 + 00 + 00 + 1F RESET + + GO,*

You now have enough information to be able to produce your own sequence. Try a longer note sequence and a longer sentence. You have the availability of including 255 in each table.

The next issue of TE will show how to write a program to display ONE segment of any particular digit, then two segments, so that you can create your own characters.

This is the beginning to writing programmes for video games and you will be shown how to prepare the internal structure of a simple moving target game.

We will also introduce some expansion and interface projects. So, be prepared.

Use the following table to create your own tunes.

NOTE TABLE

G	01
G#	02
A	03
A#	04
B	05
C	06
C#	07
D	08
D#	09
E	0A
F	0B
F#	0C
G	0D
G#	0E
A	0F
A#	10
B	11
C	12
C#	13
D	14
D#	15
E	16
F	17
F#	18
Repeat	1E
Return	1F

Use this table to create your own words:

LETTER TABLE

	00
A	01
B	02
C	03
D	04
E	05
F	06
G	07
H	08
I	09
J	0A
K	0B
L	0C
M	
N	0D
O	0E
P	0F
Q	10
R	11
S	12
T	13
U	14
V	15
W	
X	
Y	16
Z	17
–	18
.	19
!	1A
Repeat	1E
Return	1F

NIM

Key sequence: AD, 3, E, 0, GO,GO.

When the game ends, press any key to restart.

You are playing against the computer in a battle of wits. The computer has an obvious advantage.

There are 23 matches and you take turns in removing 1, 2, or 3 matches. The object of the game is to make the computer take the last match.

At each turn you can only take 1, 2, or 3. The computer lets you go first. The number of matches is displayed on the last two digits of the display. When you press a button, say 3, this will be displayed as **Y 3.** This indicates you took 3 matches. Then it will display **I 3.** This will mean the computer took 3 matches.

It is now waiting for your next move. Be careful, the computer is smart. It is waiting for you to make your first mistake. It will then take immediate advantage of it.

If you are playing a purely random game, you will find yourself holding the last match every time. The computer will let you know too! Read the message it displays!

The computer is a pretty bad loser but be thankful it doesn't self-destruct in disgust!

If you are playing a careful **well-calculated** game, you can **WIN.** So, try your skill and see the winning message.

LUNA LANDER

Key sequence: AD, 4, 9, 0, GO, GO.

Set speed control to your level of skill (strength of gravity). When the game ends, press any key to restart.

You are in a luna module, orbiting some 50 kilometres above the luna surface. You have 20 litres of astro fuel left and you have to land your spacecraft without denting either the moon or the craft.

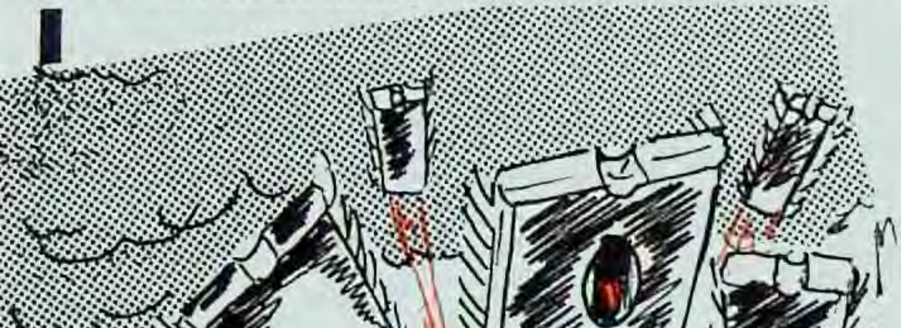
Gravity is constantly pulling you down and you can only slow your descent by blasting with your retro rockets.

Your height is indicated by the first two digits and this starts at 50. Watch yourself descend without blasting your retros and as you fall, you will descend faster and faster - until you HIT!

Press any key to restart (except reset). To blast for a short time, press: +. This may slow you a bit and to slow yourself down more, press + several times. If you over-do this command, you will slow down to zero velocity and even start going UP! Never move upwards as this is a waste of fuel.

Every time you blast, your fuel goes down by ONE LITRE. Once your fuel runs out, you can't fire any more and you start falling towards the luna surface.

So, use your fuel wisely to survive!



INVADERS

Key sequence: AD, 3, 2, 0, GO, GO.

Set speed control to your level of skill. When the game ends, press any key to restart.

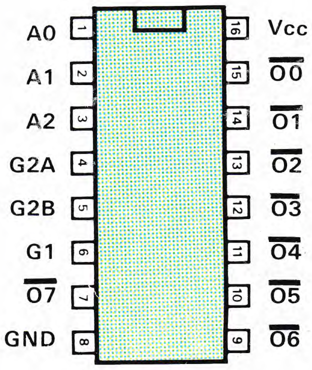
The object of **INVADERS** is very simple. You shoot anything that moves! Your position is represented by the number on the left. The invaders appear from the right. They shift across the display and if they touch you - "POW". The game ends and the score is shown on the screen.

You can't stop the invaders advancing but you can defend yourself by blowing them up. The fire button is button 0 but you can only destroy those invaders which have the same number as your space-gun.

To change your number to match the first invader, press the + button. You can only increase your number and not reduce it. By using the + and fire keys you will be able to keep the invaders at bay. To improve your skill, advance the speed control. You can also destroy those behind the front invader by matching the numbers.
Try your fire power, you'll find it most absorbing.

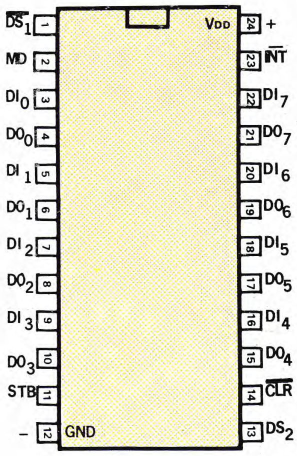
DATA SHEET 10

74LS138



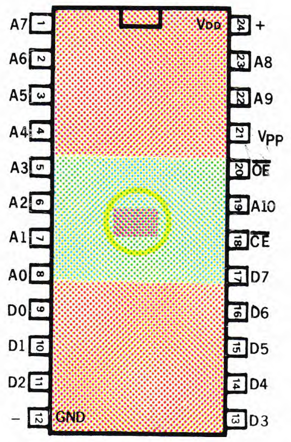
1 - 8 DECODER

8212



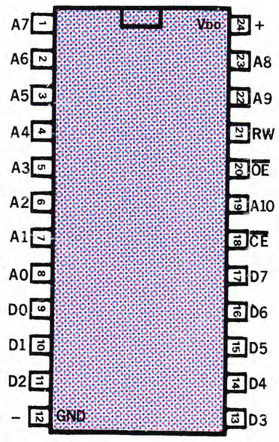
LATCH

2716



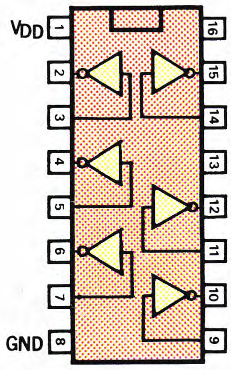
2k x 8 BIT EPROM

6116



2k x 8 BIT RAM

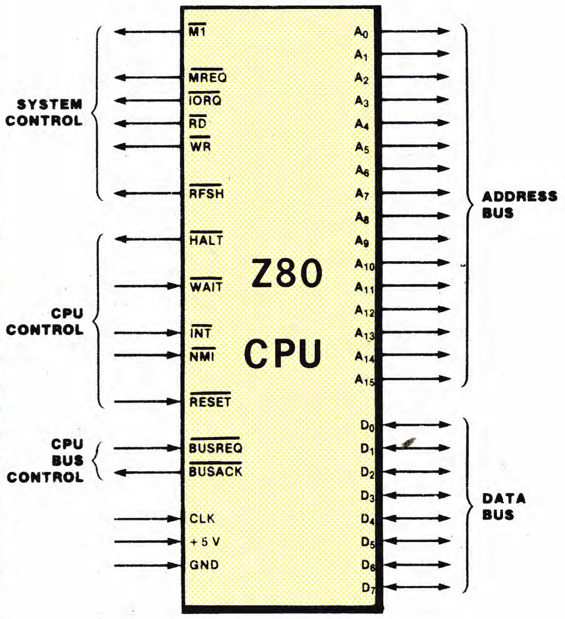
4049



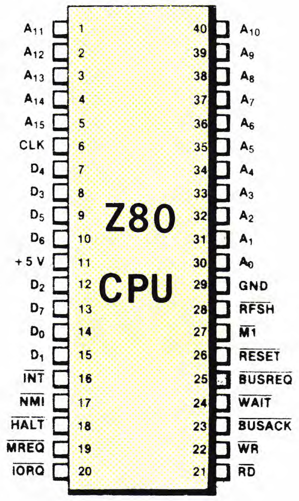
HEX INVERTER

Z80 CPU

SOURCE: SGS DATA BOOKS.

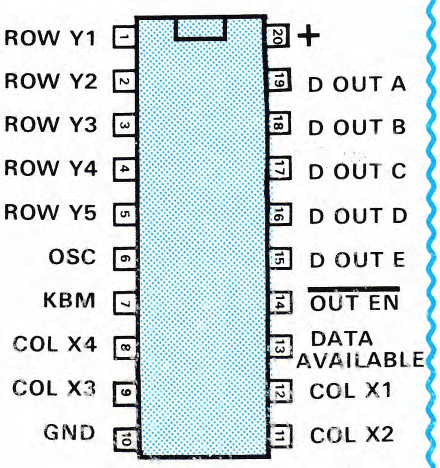


Z80 LOGIC FUNCTIONS

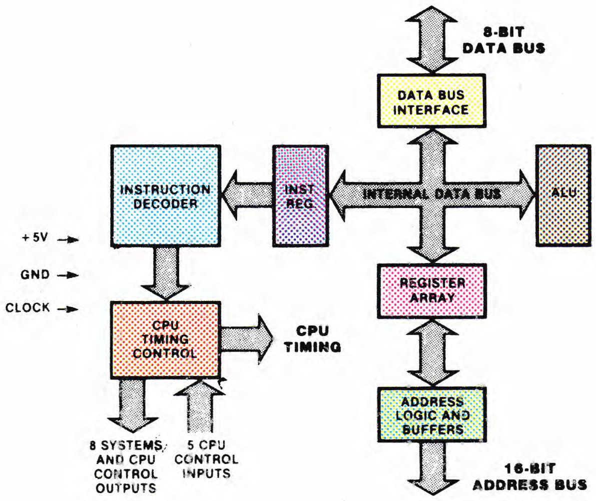


Z80 PIN OUTS

74C923



20-KEY ENCODER



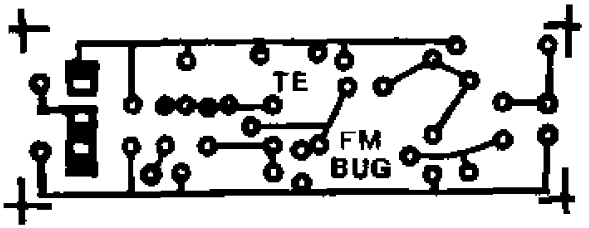
Z80CPU BLOCK DIAGRAM

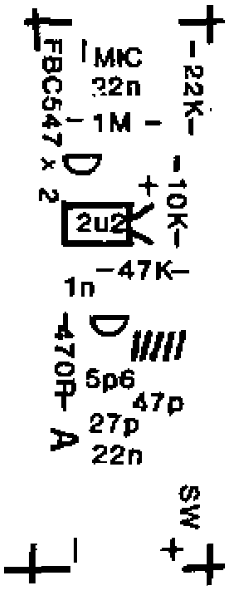
CPW

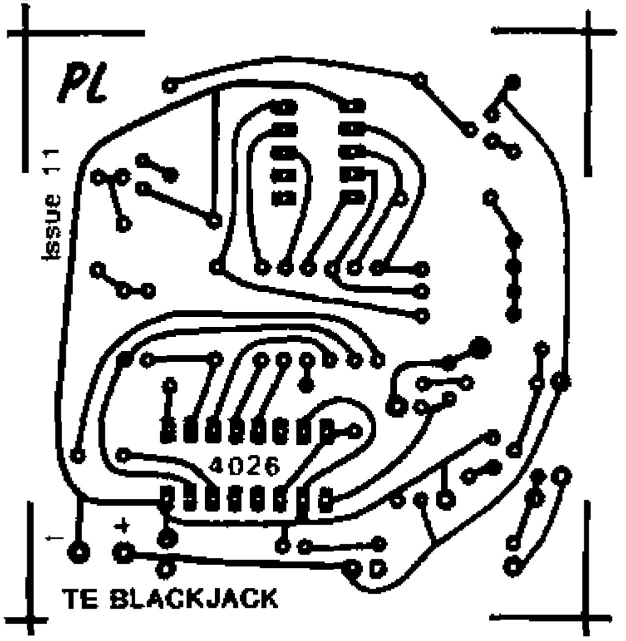
KS

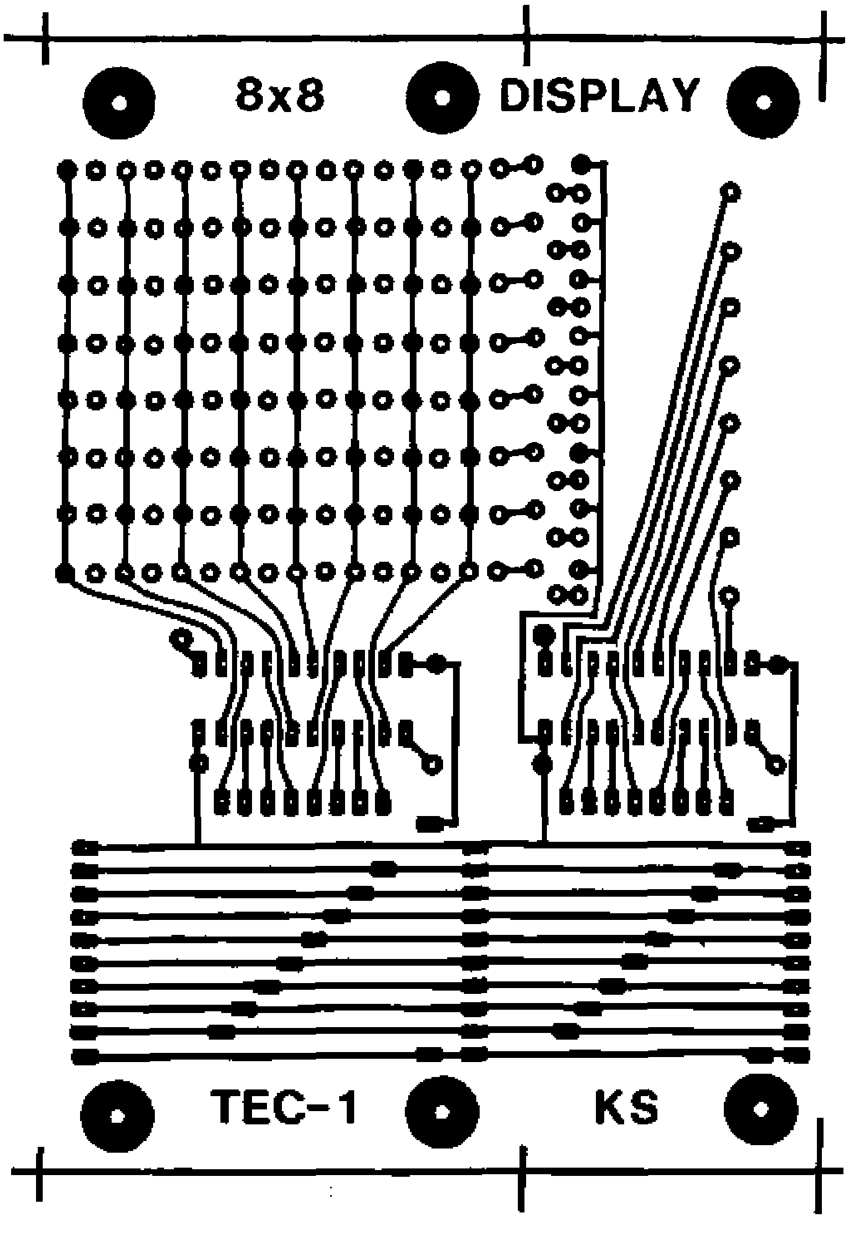
DESIGNED and PUBLISHED by COLIN MITCHELL at 35 Rosewarne Ave., CHELTENHAM, VICTORIA, 3192 (03) 584 2386

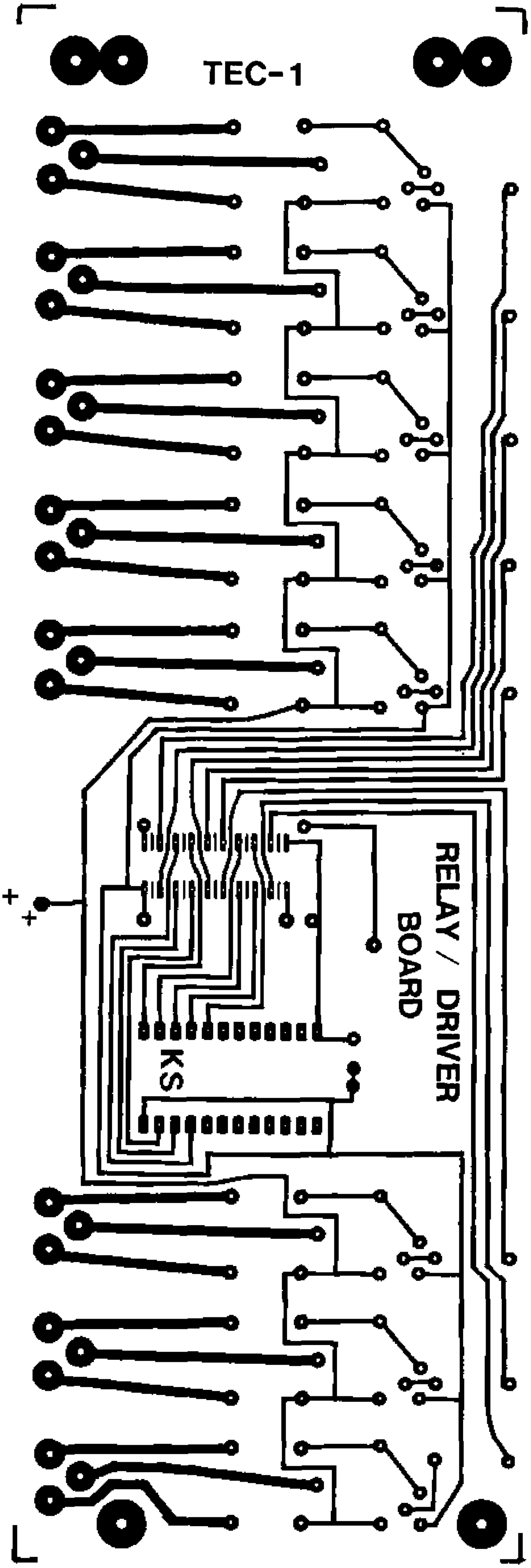
PC ARTWORK

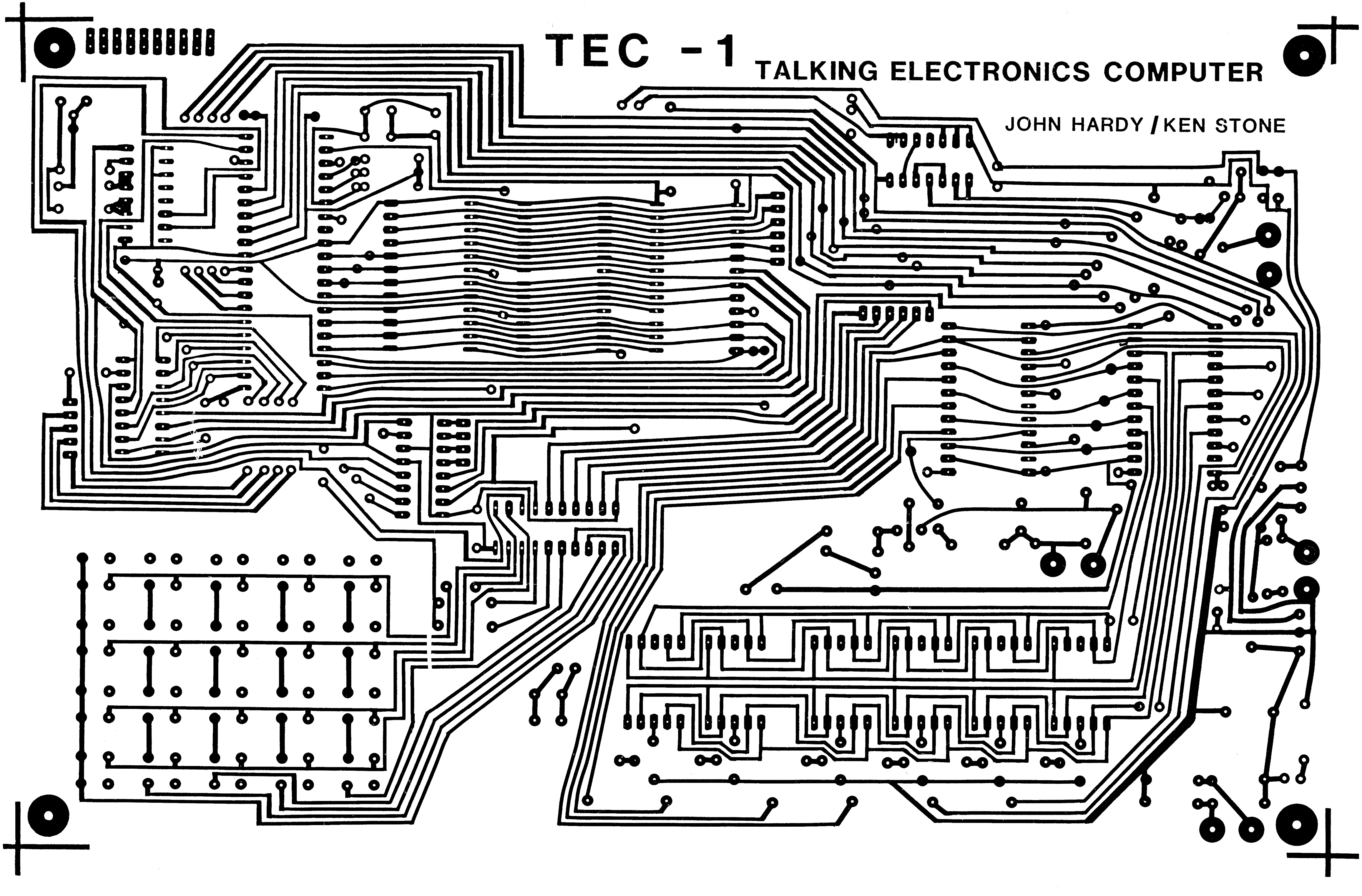

TE
FM
BUG

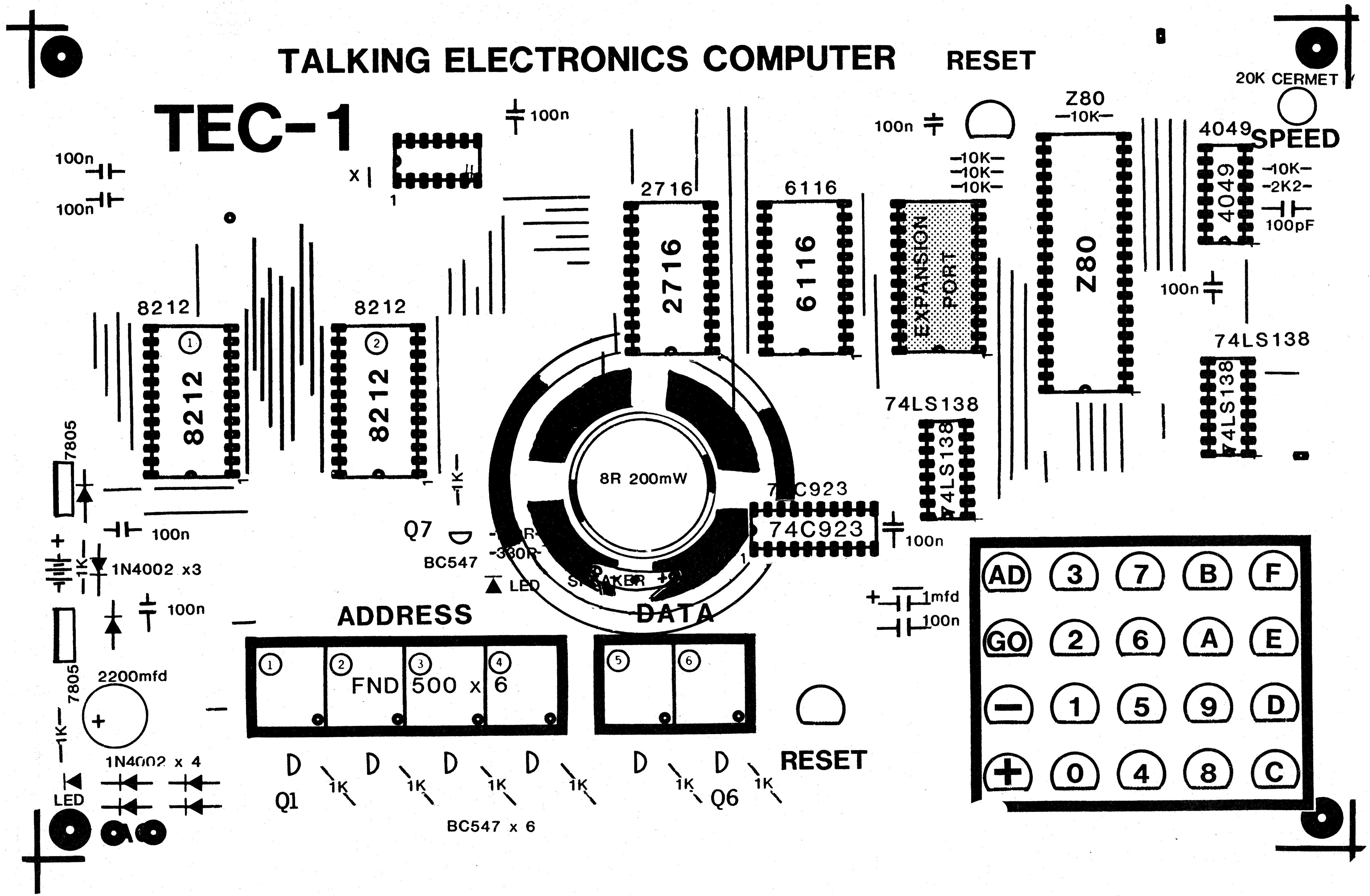

MIC
22n
1M
22K
FBC547 x 2
D
10K
2u2
47K
1n
D
470R
5p6
47p
27p
22n
A
SW

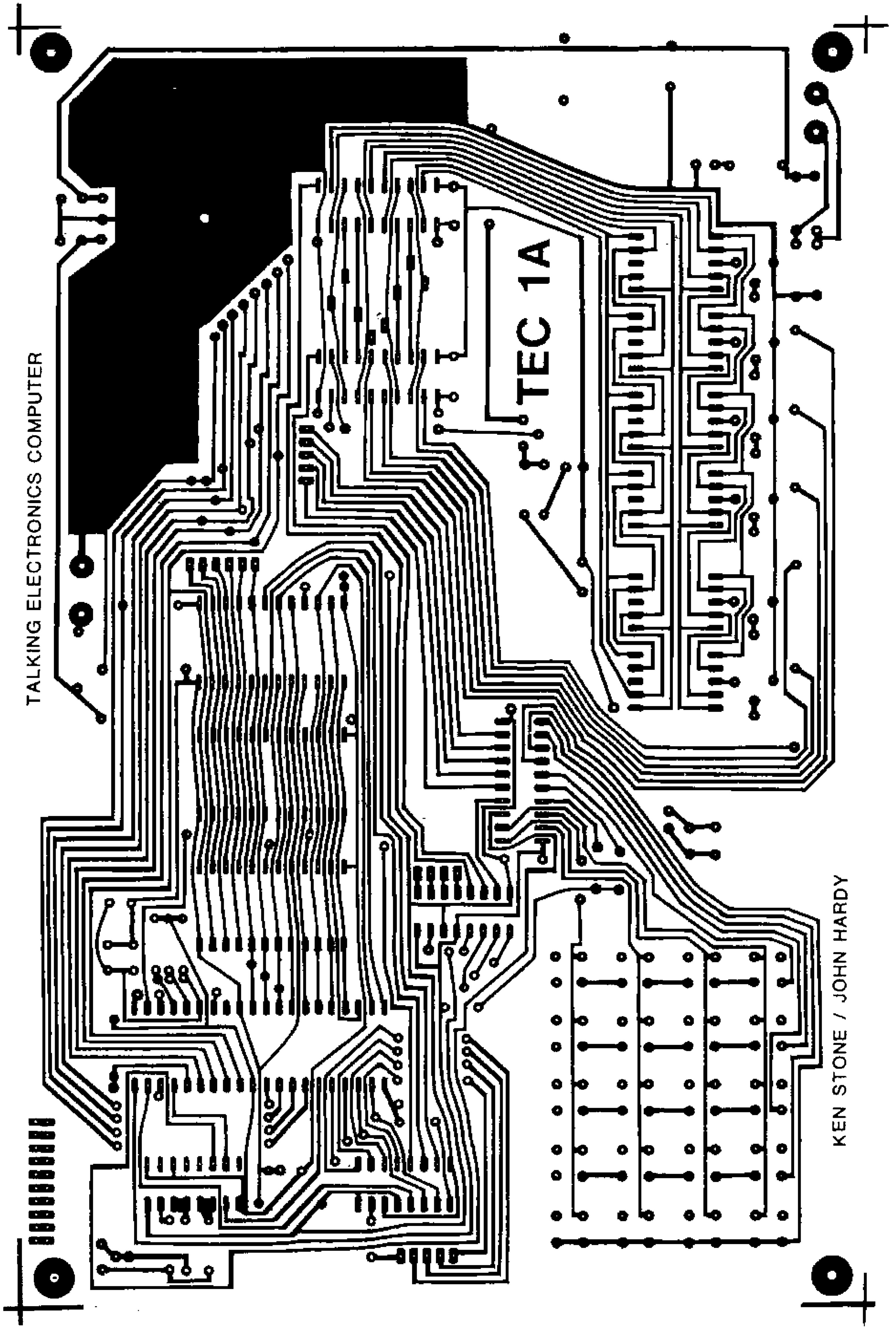

PL
Issue 11
4026
D
TE BLACKJACK


8x8
DISPLAY
TEC-1
KS


TEC-1
RELAY / DRIVER
BOARD
KS


TEC -1
TALKING ELECTRONICS COMPUTER
JOHN HARDY / KEN STONE


TALKING ELECTRONICS COMPUTER
TEC-1
RESET
20K CERMET
SPEED
Z80
-10K-
4049
-10K-
-2K2-
100pF
100n
100n
100n
100n
x
1
2716
6116
-10K-
-10K-
-10K-
EXPANSION PORT
8212
8212
① 8212
② 8212
74LS138
74LS138
74LS138
74LS138
74C923
74C923
100n
8R 200mW
SPEAKER
Q7
BC547
-1K-
-330R-
LED
7805
7805
100n
1N4002 x3
-1K-
+
100n
2200mfd
+
-1K-
1N4002 x 4
LED
ADDRESS
DATA
① ② FND ③ 500 x ④ 6
⑤ ⑥
+ 1mfd
100n
RESET
D 1K D 1K D 1K D 1K D 1K D 1K
Q1
Q6
BC547 x 6
AD 3 7 B F
GO 2 6 A E
− 1 5 9 D
+ 0 4 8 C

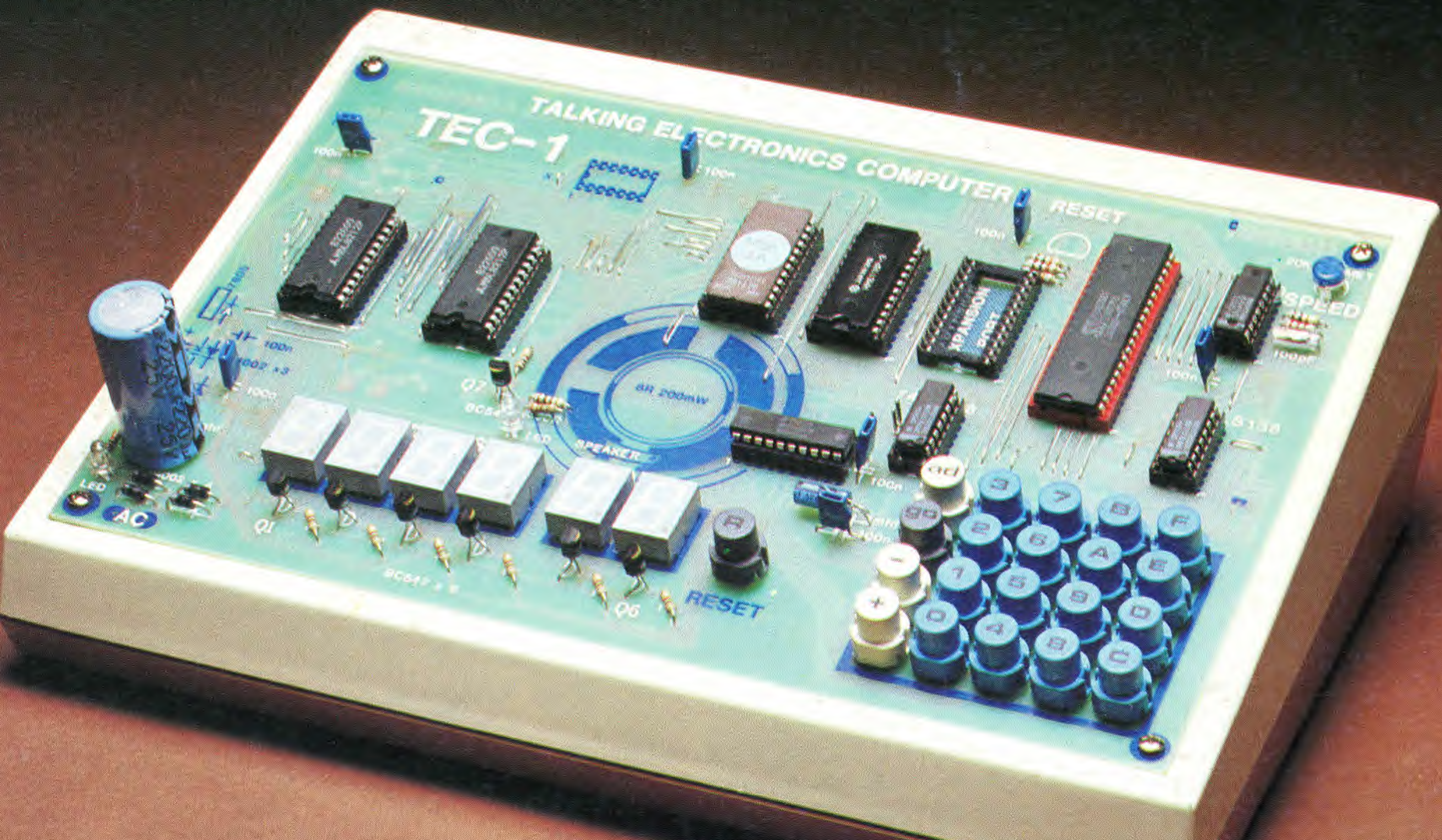

TALKING ELECTRONICS COMPUTER
TEC 1A
KEN STONE / JOHN HARDY

TALKING ELECTRONICS®

★ $1.80
$2.40NZ
Issue No 11

2 'Add-ons' for the TEC-1

★ 8x8 DISPLAY
★ RELAY DRIVER BOARD



DUAL POWER SUPPLY
BLACK JACK
FM BUG

TEC-1

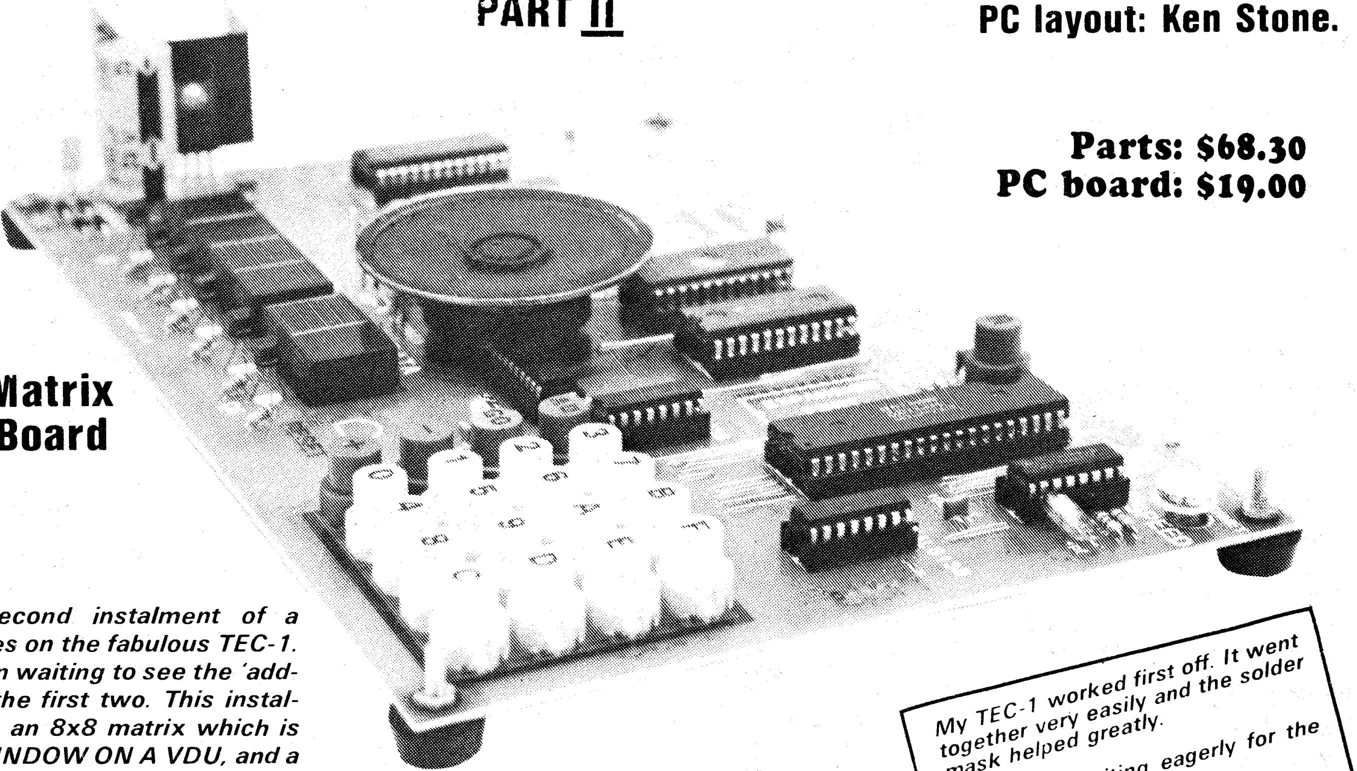
TALKING ELECTRONICS COMPUTER

PART II

—by John Hardy
PC layout: Ken Stone.

Parts: $68.30
PC board: $19.00

★ **8 x 8 Matrix**
★ **Relay Board**



This is the second instalment of a continuing series on the fabulous TEC-1. If you have been waiting to see the 'add-ons', here are the first two. This instalmant describes an 8x8 matrix which is effectively a WINDOW ON A VDU, and a RELAY BOARD which contains a set of 8 relays so that the TEC-1 will access the outside world.

You can operate globes or motors via the relays or drive them directly via the set of transistors included on the board.

The 8x8 matrix is multiplexed and driven by its own set of latches. In the ultimate you will be able to get incredible movement, but in the elementary stage its simple illumination and shift patterns.

Now that you have got this far, read on.. .. .. ..

The introduction of the TEC-1 in the previous issue caused quite a lot of interest from a new group of hobbyists. Was this due to the colour cover or the presentation of a cheap computer? Who knows?

In any case, we are pleased it took their attention. Everyone will benefit with the increased sales it produced.

We noted the number of orders increased dramatically with many coming from names and places not on our mailing list or files.

The requests for TEC-1 outstripped the availability of kits and we soon realized the small markets in Australia had to be by-passed in preference to direct importing.

Sales are still peaking but I think many readers are still waiting for the full range of "add-ons" before launching into purchasing a kit. Let's hope some of your answers will be answered when you see the extent of the projects in this issue. And this is only just the beginning.

We have already designed more than 9 different expansions for the TEC-1 and this will take it into the field of a fully-fledged demonstrator.

Within the first week we received 5 letters from constructors who had the TEC-1 operating from the instant of switch-on.

Although extremely simple, the TEC-1 works very well. Some of its features are novel while others are a little outdated. The speed control is a novelty while the 8212's have been around for years and are now getting towards the end of production. We found this out as they are now quite

My TEC-1 worked first off. It went together very easily and the solder mask helped greatly.

I am now waiting eagerly for the next installment.

Steven Truscott, 2287

When the TEC-1 was introduced in issue 10, my son and I agreed it should be a good place to learn about computers. We built the kit and it worked straight away. We were quite impressed by the quality of the PC board and the technical details in issue 10.

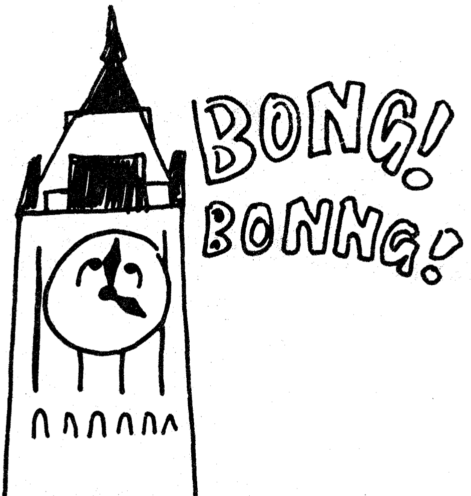
We are now in the process of making a case for the TEC-1 and are in complete agreement that the computer should be exposed so that we keep in touch with the "operations". We will be fitting a hinged perspex cover to keep out dust etc. The only problem is the heatsink on the 7805. We have decided to mount the regulator under the board, near one corner and run three lines to the appropriate lands. Everything will then be neat, firm and tamper-proof.

Martin Hulsman, 7310.

. . . .In the expansion port on the TEC-1 I would mount one of those IC sockets with a little lever on it. They are expensive but make it easier to remove the expansion plugs.

Raymond Green.

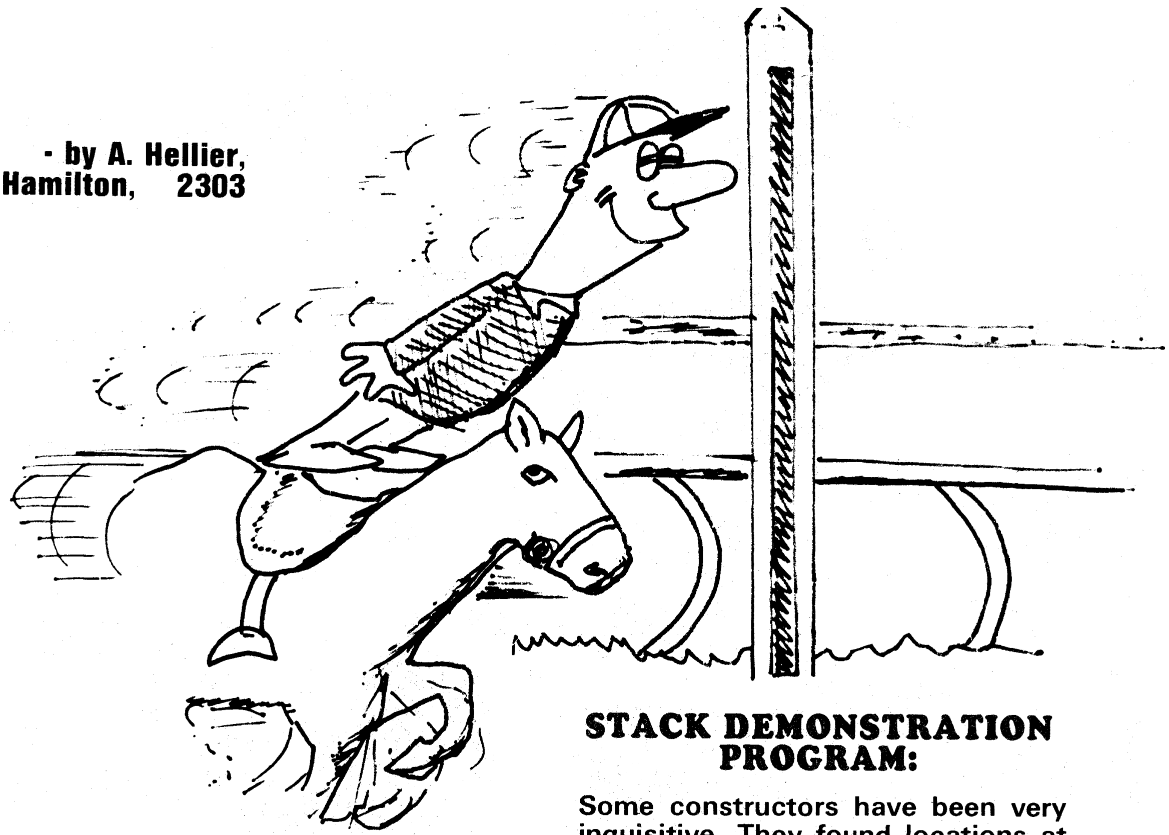
. . .cont. P. 14.



- by A. Hellier,
Hamilton, 2303

BIG BEN CHIMES

```
RESET 00 + 09 + 3E + 00 + 32 +
00 + 08 + 3E + 09 + 32 + 01 + 08
+ CD + BO + 01 + 3E + 0A + 32 +
01 + 08 + CD + 70 + 02 + C3 +
02 + 08.

ADDRESS: 0900 + 11 + OD + OF
+ 08 + 00 + 00 + 00 + 00 + 08 +
OF + 11 + OD + 00 + 00 + 00 +
00 + 11 + OF + OD + 08 + 00 +
00 + 00 + 00 + 08 + OF  + 11 +
OD + 00 + 00 + 00 + 00 + OD +
00 + 00 + 00 + 00 + OD + 00 +
00 + 00 + 00 + OD + 00 + 00 +
00 + 00 OD + 1F.

ADDRESS OAOO + 02 + 09 + 07
+ 00 + 02 + 05 + OD + 00 + 00 +
00 + 00 + 00 + 00 + 00 + 1F.

RESET  +  +  GO  GO.
```

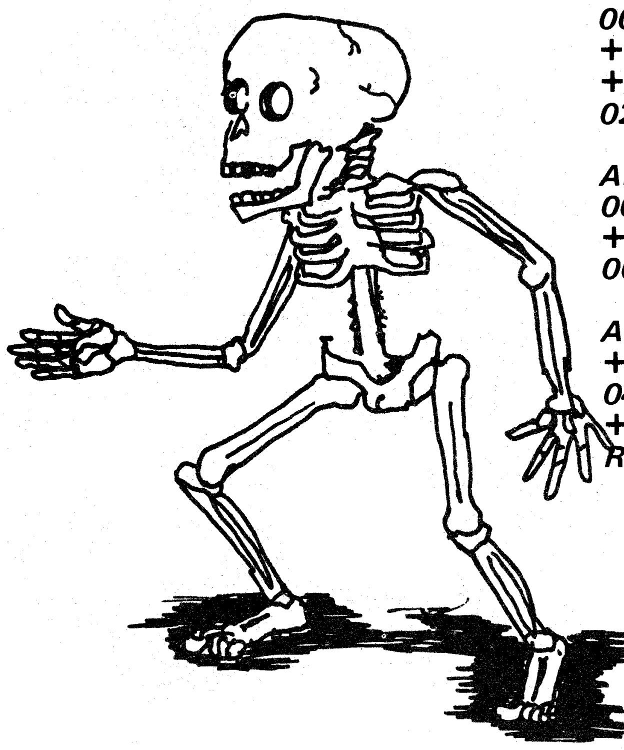


WINNERS CALL

```
RESET + 02 + 08 + 06 + 0A + OD
+ 12 + 12 + 12 + OD + OD + OD
+ OA + OD + OA + 06 + 00 + 06
+ OA + OD + 12 + 12 + 12 + 06 +
06 + 06 + OD + OD + OD + 00 +
00 + 1E.

ADDRESS 01B0  GO  GO.
```

"YOUR'RE DEAD" FUNERAL DIRGE

```
RESET 00 + 09 + 3E + 00 + 32 +
00 + 08 + 3E + 09 + 32 + 01 + 08
+ CD + BO + 01 + 3E + OA + 32
+ 01 + 08 + CD + 70 + 02 + C3 +
02 + 08.

ADDRESS 0900 + 03 + 00 + 03 +
00 + 03 + 03 + 00 + 06 + 00 + 05
+ 05 + 00 + 03 + 00 + 02 + 03 +
00 + 1F.

ADDRESS OAOO + 16 + OE + 14
+ 11 + 05 + 00 + 04 + 05 + 01 +
04 + 1A + 1A + 00 + 00 + 00 + 00
+ 00 + 1F.
RESET  +  +  GO  GO.
```



STACK DEMONSTRATION PROGRAM:

Some constructors have been very inquisitive. They found locations at the high end of RAM which they could not remove! (This is because the TEC-1 was replacing them again). This was quite puzzling as we know anything in RAM can be removed and written over.

But this area is special and is called the STACK area.

When a PUSH instruction is executed by the Z80, the contents of the location are loaded into this area. The stack starts at OFFO for MON-1A EPROMS and OFDO for MON-1B EPROMS. It advances downwards, towards the LOW addresses in the 6116.

Each time a POP (or PULL) instruction is executed, the last item to put onto the stack is removed and the stack gets smaller. Otherwise it gets bigger and bigger.

If a program contains too many PUSH instructions, the stack will grow and eventually hit the program. This will make the computer CRASH!

The simple program below pushes register pair (they must be in pairs) **AF** into the stack and completely fills the RAM.

Try the program and watch the computer CRASH.

```
11 AA BB
D5
C3 03 08
```

RESET, GO.

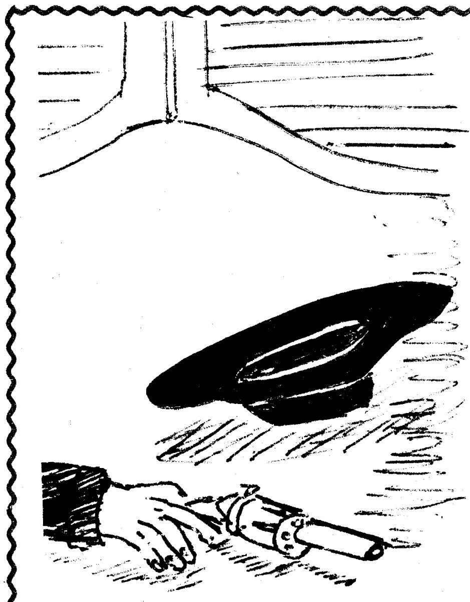
Increment the address and prove the 6116 is completely filled with AA, BB, AA, BB etc.

Change AA, BB to CC, DD and repeat. Check the RAM and read its contents.

QUICK DRAW

THE PROGRAM

```
        LD A,00        800  3E 00
        OUT(1),A       802  D3 01
START   LD DE,00       804  11 00 00
DELAY   DEC DE         807  1B
        LD A,D         808  7A
        OR E           809  B3
        JP NZ Delay    80A  C2 07 08
        LD A,E3        80D  3E E3
        OUT (2),A      80F  D3 02
        LD A,08        811  3E 08
        OUT (1),A      813  D3 01
LOOP 1  HALT           815  76
        AND OF         816  E6 0F
        CP OC          818  FE oC
        JP Z,Right     81A  CA 24 08
        OR A           81D  B7
        JP Z,Left      81E  CA 29 08
        JP Loop 1      821  C3 15 08
RIGHT   LD A,01        824  3E 01
        JP Finish      826  C3 2B 08
        LD A,20        829  3E 20
        OUT (1),A      82B  D3 01
LEFT    LD A,20        82D  3E 28
FINISH  OUT (2),A      82F  D3 02
        HALT           831  76
        JP Start       832  C3 00 08
```



QUICK DRAW is a reaction game for two players.

To start the game, press RESET, GO.

After a DELAY, as determined by the delay routine at 804, the letter G will appear on the screen. The first player to press his button will be detected by the computer and result in the figure 1 appearing on the appropriate end of the display.

Player 1 uses the + button and player 2 uses the 'C' button.

Any button can be pressed to reset the game.

The first instruction is to load the accumulator with zero and output this to port 1 to prevent odd segments lighting up when the game is reset.

At address 804, the register pair DE is loaded with the value 00, 00. Surprisingly, this creates the longest delay as the first operation in the delay routine is to decrease the lower byte (register E) by one. This immediately removes the value of zero from the pair and when D is loaded into the accumulator, and the logical OR operation performed, the answer will only be zero when both the accumulator and register E are completely zero.

If one or both are not zero, the program will jump to instruction 807 whereupon register pair DE will be decremented by ONE. This loop will be cycled 256 x 256 times and each time will occupy quite a number of machine cycles.

This results in the letter G (for GO) taking a few seconds to appear on the screen. This creates the same effect as the delay circuit in the Quick Draw project described in issue 5.

When the register pair becomes zero, the program is advanced one address and the accumulator is loaded with the value E3. The value E3 will produce the letter G on the screen and the location of this letter is determined by loading the accumulator with 8 and outputting it to port 1.

The computer is now HALTED and waits for an input instruction. If any of the keys are pressed, the 74c923 will activate the NON-MARSKABLE INTERRUPT line and present data to the Z80 according to the value of the key.

If key C is pressed, the value 1010 is placed in the accumulator. This is then logically ANDed with the value F (1111) and the result appears in the accumulator.

When a number O-F is ANDed with 1111, the answer will be exactly the same as the number itself. In actual fact, this AND OF operation is not required for the jump right command and you can ignore it.

After the AND OF operation, the number we are looking for is 1010 (for a jump RIGHT). The answer is compared with OC and the Z80 does this by effectively performing a subtraction operation in which the value C is subtracted from the contents of the accumulator.

If the answer is zero, the computer is instructed to jump to address 824. If the answer is not zero, a logical OR operation is performed with the accumulator as the operator and also the operand.

Since the accumuator is zero, the answer will be zero. Thus the program will advance via a jump instruction, to 829. If neither of the conditions are met, the CPU will jump to 815 and wait for a key to be pressed.

If the processor advances to location 824, the accumulator will be loaded with the value 1 and told to jump to address 82B. This value 1 is outputed to Port 1 and sets one of the latches ready to display the far right-hand digit.

At address 82D, the accumulator is loaded with 28 so that the segments 'a' and 'b' will illuminate when the value 28 (not twenty eight but two, eight) is outputed to Port 2.

Finally the processor is told to HALT at address 831.

On the other hand, if the program advances to 81E, the processor is told to jump to address 829 where the accumulator is loaded with 20 so that the far left-hand display will be activated when port 2 is given the value 20.

The program can be re-started by pressing ANY key, and the accumulator is loaded with zero at 800 so that odd characters do not appear on the screen.

Here are 6 simple experiments which can be performed on this program to better understand how it operates:

1. Load address 801, 2, 3 and 4 with 00 and play the game a few times. Notice how odd figures appear on the screen. Replace the correct program values and continue:
2. At address 812, load the value 10, or 04 or OF and note the different effects.
3. At address 816 and 817, insert 00 00. What effect does this have?
4. At address 819, insert OD or OE or OF. What is the result?
5. At address 829, load the value 10 and see the result.
6. Finally, insert the value 01 at location 806. Try the value 06, OA BB or FF. What effect do they have?

difficult to obtain. The other chip on short-supply is the 74c923 as it is only made by one manufacturer.

The FND 500 or FND 560 displays are no longer in production by Fairchild (as they have ceased to produce OPTO devices) however other suppliers have produced identical replacements.

Apart from this, the TEC-1 is straightforward.

Out of the first 300 kits we had reports from only 6 readers who had trouble in getting the computer to work.

Paul had incorrectly placed the "six" button in the keyboard so that the flat was 90° out of position. The TEC-1 came on at address 0800 but the keyboard did not operate.

The 74c923 detected key 6 as being pressed due to the wiring of the contacts and the NMI being activated.

We traced the fault by removing the 74c923 and checking pins 8, 9, 11, 12 and 1, 2, 3, 4, 5 with an ohmmeter. The short between pins 2 and 9 showed key 6 to be at fault and that was when we noticed it!

Another TEC-1 came in with a very faulty + button and a broken PC line under one of the 7-segment displays.

Two computers had shorts between tracks around the memory section where the tracks are very close to one another.

But possibly the worst effort came in the post last week. The 74c923 socket had been made up with a 14 pin and an 8 pin so that 2 pins projected too far. The chip has been inserted so that pin 2 connected with pin 1 on the circuit. The other fault was a jumper missing near the first 8212. Both of these faults showed lack of inspection. When they were repaired, the computer worked perfectly.

Finally a constructor arrived with a TEC-1 under his arm. It gave an occasional display of odd segments and a beep from the speaker when switched on. The trouble was traced to a faulty Z80!

Apart from the above cases, the TEC-1 seems to offer a high degree of success.

If you have any problems with your unit, let us know. We want to present them in the next issue, under FAULT FINDING.

The introduction of the TEC-1 is primarily intended to unlock three areas of microprocessor design. These are:

1. To teach Machine Code programming.
2. To teach the art of creating Video Games and,

3. To access the real world.

MACHINE CODE programming is the skill of telling a microprocessor what to do by writing directly into a memory bank. The memory can either be a tempory storage (RAM) or permanent storage ROM. The main difference between these two is ROM will retain its memory when the power is turned off whereas RAM will lose its contents.

We can use this feature to write into RAM and then erase the program by turning off the power. Alternatively we can simply write over the old program. Both ROM and RAM are used in the TEC-1.

The Z80 has the capability of accepting over 700 instructions, some of which are 2-byte. These take the form of a two digit number which is written in hexadecimal form. The first large table in this installment shows how to write any of these numbers and explains how we arrive at C2 or E5 or D7 as a value in this instruction set.

A typical instruction is 3E. This means "load the accumulator (register A) with the following value . . ." Once you remember some of these instructions you will see why we have concentrated on Machine Code Programming.

Instructions such as 76 for HALT, C9 for RETURN and C2 for JUMP NOT ZERO are quite easy to remember. The meaning of JUMP NOT ZERO needs a little explanation. After C2 you must insert 2 bytes because the computer will interpret whatever is placed as the next two bytes as an address location. For example, 20 08 will tell the processor to jump to 0820. The instruction C2 also infers that if the program IS ZERO, the processor will proceed to the next instruction.

Machine Code is the only approach for video game development. It produces the fast-moving games as seen in the latest coin-in-the-slot machines. Any of the games on cassette are theoretically possible with the TEC-1. Mind you, we will not be advancing to the complexity of colour or the swirling action of VENUS, but in a developmental way you will have the opportunity to program sections of a video game and watch the result.

PROGRAMMING STARTS HERE:

By now you will have completed the first 7 experiments and possibly tried some additional programming of your own.

But just in case you did not absorb all the facts, we will go through some of the experiments again to make sure everything is understood.

Just before we start, key the following:
Reset, two, +, eight, +, two, +, five, +, fourteen, + four, +, sixteen, +, 1A, +, zero, +, zero, +, zero, + zero, +, zero, +, 1E, ADdress 2, 7, 0, GO, GO.

DO YOU AGREE?

You should remember a few simple programs like this, to impress your friends.

The Z80 instruction-set has 8 special restart instructions which are single-byte instructions. By keying in one of the following: C7, CF, D7, DF, E7, EF, F7, or FF the computer will go to address location 00 or 08, 10, 18, 20, 28, 30, 38 and will receive information to go to the beginning of the 2 tunes and 3 games.

The 2 tunes are accessed by EF and F7. Push Reset, EF, GO. EF tells the Z80 to go to location 28 in the EPROM. At this address is an instruction 21 which tells the Z80 to load register pair HL with the contents of another location which is the beginning of the song table.

F7 directs the Z80 to go to location 30 and this address directs the Z80 to another song table.

We do not use C7, E7 or FF but the other 3: CF, D7 and DF select the commencement of the three games.

Try them.

When the computer is turned on, or the reset button pressed, the first available address is 0800.

This means the address locations from zero to 0800 have been allocated to the 2716 EPROM and most of this has been filled with start-up instructions and games. This includes a music table and letter table which are user accessible and a number table which is only computer accessible.

Actually the MON-1 EPROM has been filled from zero to 05A7, and we will now look at how many locations this represents.

LEARNING HEX

We know the TEC-1 is programmed in hexadecimal. This means the locations commence at 0000 and increase: 0001, 0002, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 3A, 3B, 3C . . . . . .9D, 9E, 9F, A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, AA, AB, AC, AD, AE, AF, B0, B1, B2, B3, etc up to 00F2, 00F3,00F4,00F5,00F6, 00F7, 00F8, F9, FA, FB, FC, FD, 00FE, 00FF.

If you fill in all the blanks between 0000 and 00FF, you will find there are 16 lots of 16 addresses. This is equal to 256 locations.

So, between 0000 and 00FF there are 256 address locations.

The next location 0100 (it is best to say oh, one oh, oh as the location is not really one hundred). Between 0100 and 01FF there are another 256 locations.
Between 0200 and 02FF there are another 256 locations.
0300 - 03FF = 256 locations
0400 - 04FF = 256 locations
0500 - 05A7 there are ??? locations.
Let's work it out. The number of locations in A7 is: A x 16 + 7
= 10 x 16 + 7 = 167.

The total number of address locations which have been programmed into the EPROM is:
5 x 256 + 167
= 1447.

We emphasise this aspect of hex numbering as it is often glossed over. Values such as 5A7 do not give us any indication of the value they represent.

To program 1447 address locations would take the best part of an afternoon as it is important to check and double-check the data at each address before running a programme. It only takes one fault in the program to upset its running with the result that hours of work will be ruined.

So, 5A7 is quite a large number and it nearly fills the EPROM. In fact there are only about 600 locations left.

The main reason for locating the beginning of the user-available section at 0800 will become obvious in a moment.

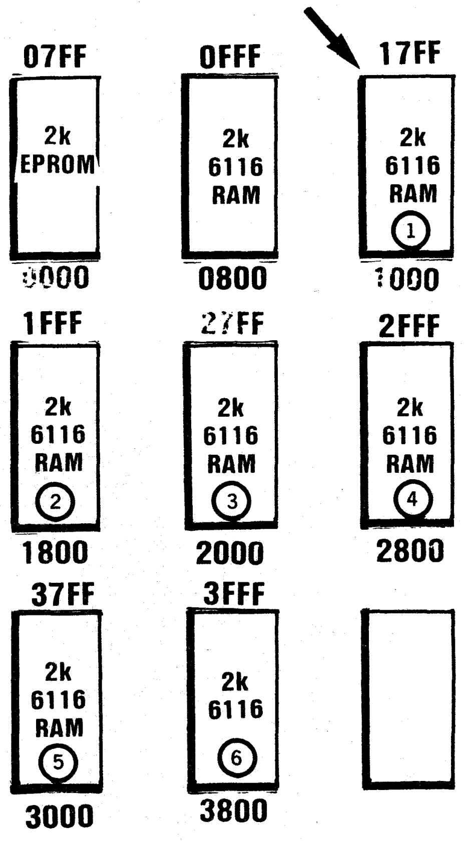
The second and most important reason for having the first user-available location at 0800 is the value it represents.

The EPROM is a 2k byte IC and this means it is capable of storing 2048 addresses. Each address will accept a number as high as 11111111 in binary, which is 255. We casually say it is a 2k EPROM but it is actually a 2k48 EPROM.

The value 2048 is equal to 0000 to ________. Let's work it out.

Divide 2048 by 256 and you will obtain the number of "groups" of locations. This comes to 8. So, 2048 is equal to 0000 to 0800 Or more accurately 0000 to 007FF. Every 0800 in hex represents 2048 locations. The next 2k starts at 0800 to 1000 or more accurately 0800 to 0FFF.

This is the EXPANSION PORT



The Memory Expansion Board, to be presented in a future article will contain 6 RAM chips. These are labelled ①-⑥ on the diagram and accessed as shown.

The hexadecimal start and finish for each "2k" of RAM on the TEC-1 is shown in the diagram above. The Z80 will access 64k of memory (65,536 bytes) or 32 chips such as 6116 or the N-MOS version 58725 by Mitsubishi. On the TEC-1, the two top address lines have not been decoded and thus the TEC will only address 16k of memory.

Between 05A7 and 07FF, the EPROM has been left blank for additional routines which will appear as MON 1A and MON 1B etc.

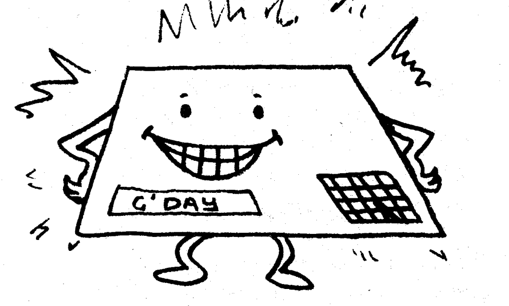
It is always wise to leave empty pockets between one program and the next in case a program needs to be extended. These empty locations can be filled at a later date with the aid of an EPROM BURNER.

Other locations in the EPROM have also been left blank. The most important of these is the "first hundred bytes". Within this section are 8 one-byte subroutine call locations such as 00, 08, 10, 18 where we can place a brief program (up to 7-8 bytes long). We can write a jump or call instruction and maybe a return instruction so that the main program needs ONLY a single byte instruction. (Normally a CALL requires 3 bytes).

You have already keyed a number of instructions and you will be starting to remember what they stand for. **3E,** for instance, means *"Load accumulator A with the following byte"*. The accumulator is one of the registers in the Z80 and there is nothing magic or complex about it.

It is merely a set of 8 flip flops which can be set HIGH or LOW to reflect the number which has been entered.

The reason why register A is so often used in programmes lies in its special feature. It is the accumulator register and this means all logic operations (such as AND, OR ADDITION and SUBTRACTION) will be performed using it.



THE ALL PURPOSE NIFTY TEC-1...

Experiment 7 combines a number of interesting features such as CALL and JUMP and we will explain what these do.

The program looks fairly simple, but this very deceptive. If you were required to program all the information to produce a running letter program, it would be like being asked to buy a bottle of lemonade but firstly manufacture and print the dollar note required to buy the bottle of fizzy. Obviously this would be an enormous task so we use AIDS to make the task easier.

In our case we use John Hardy's letter writing routine and his running or shift routine and put them together to get our sentences.

We will go through the program in the same way as carried out by the Z80:

By pressing the reset button, +, +, the computer will start at location 802. The small amount of operating brains inside the Z80 will instruct it to look at location 802 in the 6116 RAM. It will find the instruction 3E and this will tell it to load the accumulator with 00. This is one way of removing the initial 'rubbish' in a register or accumulator A.

The next instruction 32 means load the contents of accumulator A (which is 00) into the address which follows, which is 0800. This means we have removed any rubbish data and loaded it with 00.

At address location 807, 'A' will be loaded with 09. The previous value, 00 will be written over but it has already done its job of being loaded into location 0800.

At location 809, the Z80 is told to load the value 09 into address 801.

All we have done to date is simply load the first available address with 00 and the second with 09. We could have done this manually but the routine calls for the contents of 800 and 801 to be altered between two different values: 0900 and 0A00 and so the computer has to do the job.

The next instruction is a CALL instruction. The Z80 is asked to look at the address 180 in the EPROM. What it will find there is quite considerable.

Firstly it loads register pair DE with the contents of 800 and 801. At these locations it finds the address 0900. Next it takes the first byte at 0900 and plays the note corresponding to this value. This requires another table comprising byte-pairs. One of which is the pitch value and the other time-value. It continues using each byte at 0900 until either a IF or IE is recognised. With IF the computer goes to line 80F in the main program and carries out a similar procedure for the Letter Table.

After the letters have scrolled across the screen, the computer returns to the main program (location 817) and this instructs the Z 80 to jump to line 802.

WHAT YOU HAVE LEARNT FROM EXPERIMENTS 1-7:

1. The first available address is 0800.
2.The dots on the display indicate when the address OR the data can be changed.
3.Incrementing the display means to increase the address value.
4.Decrementing the display means to decrease the address location.
5. The TEC-1 is programmed in MACHINE CODE.
6. The key pad is a HEX PAD as it contains the numbers 0-9 and letters A-F.
7. The values 00 to FF represent the numbers ZERO to 255:

HERE IS THE COMPLETE TABLE:

OUR COMPLETE HEX TABLE

0	00								
1	01	51	33	101	65	151	97	201	C9
2	02	52	34	102	66	152	98	202	CA
3	03	53	35	103	67	153	99	203	CB
4	04	54	36	104	68	154	9A	204	CC
5	05	55	37	105	69	155	9B	205	CD
6	06	56	38	106	6A	156	9C	206	CE
7	07	57	39	107	6B	157	9D	207	CF
8	08	58	3A	108	6C	158	9E	208	D0
9	09	59	3B	109	6D	159	9F	209	D1
10	0A	60	3C	110	6E	160	A0	210	D2
11	0B	61	3D	111	6F	161	A1	211	D3
12	0C	62	3E	112	70	162	A2	212	D4
13	0D	63	3F	113	71	163	A3	213	D5
14	0E	64	40	114	72	164	A4	214	D6
15	0F	65	41	115	73	165	A5	215	D7
16	10	66	42	116	74	166	A6	216	D8
17	11	67	43	117	75	167	A7	217	D9
18	12	68	44	118	76	168	A8	218	DA
19	13	69	45	119	77	169	A9	219	DB
20	14	70	46	120	78	170	AA	220	DC
21	15	71	47	121	79	171	AB	221	DD
22	16	72	48	122	7A	172	AC	222	DE
23	17	73	49	123	7B	173	AD	223	DF
24	18	74	4A	124	7C	174	AE	224	E0
25	19	75	4B	125	7D	175	AF	225	E1
26	1A	76	4C	126	7E	176	B0	226	E2
27	1B	77	4D	127	7F	177	B1	227	E3
28	1C	7B	4E	128	80	178	B2	228	E4
29	1D	79	4F	129	81	179	B3	229	E5
30	1E	80	50	130	82	180	B4	230	E6
31	1F	81	51	131	83	181	B5	231	E7
32	20	82	52	132	84	182	B6	232	E8
33	21	83	53	133	85	183	87	233	E9
34	22	84	54	134	86	184	B8	234	EA
35	23	85	55	135	87	185	B9	235	EB
36	24	B6	56	136	88	186	BA	236	EC
37	25	87	57	137	89	187	BB	237	ED
38	26	88	58	138	8A	188	BC	238	EE
39	27	89	59	139	8B	189	BD	239	EF
40	28	90	5A	140	8C	190	BE	240	F0
41	29	91	5B	141	8D	191	BF	241	F1
42	2A	92	5C	142	8E	192	C0	242	F2
43	2B	93	5D	143	8F	193	C1	243	F3
44	2C	94	5E	144	90	194	C2	244	F4
45	2D	95	5F	145	91	195	C3	245	F5
46	2E	96	60	146	92	196	C4	246	F6
47	2F	97	61	147	93	197	C5	247	F7
48	30	98	62	148	94	198	C6	248	F8
49	31	99	63	149	95	199	C7	249	F9
50	32	100	64	150	96	200	C8	250	FA
								251	FB
								252	FC
								253	FD
								254	FE
								255	FF

MOVING ON. . . .

What is the next thing you would like to do?
How about turn on one segment of the display?

Key in this program:

```
RESET
3E 01
D3 01
3E 01
D3 02
76
RESET, GO.
```

This is what you have done:
Reset 3E + 1 + D3 + 1 + 3E + 1 + D3 + 2 + 76, Reset, GO.

The top LED in the first display lights up. We say the first display as it is the lowest priority digit.

In English, this is what you have done: Load register A with the value 1. Output this to port 1. Load register A with the value 1 and output it to port 2. Halt.

This is how the program is written:

```
LD A,01      800   3E 01
OUT (1),A    802   D3 01
LD A,01      804   3E 01
OUT (2),A    806   D3 02
HALT         808   76
```

We can read the value of each location by pressing RESET and then stepping through the program by pressing: + + + + + + + + + + + + +

We can alter the position of the LED which is to be lit, by altering the value of locations 801 and 805.

The whole program does not have to be re-typed. Any location can be altered as follows:

Either press Reset and + or – – – –
to get address 801.
Change 801 to 08
Press RESET, GO.

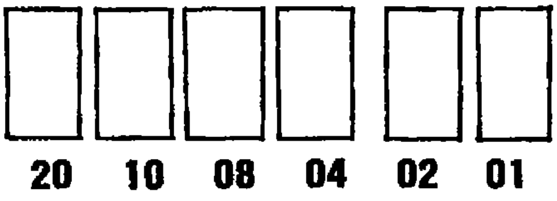
Note the LED has moved to the 4th display.

Try the values: 02, 10, 04, & 20.

You have accessed each display. Return to the first display by changing the value at 801 to: __ __
Increment to 805 and change the value 01 to: 02, 04, 08, 10, 20, 40 and finally 80.

Are you impressed? You have accessed each of the segments including the decimal point. You have become master of the display. With a little more instruction you can illuminate more than one LED in the display. But first let's see what you have learnt.

The displays are accessed as follows:



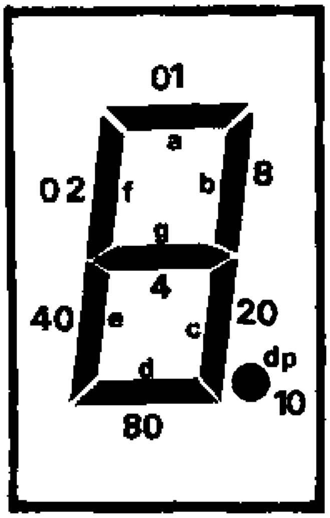
The value of each display is twice the previous and they increase from RIGHT to LEFT.

Port 1 is the cathode port. A 'HIGH' or 'ONE' in the appropriate bit 5 - 0 activates cathode 5 or 4 or 3 or 2 or 1 or 0.

This is how it works: The numbers 20, 10, 8, 4, 2, 1, are converted to binary and the computer sees them as:

```
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
```

The last line in the table is easy to read. It is 1. The second lowest line is 2, then the next line has the value 4, then 8. The first and second lines are a little more difficult to explain because they are not 16 and 32. That's the binary number but we are interested in the value we have to punch into the hex keyboard. The answer will be covered in a moment. For the present, we will look into the concept of converting binary numbers into HEX numbers.



Firstly we will give you the answers. Each segment on a display has the following values. The top LED (segment A) is lit with a value 1. The decimal point needs a value 10, the centre LED (segment G) needs the value 4 and so on.

What do you think the following program will produce?

```
3E 01
D3 01
3E 20
D3 02
76
```

Try it. Start at 0800. Enter the program, press reset, GO.

We can illuminate more than one segment at a time and more than one display at a time by changing two locations as shown in the following examples:

Try each of these examples and see a pattern of addition appearing.

At location 805, insert:

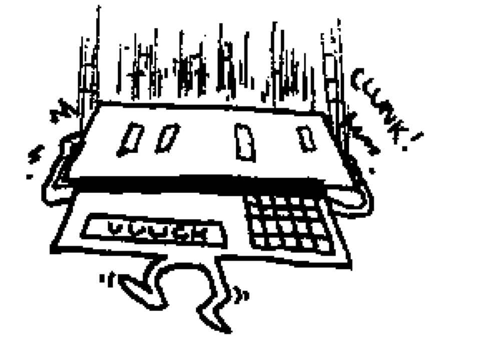
(a) 3
(b) 9
(c) 15
(d) 34
(e) 62
(f) A
(g) D
(h) F

The letters A, D and F will be quite a surprise. They also produce a reading on the display and it is obvious they have a value. Their values will be covered in the section CONVERTING BINARY TO HEX and HEX TO BINARY.

Back to the display.

More than one display can be illuminated at a time by inserting a different value at location 801. Keep say "62" at location 805 and insert the following at 801:

(a) 9
(b) 27

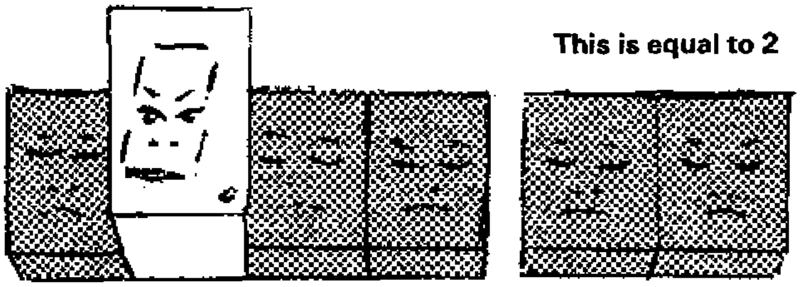


... WITH IN/OUT BOARD....

(c) 31
(d) 10
(e) A
(f) 3B
(g) 2F

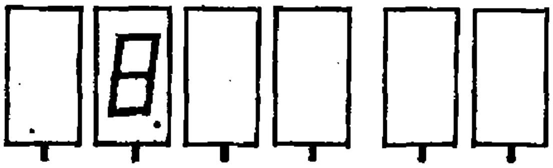
This is as far as we can go with blind experimenting. We must cover some of the background theory on hex numbers to understand what we are doing.

In experiment (d) above, we turned on the fifth display thus:

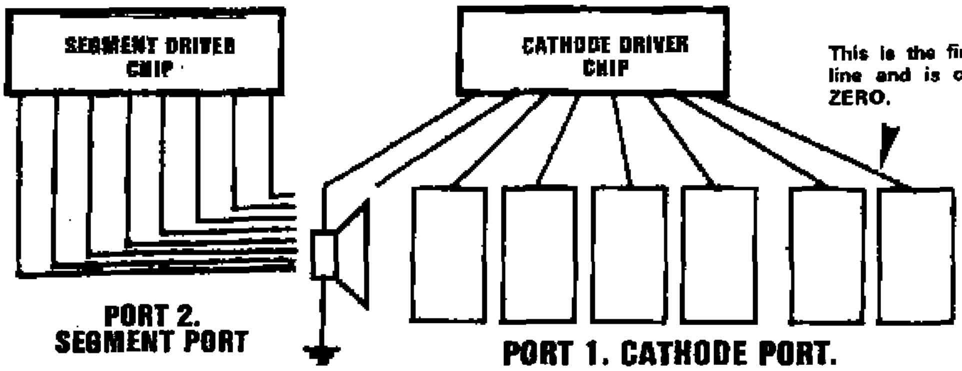


WAKING UP THE FIFTH DISPLAY

To do this we must switch the appropriate display transistor ON. (Don't worry about the segment drivers at this stage).



In the TEC-1 the displays are connected to the 8212's as follows:



The display we wish to illuminate is in the fifth output line and its binary number is: 00010000. The keyboard is in hex so to convert this to a hex number we have to break it into two groups of four:

0001   0000

This represents 1   This represents 0

The answer is 10.

This is not called ten. It is called one, zero or one, oh. Here are some more examples:

1. To iluminate the first, third, and fifth digits, this is the procedure:

The high lines from the 8212 will be:

00101010

break this into 2 groups of four:

0010   1010

This is equal to 2   This is equal to A

The answer is 2A.

OK, you don't understand how we get A. Look at the table on P 71 of issue 10. The hex number for 1010 is A.

Try these: Write the hex number for:

(a) 1100 =
(b) 1001 =
(c) 1101 =
(d) 1111 =

2. To illuminate the first, second and third displays, the HIGHs must appear in the following places:

00111000

0011   1000

3   8

Answer: 38 (in hex)

3. To illuminate all the displays:

00111111

0011   1111

3   F

4. To bip the speaker:

10000000

8   0

5. To click the speaker and turn on displays: one, two, three and four:

10111100

1011   1100

B   C

Answer: BC

6. To access the vacant cathode:

01000000

0100   0000

4   0

Answer: 40. You will notice NOTHING!

Example 3 above shows that the max hex value to illuminate ALL THE DISPLAYS is 3F. Remember this.

This is the program we are using:

This value determines which display(s) are lit and is called CATHODE ACCESSING.

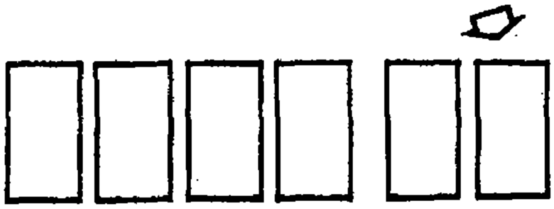
```
3E 01      Max 3F
D3 01
3E 01      Max FF
D3 02
76
```

This value determines the segments which are lit and is called SEGMENT ACCESSING.

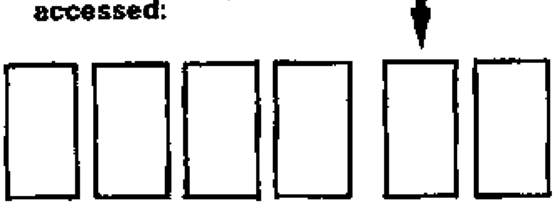
The program has two variables: lines 1 and 3. For line 1, the maximum value is 3F (this is 64 different possibilities) and line 3 has a maximum of FF (this is a maximum of 256 possibilities). These are independent variables and either can be changed at any time to any value in the range as specified above. The result is thousands of different combinations.

Here are some of the possibilities and how they are obtained:

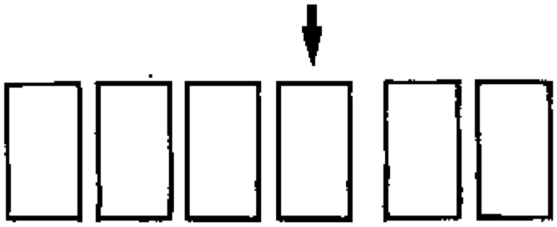
If line 1 has the value 1, we will be accessing this display: (line 3 can be any value from 00 to FF).



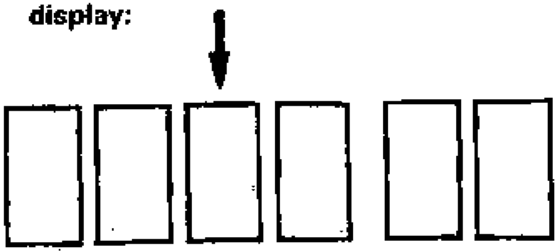
If we program 3E 02 into the first line, the following display will be accessed:



The value 3E 04 will access this display:



The value 3E 08 will access this display:

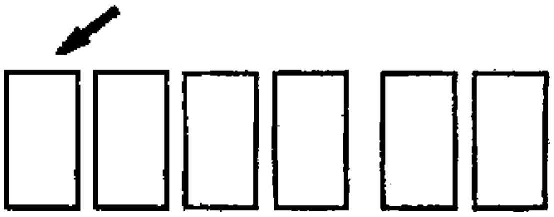


Can you see why?

The computer converts 08 to binary and it becomes 1000.
The "one" or "HIGH" corresponds to the display in the diagram above

3E 20

corresponds to this display:



Note: 20 is the Hex number and is obtained by separating 20 into 2 0.

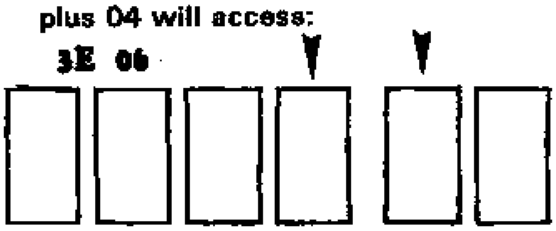
2	0
0010	0000

This gives 0010 0000 as the binary equivalent and shows the HIGH is bit 5 and this will illuminate the first digit on the display. (Note: The first output line is bit 0).

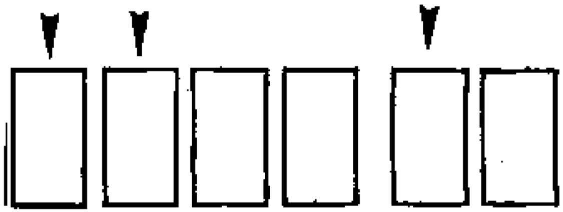
[The actual combination of segments which will be illuminated will depend on line 3 of the program and this will be explained in a moment.]

More than one display can be turned on at the same time by adding CATHODE ACCESS values. Thus 02 plus 04 will access:

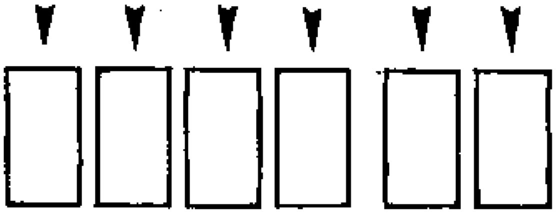
3E 06



3E 32 will access:



3E 3F will access:

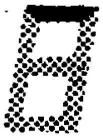


SEGMENT ACCESS

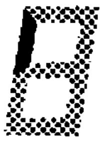
Line 3 of the program determines the letter, number or pattern which will appear in the display(s). We will take the simple case of the lowest priority display being accessed (line 1 will be 3E 01) and we will produce some interesting patterns, numbers and letters in (or on) the display.

Here are some of the results of changing the data in line 3:

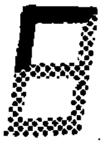
3E 01



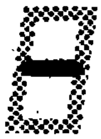
3E 02



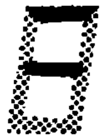
3E 03



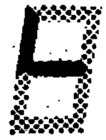
3E 04



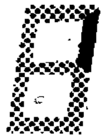
3E 05



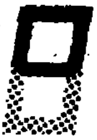
3E 06



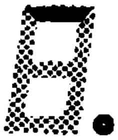
3E 08



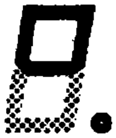
3E 0F



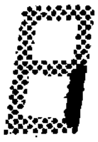
3E 11



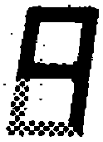
3E 1F



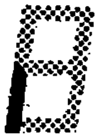
3E 20



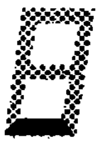
3E 2F

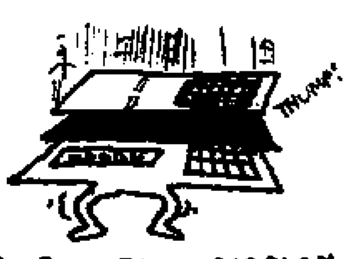


3E 40

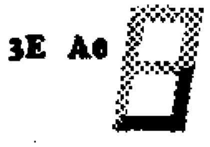


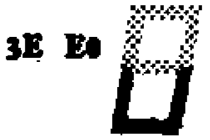
3E 80

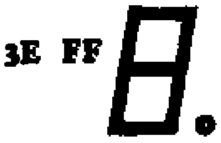




.... 8X8 MATRIX DISPLAY BOARD..,







Here is an equally interesting program to automatically increment the SEGMENT ACCESS value. The result will be to produce every combination possible on the display. Watch the results carefully and see if you can predict the next segment to appear.

At address 0800, type the following program:
(And the DELAY ROUTINE at 0A00)

```
LD A,01        800   3E 01
OUT (1),A      802   D3 01
LD B,00        804   06 00
LD A,B         806   78
OUT (2),A      807   D3 02
INC B          80A   04
CALL 0A00      80B   CD 00 0A
JP 0806        80E   C3 06 08


 0A00
                A00   11 FF FF
                A03   1B
  This is the   A04   7B
     DELAY      A05   B2
    ROUTINE     A06   C2 03 0A
                A09   c9
```

By reducing the DELAY TIME, the display will cycle at a faster rate. Try the value 06, A0, C0, 10 and 02 for the value A02 and determine which value is the most suitable.

SUMMARY

This is the program you have been entering into the TEC-1 to illuminate the segments. The two variables are located at 801 and 805.

```
800  3E  04
802  D3  01
804  3E  10
806  D3  02
808  76
```

The data at address 801 can range from 01 to 3F and it determines the combination of digits which will be illuminated.

The data at address 805 can range from 01 to FF and it determines the combination of segments which are illuminated.

Expt 8: ILLUMINATING ONE SEGMENT

Aim: To illuminate one segment on the display .

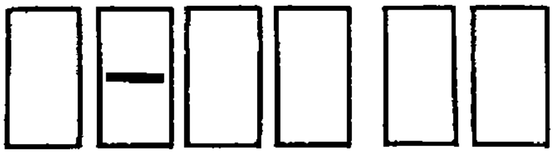
Theory: To master the display you must be able to access (locate) any segment.
The TEC-1 display has 6 digits and each has 8 light emitting diodes. This produces 6x8 = 48 locations.

The aim of this experiment is to illuminate one of these segments.

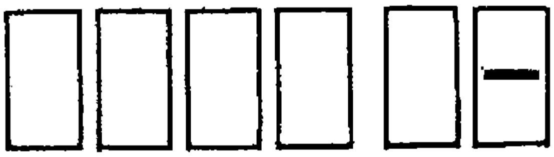
Key the following:

Reset, 3E + 10 + D3 + 01 + 3E + 04 + D3 + 02 + 76 Reset Go.

The display will show one centre segment in the 5th display thus:

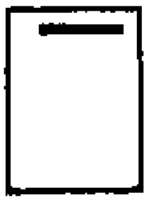


Now key in this program:
Reset 3E + 01 + D3 + 01 + 3E + 04 + D3 + 02 + 76 Reset, Go.
The result is:

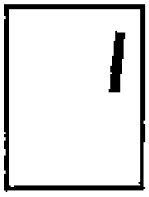


The only difference between the two programs is the byte at 801.
You are now going to make a discovery yourself:
This is how to do it:

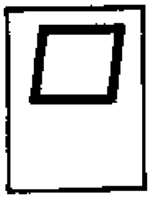
Press Reset.
Press + to 805. Change the data (now showing 04) to 01.
Press Reset, Go.
The segment 'a' illuminates thus:



Press RESET. Press + + to get address location 805.
Change data to 08.
Press RESET, GO.
Only segment b illuminates thus:

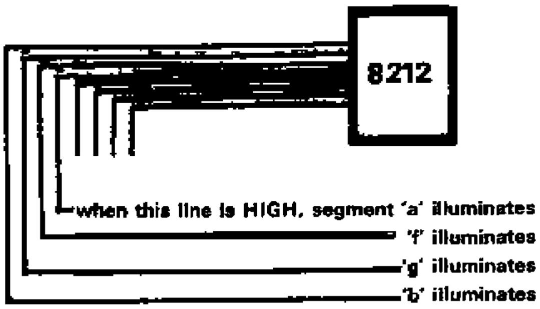


Press RESET. Press + + to 805.
Change data to OF.
Press RESET, GO.
Four segments illuminate thus:



Let's see how this comes about.

The seven-segment display is labelled a-g and the decimal point h. Each segment is illuminated via a binary number sent from the Z80 to the 8212 latches.



So far, so good.

You can see we have activated the 4 lowest value lines and this has produced a rectangle made up of segments a, b, f and g.

Now look at how we programmed a, b, f and g.
The numbers we used were 1, 2, 4, and 8.
Can you see the connection?
It's binary.

The computer converts our keyboard number to a binary number thus:

a = 1
b = 10
c = 100
d = 1000

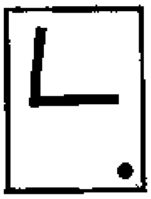
The next segment in the series is 10000, which is 16 in binary terms.

So, let's see what happens when we put 16 into the program at 801.

Key this sequence:

RESET 3E + 16 + D3 + 02 + 3E + 10 + D3 + 01 + 76 RESET GO.

Has something gone wrong? We get this result:

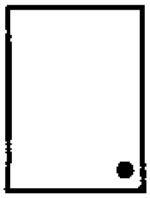


The one fact we have omitted is the TEC-1 is programmed in hexadecimal numbers. This means 16 should be typed as 10 (one, oh or one zero) since the computer counts: 1,2, 3,4,5,6,7,8,9,A,B,C,D,E,F,10. Thus 10 is equal to 10000 in binary.

Put 10 into the program thus:

RESET 3E + 10 + D3 + 02 + 3E + 10 + D3 + 01 + 76. RESET GO.

You will find the decimal point will illuminate thus:



Now add the value of segments a, b, f and g to the value of the decimal point:

This is done by adding 1 + 08 + 02 + 04 + 10 to get 1F.

Place this value in the program:

RESET *3E + 1F + D3 + 02 + 3E + 10 + D3 + 01 + 76* RESET GO.

Success.

We have just added five Hex numbers.

Let us illuminate the next segment in the series. It will have the binary value 100000.
This is equivalent to 32 in binary, but the binary value is not important. It is the hexadecimal value we are interested in. What is 100000 in Hex?
Break the value into 10 0000 so that the four last numbers form a group. The value 10 is 2 in binary and 0000 is zero. Thus the hex value is 2 0 = 20.

Place 20 in the program. Segment 'c' will illuminate.

To combine segment c with a, b, f, and g, we add their Hex values together: *20 + 01 + 08 + 02 + 04 = 2F.*

Insert 2F into the program.
Press RESET, GO.

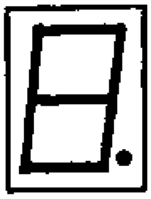
The result is a figure **NINE!**

You can now see that each number and letter in the display is produced by a set of HIGHs on the appropriate lines.

To access the next segment in the series, we must place a HIGH on the 7th line: 0 1 0 0 0 0 0 0. This is 40 in Hex terms and will result in segment 'e' iluminating. When the 8th line is HIGH, segment 'd' will be illuminated.

To illuminate ALL the segments on the display (including the decimal point) we must add the following values: *1 + 2 + 4 + 8 + 10 + 20 + 40 + 80. This is eqial to FF*
(1 + 2 + 4 + 8 + = F) (10 + 20 + 40 + 80 = F0)

Program FF into the sequence at location 801. The result is:



●●●●●●●● ●●●●●●●●●●●●●●●●●

If you programmed our SEGMENT ACCESS routine on P 20 and watched the display carefully, you will have been amazed at the number of recognisable letters and numbers which can be created on a simple 7-segment display.

Our requirement at this stage is to be able to produce some, if not all, of the characters using the information we have learnt in experiment 8.

These are the facts we need:

1. Segment values:

a	*= 01*
b	*= 08*
c	*= 20*
d	*= 80*
e	*= 40*
f	*= 02*
g	*= 04*
h	*= 10*

2. The value of each segment is added in hex form when we need to illuminate two or more segments.

Answers to questions in col 3.

1. *2 = Cd, 3 = Ad, 4 = 2E, 5 = A7, 6 = E 7, 7 = 29, 8 = EF, 9 = 2F.*

2. A = 6F, b = E6, C = C3, d = EC, E = C7, F = 47.

Example: To produce the number **ONE** on a display we must acces segments b and c. This requires the value 28 (in HEX) to be inserted into the following program at location 805:

```
800  3E 01
802  D3 01
804  3E 28
806  D3 02
808  76
```

QUESTION: What would result if **28** were placed at location 801 instead of 805?

Answer: The first and third displays would illuminate with segments "a".

This means you must take care to present the data to the correct output port. In fact every program must be ABSOLUTELY CORRECT as the computer will not be able to correct any of your mistakes.

PROBLEMS:

1. What Hex value must be inserted into the program above to produce the following numbers:

1 = 28
2 = Cd
3 =
4 =
5 =
6 =
7 =
8 =
9 =

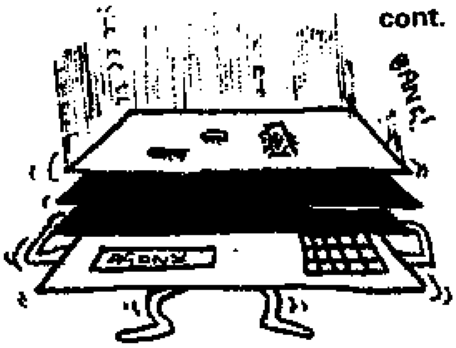
2. Work out the value to illuminate these letters:

A =
b =
C =
d =
E =
F =

3. Draw the result of entering these Hex numbers into the program:

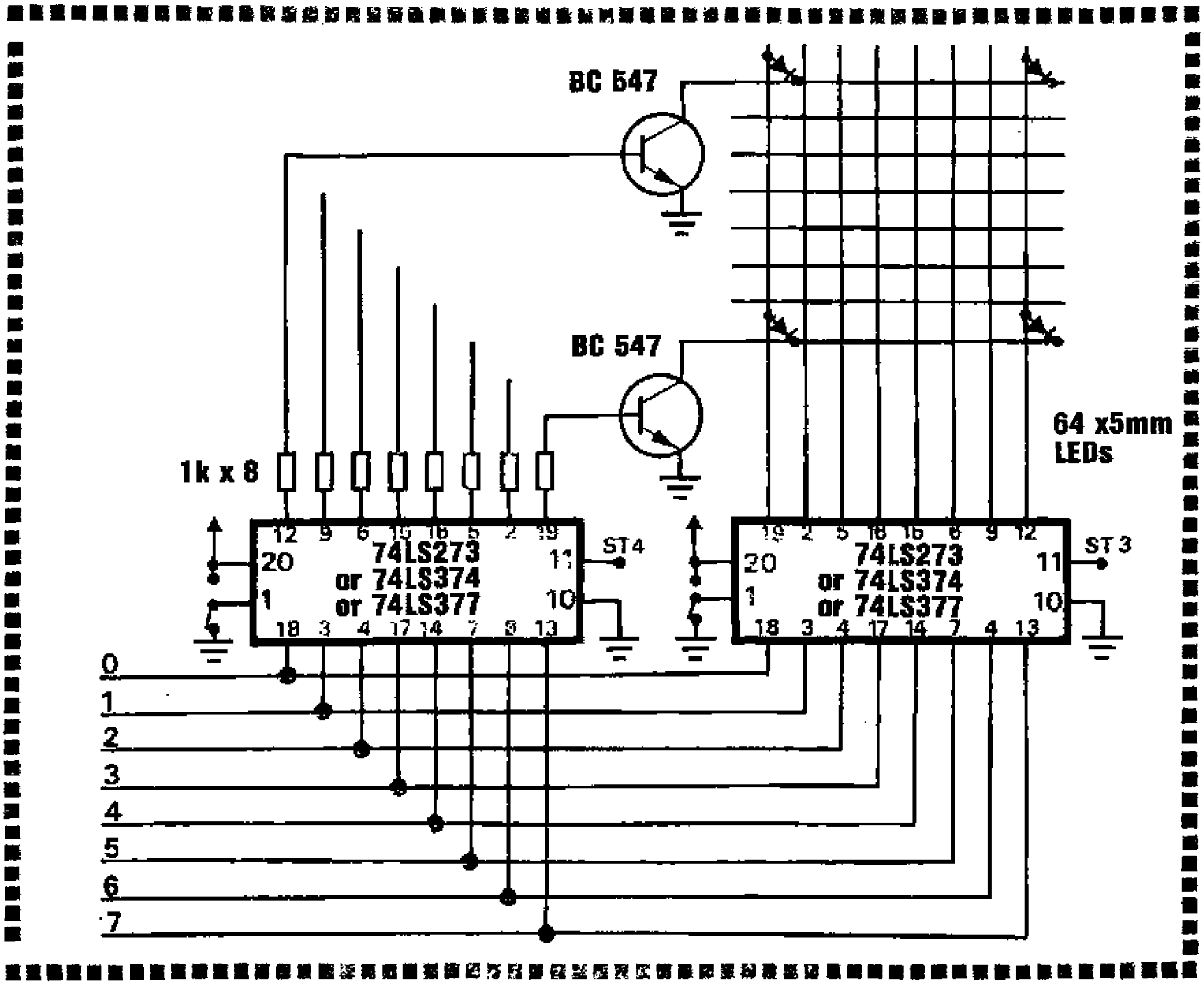
(a) 32 =
(b) 14 =
(c) 49 =
(d) 88 =
(e) A4 =
(f) C3 =
(g) DD =
(h) F1 =
(i) FE =

cont. P.26. . . . .



.. AND SPEECH SYNTHESIZER.....

8x8 MATRIX



This is our first "add-on" for the TEC-1. It is an array of 64 LEDs arranged in a matrix of 8 LEDs by 8 LEDs. Actually it has almost the same number of LEDs as the display on the TEC but in this design the LEDs are arranged in ROWS to create a very interesting display.

The whole concept of the 8 x 8 matrix is to produce the equivalent of a WINDOW ON A VIDEO SCREEN.

Each LED represents one pixel and this will enable us to produce characters, letters and movement equal to 8 pixels by 8 pixels.

This may be only a small fraction of the area of video screen but it is the best place to start. If you can produce effects and movement on a small scale, a full-size VDU screen is only an enlargement of our 'window'.

If you have seen the advertising signs composed of thousands of LEDs or globes on which moving letters and characters are displayed, you will be interested to know the same effect can be produced with this project.

Modules of the 8x8 display can be placed side-by-side to create a long display. The PC board is designed to be cut so that the pattern runs as a continuous display.

At this stage it is not out intention to promote the extended display as it requires a slightly different driving circuit. To achieve a readable brightness with more than 8 columns of LEDs, it is necessary to introduce blocks of columns which are latched or even latching for each column. This will enable each LED to be turned on to full brightness and produce a bright display.

PARTS LIST

8 -	1k ¼watt
8 -	BC 547 transistors
2 -	74LS273 or
2 -	74LS374 or
2 -	74LS377
64 -	5mm red LEDs
2 -	matrix pins
2 -	matrix connectors

30 cm hook-up wire, 12 colours.
30cm tinned copper wire

15cm - Heat-shrink tubing

1 -	24 pin DIP HEADER
1 -	8x8 DISPLAY PC BOARD

In our design, the LEDs are multiplexed and this means they are being turned on for one-eighth of the time during one cycle. The result is a dull display but one which can be read under normal lighting conditions.

We are presenting this project slightly ahead of time so that it will be ready when needed.

There are a number of MACHINE CODE instructions which can only be investigated on a display format and this project is a necessary part to understanding the Z80.

The greatest visual impact for this type of display revolves around programmed lighting and effects such as COCA-COLA signs and some of the background effects at discos and TV shows.

Most of the dazzling effects behind singers and dancers on TV have some form of micro-processor controlled lighting. The effects that can be produced are limitless.

We have chosen LEDs in our design for cheapness and simplicity but they could quite easily be replaced with miniature 6v or 12v globes. The only extra components would be the addition of one extra transistor in each line as an emitter follower. This will enable the extra current to be supplied to the globes.

Our 8x8 display is most effective when flashing blocks of LEDs (or globes) and having them jump from one position to another. In addition, bands of LEDs can be made to pass across or up the screen . These effects are very effective and very simple to produce.

But first you must understand the Machine Code instructions involved and how to include them in a program. This will be our endevour in the latter part of the course in this issue and the time is right to prepare the display project so that it can be plugged into the TEC-1 when the time comes.

Believe me, you will be most impressed with the results.

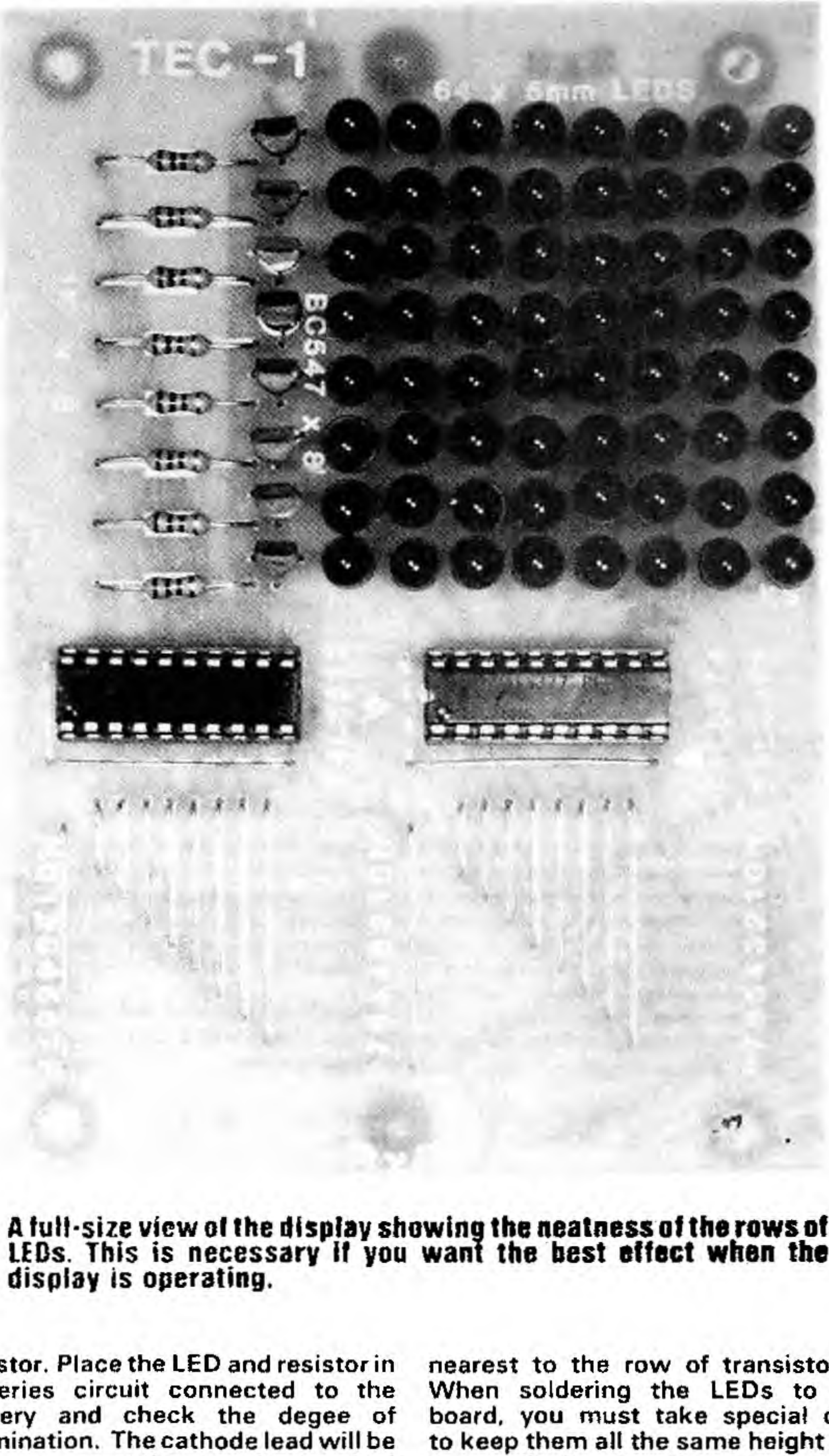
CONSTRUCTION

Before starting any of the construction, it is absolutely essential that you know which lead of the light emitting diode is the cathode. There is only one guaranteed way of determining this. You need a 3v to 6v battery and a 100 ohm or 220ohm resistor. Place the LED and resistor in a series circuit connected to the battery and check the degee of illumination. The cathode lead will be the one nearest the negative terminal of the battery. There are no other sure-fire methods of determining this as some LEDs have their long lead cut differently to the accepted practice.

We have seen some LEDs with the outline (inside the LED) around the opposite way to the general rule. So, it can be quite confusing. You must test each LED or at least a sample from the batch.

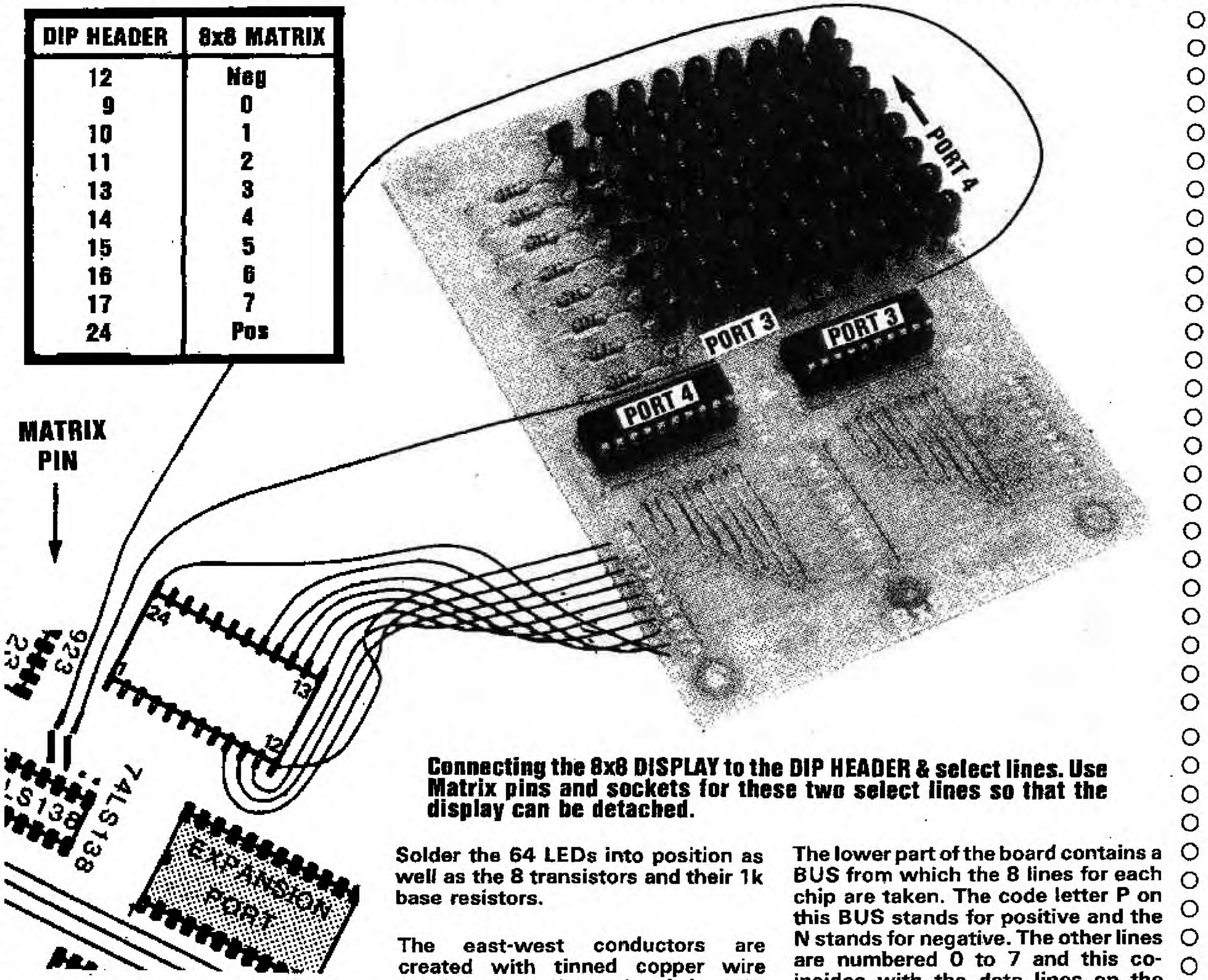
Next you must be certain which way they are to be inserted in the PC board. A mistake will take a very long time to rectify. The cathode lead is nearest to the row of transistors. When soldering the LEDs to the board, you must take special care to keep them all the same height and perpendicular to the board. The neatness of the dispay will depend entirely on how well you position the LEDs.

At first you may think one lead of the LEDs is not connected to the circuit. But this is where we have had to improvise. Multiplexing requires one line of conductors to travel north-south and the matching line to travel east-west. This would normally require a double-sided PC board, but since they are very expensive and difficult to solder, we have opted for the cheaper approach.



A full-size view of the display showing the neatness of the rows of LEDs. This is necessary if you want the best effect when the display is operating.

DIP HEADER	8x8 MATRIX
12	Neg
9	0
10	1
11	2
13	3
14	4
15	5
16	6
17	7
24	Pos



Connecting the 8x8 DISPLAY to the DIP HEADER & select lines. Use Matrix pins and sockets for these two select lines so that the display can be detached.

Solder the 64 LEDs into position as well as the 8 transistors and their 1k base resistors.

The east-west conductors are created with tinned copper wire running along the ends of the LED leads and this connects to the collector of the driver transitor via the PC circuit. The only lead which has to be cut short is the anode lead, to prevent it touching the tinned copper wire.

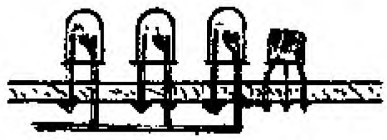


Diagram showing how the 'COMMON' line is created on the underside of the board. Both leads of each LED are soldered to the PC board. But only the ANODE lead is cut short. The CATHODE leads are either joined with a length of tinned copper wire which runs below the board, or each lead is bent over and soldered to the next lead to produce a rigid conductor which runs at right-angles to the copper tracks on the board.

The lower part of the board contains a BUS from which the 8 lines for each chip are taken. The code letter P on this BUS stands for positive and the N stands for negative. The other lines are numbered 0 to 7 and this coincides with the data lines on the latches.

The two output latches can be: 74LS347, or 74LS377 or 74LS273 or a combination of any two. The information on the overlay shows which jumper link must be included for the type of latch you choose.

Eighteen jumper links connect between the data bus and the latches to complete the assembly. The only wiring left is the connecting wires between the DIP HEADER plug and the PC board. This plug is designed to fit into the expansion port socket on the TEC-1.

The 10 lines from the bus on the display board connect to the DIP plug and the two spare lines connect to the chip select outputs near the 74LS138 (near the keyboard encoder). These lines are for ports 3 and 4. Solder two matrix pins to these output holes and use a matrix-pin connector soldered to the hook-up wire to connect to these pins.

DIP HEADER

A DIP HEADER is a plug which has thin pins similar to the pins on an IC, on the underside. On the top are cup-shape (or 'Y' shape) terminals to which you can make a solder connection.

Leads can be soldered to the terminals to create a low-cost adaptor.

It is suggested that heat-shrink tubing be placed over each lead before soldering to the Dip Plug. When all the leads are attached, the sleeving is slid over each terminal so that the conductor is strengthened.

This will prevent fine wiskers of wire shorting from one pin to the other and creating havoc.

MATRIX PINS & SOCKETS

These are the cheapest and best way of connecting a single line to a printed circuit board.

The 8x8 display is now ready for testing and we will give 3 simple programs to test the operation of the LEDs. This will check their illumination, their OFF response and the correct wiring of the data lines and chip select lines.

To check the brightness of the LEDs, insert this program at 800:

```
3E FF
D3 03
3E FF
D3 04
76
Reset
GO
```

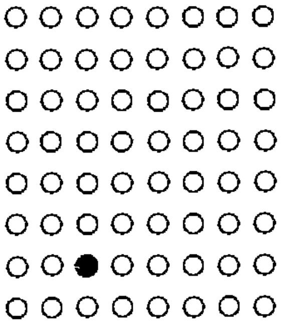
Replace any dull LEDs.

To make sure all the LEDs are extinguished when they are not being accessed, change the program above to:

```
3E 00
D3 03
3E 00
D3 04
76
```

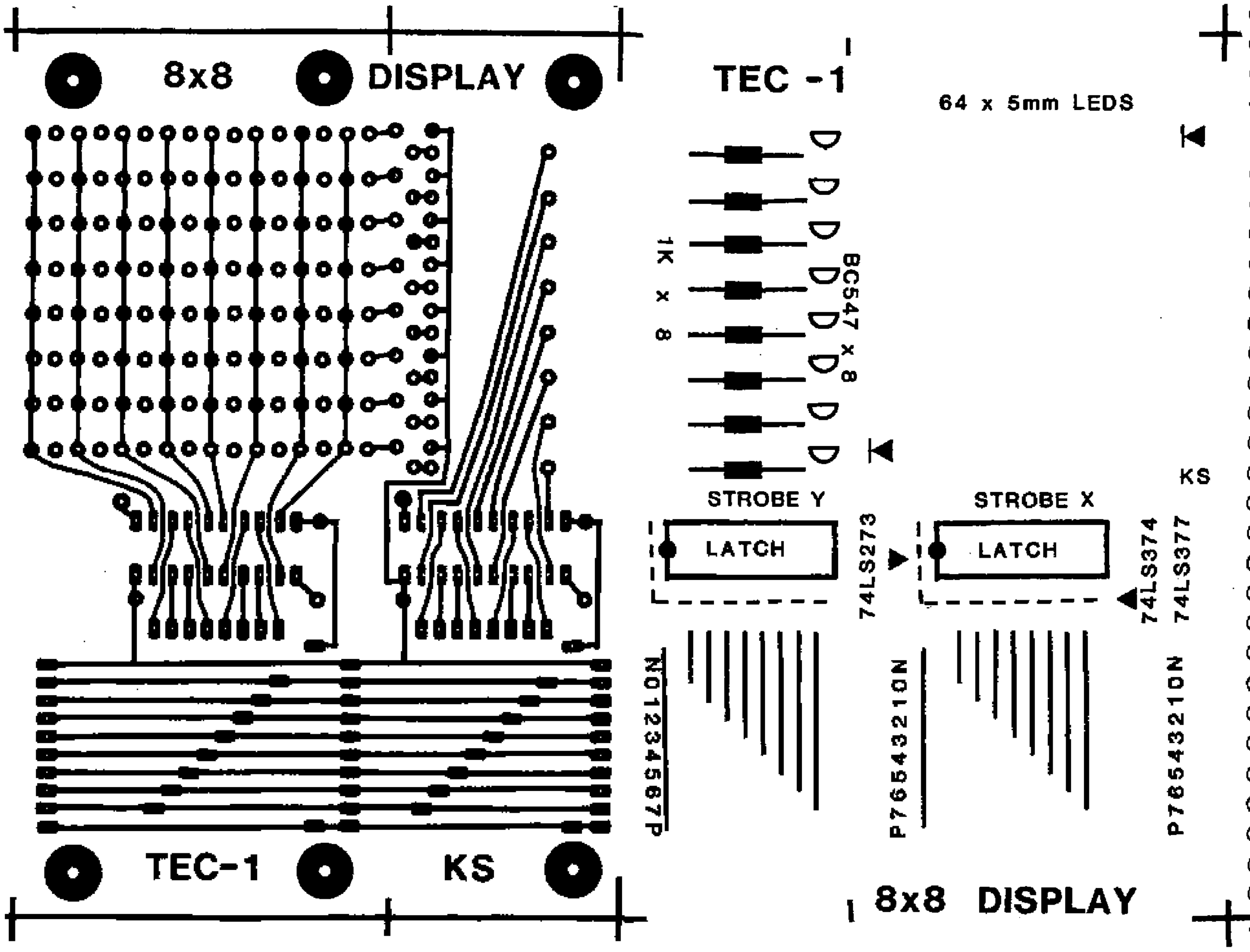
To check the data and chip select lines, insert this program:

```
3E 04
D3 03
3E 02
D3 04
76
Reset
GO
```



The LED which will illuminate is shown in the diagram. If any other LED illuminates, check the select lines.

Now go back to experimenting with the display on the TEC-1. When you get to p.30, you will be able to use the 64 LED display to create some startling effects.



The PC layout and overlay for the 8x8 Matrix. All LEDs face one direction and must be soldered as explained in the text. Apart from the jumpers connecting the BUS line, ONE link is required for each of the latch chips and this must be placed according to the type of chip you use.

....from P. 21.

ILLUMINATING TWO OR MORE DIGITS

More than one display can be illuminated at the same time and this is achieved by changing the value at 801 in the program above.

Example: To fill the six displays with the letter A we program the following:

```
3E 3F
D3 01
3E 6F
D3 02
76
```

Problems:

4. Fill the six displays with the following:
(a) 1's
(b) 5's
(c) b's
(d) E's

5. Place a 'C' at each end of the display.

6. Fill the first and last two displays with the value '8'.

7. Fill only the address displays with 4's.

8. Illuminate only segments a and d on the six displays.

TEC-1 AS A PROGRAMMABLE LOGIC DEVICE

- by John Hardy

The truly unique thing about computers is not that they can preform arithmetic in a twinkling of an eye but it is the way they can be used to simulate any digital (and almost any analog) circuit under the sun.

The microprocessor (Z80) can be likened to a bag of AND and OR gates, a thousand flip flops and tens of thousands of inverters.

You have an 8-bit data bus which means that you can simulate an 8 input NAND (with the aid of a program) and you can output 8 bits of data on this bus.

You are not simply restricted to 8 of this and 8 of that. You can output 16, 32, 64, or even 1024 bits, as long as you break it up into 8-bit groups.

By systematically dealing with 'bits', you can perform a multitude of digital functions.

The TEC-1 is first and foremost a binary computer. While superficially it appears that the computer operates on hexadecimal numbers (9B, 3E, 2C etc.) deep in the heart of the computer binary numbers are the norm.

The problem with binary numbers is their unfamiliarity to humans. Imagine if you wrote a program in binary and made a mistake. It would be very difficult to spot. Take the following example. Can you see the difference between the two?

```
01011010     01011010
10110101     10110101
11001111     11001111
10110101     10110111
11101011     11101011
```

It is possible to check for binary errors but they don't show up easily. Hexadecimal is a short-hand way of representing binary. It is based on breaking up an 8-bit binary number into two 4-bit numbers and converting these into two hexadecimal digits.

Comparing the two sets of numbers above, the difference is quickly spotted when they are converted to hex values as shown below:

```
5A     5A
B5     B5
CF     CF
B5     B7
EB     EB
```

Hex was therefore chosen for use in the TEC-1 but we must never forget that ALL DIGITAL COMPUTERS WORK IN BINARY.

When dealing with computer problems, we should always visualize the inner registers as holding 'bits' and that the computer performs BINARY operations. We then convert to Hex after this. While this might seem awkward, the conversion between Hex and binary can be done quite quickly after a little practice.

- John.

CREATING MOVEMENT!

All the programs up to now have been static.

We will now create some life and movement!

This will introduce a SHIFT or ROTATE function into the program. The rotate function we have selected is located at 80C in the program which follows. This is a two-byte instruction and tells the Z80 to rotate a HIGH bit left circular through the B register. You will understand what we mean by this statement in a few minutes.

This shift operation will take 8 DELAY PERIODS to complete one cycle and will include toggling or clicking the speaker.

RUNNING SEGMENT 'a' ACROSS THE SCREEN

The Program: CB 00-runs segment ←
CB 08 runs segment →

at 0800:

```
LD A,01       800   3E 01
OUT (2),A     802   D3 02
LD B,01       804   06 01
LD A,B        806   78
OUT (1),A     807   D3 01
CALL DELAY    809   CD 00 0A
RLC B         80C   CB 00
JP LOOP       80E   C3 06 08
```

at 0A00:

```
0A00   11 FF FF
       1B
       7B
       B2
       C2 03 0A
       C9
```

This is what the program is saying and instructing the Z80 to do:

The first instruction is to load register A with the value 1.

This is then passed to the SEGMENT PORT latch and this value remains fixed for the whole program.

The remainder of the program concerns port 1, the CATHODE PORT and as the different cathode are accessed, the effect is to run a pattern across the screen.

The next instruction is to load register B with the value 1. This value is then loaded into register A via the instruction 78. The reason for this will be explained in a moment.

The contents of A are now outputted to port 1 with the result that segment 'a' on the lowest priority display will be lit.

We now call a DELAY ROUTINE so that this display will be illuminated for about half a second.

The HIGH bit in register B is then shifted RIGHT. This is performed within register B by the Z80. The program is then incremented to the next instruction and this tells the Z80 to jump to address 806.

The output of the DELAY ROUTINE appears in register A and when this value is zero, the delay routine returns to address 80C.

This means we must use another register to provide our shift routine and in this case we have chosen register B.

Quite a number of variations can be produced with this program by changing the data at some of the locations. These can be carried out after the main program has been entered.

The main program starts at 800 and the delay routine is located at 0A00.

Now try these variations:

1. To run the segment left-to-right, change location **80d** from **00** to **08.**

2. To increase the SPEED of the display: Change A02 from FF to 0F or 06.

3. To run segments 'a' and 'd' across the screen: Change location 801 from 01 to 81.

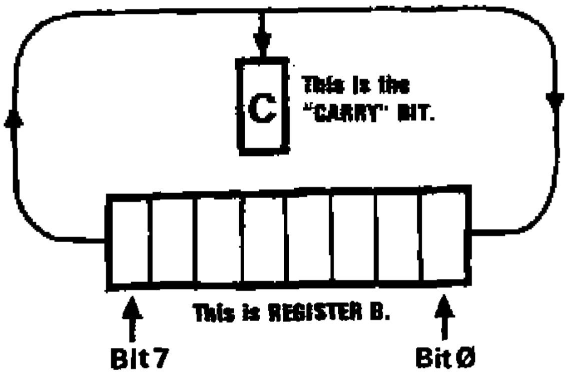
4. To run the number 7 across the screen, change location 801 to **29.**

5. To run the letter A across the screen, change location 801 to **6F.**

The program we have investigated introduced the ROTATE REGISTER B LEFT instruction **CB 00** and ROTATE REGISTER B RIGHT instruction **CB 08.**

RRC B = CB 08

RLC B = ROTATE LEFT CIRCULAR REGISTER B.



The diagram shows register B as 8 boxes. These can be considered as flip flops. The lowest value flip flop is at the right hand end of the row and is labelled Bit zero (Bit 0). This is the Least Significant Bit (LSB).

The Most Significant Bit (MSB) is called Bit 7.

The instruction RLC B has a Machine Code instruction **CB 00** and this causes the most significant bit to emerge from the register and enter it again to become the least significant bit. In this process it does not pass through the CARRY bit but does set the C flag to the original status of the register's most significant bit.

In other words, if the bit in question is a HIGH, the C flag becomes HIGH, if the bit is LOW, the C flag goes LOW.

RLC B = CB 00

RRC B = ROTATE RIGHT CIRCULAR REGISTER B.

This instruction is a reversal of the path shown above. The C flag, however, is altered as above. The ONLY difference between the two instructions is the direction of rotation.

The point to remember in these Machine Code operations is RLC and RRC can be performed on registers A, B, C, D, E, H and/or L and are 8-stage shift operations.

In the next program, on P.28, the instruction which will produce a shift operation across the screen is the instruction **RRA** or **RLA.**

After each shift is performed, the contents of the 'A' register must be 'hidden' or SAVED to prevent it being destroyed.

To do this we must load the contents of register A into another register before calling the DELAY ROUTINE. We could load it into B, C, D or even E register and load it back again when required.

However this will tie up one more of our valuable registers and a better solution is to call upon 2 interesting instructions which load the contents of A into an area of RAM in the 6116 chip.

The code word for saving the contents of a register is called PUSH and recalling it is POP.

The PUSH instruction will take the contents of register A to an area called the STACK.

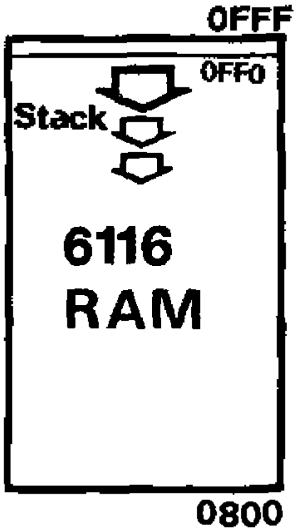
This area is located in the 6116 RAM at address 0FF0. (This is only 16 bytes from the end of this chip's memory and is usually considered to be the unused end of the RAM.)

The highest 16 bytes are used as a scratch-pad area.

The PUSH and POP instructions are similar to stacking plates or trays in a pile. Trays are "pushed" or piled onto the top of the stack and are "popped" or removed from the top.

In the computer the area called the STACK is filled DOWNWARDS. This is an ideal way of using the top part of the RAM and it can be increased in size until it meets the program.

Thus we start with address 0FF0 and work downwards thus: 0FEF, 0FEE, 0FED etc. To keep track of the last address, the Z80 has a register called SP. This is the STACK POINTER register and always points to the byte with the lowest address.



The STACK starts at 0FF0 and heads DOWNWARDS in the 6116 RAM. The data in the EPROM decides this and is 0FE0 when using MON-1B EPROMS.

The Z80 has two instructions for operating on the stack. These are PUSH and POP (or Pull). Both instructions require a register PAIR (such as HL, AF, BC, DE) to be specified as the SOURCE for PUSH and the DESTINATION for POP.

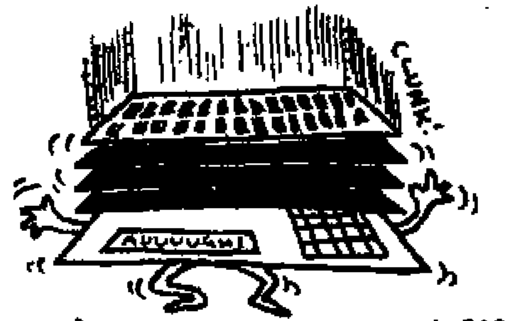
We PUSH new bytes onto the stack and POP bytes off the top.

The Z80 processes this operation TWO BYTES AT A TIME and results in a new byte on the top of the stack with either operation.

The top byte has the lowest address and the memory is filled downwards. The STACK POINT register decreases with a **PUSH** instruction and increases with a **POP** instruction.

Bytes are entered onto the stack, HIGH byte first, then LOW byte. The bytes are removed LOW byte first, then HIGH byte.

In the next program we will investigate the PUSH and POP instructions.



.... AND MEMORY EXPANSION...

To run the 'g' segment from Left-to-Right:

at 800:

```
LD A,04        800   3E 04
OUT (2),A      802   D3 02
LD A,01        804   3E 01
OUT (1),A      806   D3 01
RRA            808   1F
Push AF        809   F5
CALL DELAY     80A   CD 00 09
POP AF         80D   F1
JP 806         80E   C3 06 08
```

at 0900:

```
                       11 FF 06
 THIS IS THE           1B
 DELAY ROUTINE         7B
                       B2
                       C2 03 09
                       C9
```

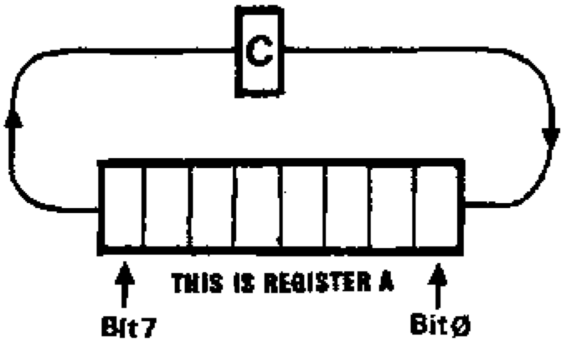
Using the program above, change the address location 808 to 17. This is the machine code instruction for rotating the accumulator left. (RLA), through the carry.

Machine codes covered:

RRA = 1F
RLA = 17

RRA & RLA have nothing to do with moving left or right on the display. They refer to shifting the information through the accumulator via the carry. This means it is a nine-stage shift in which no output is activated when the bit is shifted into the 'carry'. This effect can be seen on the displays when a long delay routine is employed. The illumination travels across the 6 displays, the next output is not used and then the speaker/LED combination is activated. A delay is then noticed before the illumination re-appears on the screen.

The main difference between the two programs is the number of SHIFT STAGES. The first program produced an 8-stage shift while the second prodŭced a 9-stage shift due to the carry bit becoming loaded with each bit from the register as it circulated as shown in the diagram.



To create a FLASHING SEGMENT

Routine at 800:

```
LD A,01        800   3E 01
OUT (2),A      802   D3 02
LD A,01        804   3E 01
OUT (1),A      806   D3 01
CALL DELAY     808   CD 00 09
LD A,00        80B   3E 00
OUT (1),A      80D   D3 01
CALL DELAY     80F   CD 00 09
JP LOOP        812   C3 04 08
```

Delay Routine at 0900:

```
               900   11 FF 07
               903   1B
               904   7B
               905   B2
               906   C2 03 09
               909   C9
```

The exact operation of the delay routine is not important at this stage. It is enough to know that it creates a delay of length determined by the number loaded into register-pair DE. If this number is 01 00, the delay will be only a few microseconds. The first byte refers to register E and this is the lower register while the second byte is the higher register and has the greater effect on the delay.

Try putting different values into location 902 to vary the length of the delay. A value such as **02** will increase the flash-rate while **FF** will create the slowest flashing.

The format of the main routine is very simple. It is an ENDLESS LOOP which means it executes part of the program over and over again.

The 'BIT' patterns for the segments to be lit are loaded into the segment register (port 2). Cathode 1 is then turned on and the delay routine is called.

The cathode register is then cleared and the delay routine is called again.

This creates the OFF cycle.

The program then jumps back to address 804 where it is instructed to turn on cathode 1. This causes segment 'a' to come on once again.

You can flash segment 'g' by loading **04** into the program at 802 thus:

3E 04
D3 02
etc...

Create flashing numbers and letters in the display by inserting the appropriate hex numbers as discovered in questions 1 & 2 on P 21.

You can also use this program to alternate from one number or letter to another. This is achieved by the second letter taking the place of the blanking routine in the program above.
Insert the value **28** at location 80C and run the program. What happens?

The segment 'a' alternates between one display and two other displays. Turn the speed of the computer down to observe this. But this is not what we wanted. We want different segments of the same display to be turned on. We have forgotten to change location **80E** from **01** to **02.**

Run the program and note segment 'a' changes once to a figure '1' and appears to be stuck on this figure.

There is a second fault in the program. Only the second part of it is being cycled.

Change location **813** to **00.** The program will now alternate between the 'a' segment and the figure '1'.

This is the introduction to simple cartooning on the screen. Try changing locations **801** and **80C** to get some interesting effects.

RUNNING AROUND THE DISPLAY

To run a single illuminated segment around the display takes a considerable amount of programming. There are a number of ways of doing this and we will use a program which uses some of the features we have covered so far.

Basically what we are doing is defining our start co-ordinates, shifting a 'bit' six places to the left and halting.

The next part of the program loads the co-ordinates of the side segment (at the top of the display) and then the lower end segment is lit.

We then define the co-ordinate on the bottom row and run the illuminated LED across the bottom of the display.

Finally we define the bottom side segment and the top side segment to arrive back at the starting point.

This will create an endless run around the display.

We will produce this program in 4 stages and check its operation at each stage.

"AROUND THE DISPLAY"

```
LD A,01          800   3E 01
OUT (2),A        802   D3 02
LD C,06          804   0E 06
LD A,01          806   3E 01
OUT (1),A        808   D3 01
LD B,A           80A   47
CALL DELAY       80B   CD 00 09
LD A,B           80E   78
RLC A            80F   CB 07
DEC C            811   0D
JP NZ,LOOP 1     812   C2 08 08
HALT             815   76
```

Push RESET, GO.

If the LED runs across the top of the display and HALTS, everything is working.

Press RESET,ADdress 815 +

Now insert the following program so that the **HALT** instruction is written over and is removed from the program.

```
LD A,02          815   3E 02
OUT (02),A       817   D3 02
CALL DELAY       819   CD 00 09
LD A,40          81C   3E 40
OUT (02),A       81E   D3 02
CALL DELAY       820   CD 00 09
HALT             823   76
```

Check the program at this stage by running it. If the LED travels across the top and down one side, it is working. Over-type 3E at address 823 and continue with the 3rd stage:

```
LD A,80          823   3E 80
OUT (02),A       825   D3 02
LD C,06          827   0E 06
LD A,20          829   3E 20
OUT (01),A       92B   D3 01
LD B,A           82D   47
CALL DELAY       82E   CD 00 09
LD A,B           831   78
RRC A            832   CB 0F
DEC C            834   0D
JP NZ,LOOP 2:    835   C2 2B 08
HALT             838   76
```

If all is ok, type the last part of the program:

```
LD A,20          838   3E 20
OUT (02),A       83A   D3 02
CALL DELAY       83C   CD 00 09
LD A,08          83F   3E 08
OUT (02),A       841   D3 02
CALL DELAY       843   CD 00 09
JP START         846   C3 00 08
```

Delay Routine at 0900:

```
11 FF 06
1B
7B
B2
C2 03 09
C9
```

Don't forget to add the DELAY ROUTINE.

The overall speed of the sequence can be varied by adjusting the SPEED control on the TEC-1.

More programs for the TEC-1 using its own display will be presented in the next issue.

MON-1A

Some of the latest kits of the TEC-1 have included a monitor EPROM marked Mon 1A. This EPROM will work in both the TEC-1 and TEC 1A as both are software compatible with each other.

The difference between Mon 1 and Mon 1A is a small additional routine at 05B0. This program was originally designed for use with music synthesisers but can also be used for a number of other applications.

The routine is a simple sequencer. It reads the data stored in RAM and deposits it at a fixed rate into the output latches.

The overall speed of the sequence can be varied by adjusting the SPEED control on the TEC.

There are two sequencing functions being performed in this program, one depositing information to its relavent latch (04) at TWICE the speed of the other (03).

The two sequences are synchronised and one output falls mid-way between the other. However the sequence-length is independent.

The end of the sequence is marked by placing an FF after the last piece of data. The sequence will then reset itself to the beginning. The other sequence will continue unaffected until it also hits an FF.

Because FF has been used to indicate the end of the sequence, you cannot use FF as a piece of data. In our application, this presents no problem, but when used with the relay board, it means all 8 relays cannot be activated at the one time.

We can go as high as FE without upsetting the program and this will turn on 7 relays, but not the lowest priority relay.

The slower sequence outputs to latch 03 and reads its data from address 0800 until it encounters FF and then resets.

The faster sequence outputs to latch 04 and reads its data from address 0B00 until it encounters FF and then it resets.

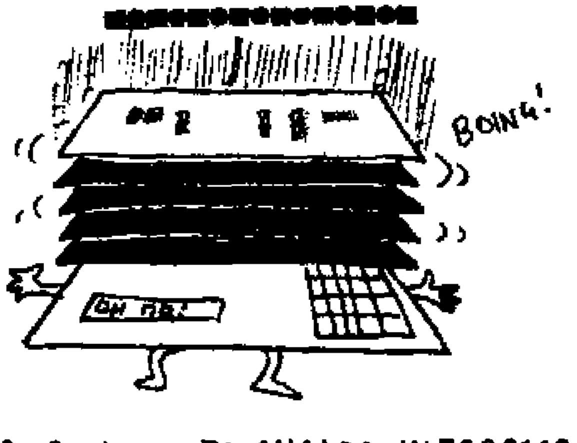
It should be noted that high memory is used by the Z80 to store its stack and thus memory above 0F00 should not be used.

A disassembly and Hex listing for this routine is given below:

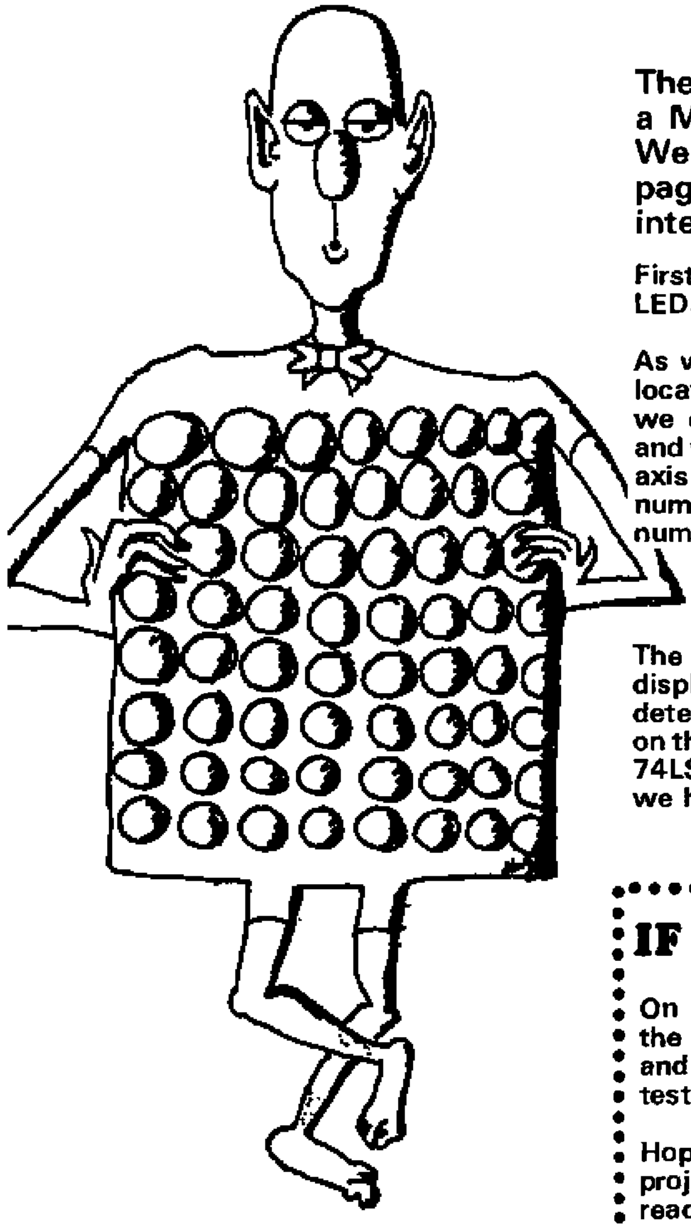
```
05B0    21    LD HL,0800
05B3    11    LD DE,0B00
05B6    7E    LD A,(HL)
05B7    FE    CP FF
05B9    C2    JPNZ,05C2
05BC    21    LD HL,0800
05BF    C3    JP 05B6
05C2    D3    OUT (03),A
05C4    1A    LD A,(DE)
05C5    FE    CP FF
05C7    C2    JPNZ,05D0
05CA    11    LD DE,0B00
05CD    C3    JP 05C4
05D0    D3    OUT (04),A
05D2    CD    CALL 05E1
05D5    13    INC DE
05D6    1A    LD A,(DE)
05D7    D3    OUT (04),A
05D9    CD    CALL 05E1
05DC    13    INC DE
05DD    23    INC HL
05DE    C3    JP 05B6
05E1    01    LD BC,03FF
05E4    0B    DEC BC
05E5    78    LD A,B
05E6    B1    OR C
05E7    C2    JPNZ, 05E4
05EA    C9    RET
```

Hex Listing:

```
05B0    21  00  08  11
05B4    00  0B  7E  FE
05B8    FF  C2  C2  05
05BC    21  00  08  C3
05C0    B6  05  D3  03
05C4    1A  FE  FF  C2
05C8    D0  05  11  00
05CC    0B  C3  C4  05
05D0    D3  04  CD  E1
05D4    05  13  1A  D3
05D8    04  CD  E1  05
05DC    13  23  C3  B6
05E0    05  01  FF  03
05E4    0B  78  B1  C2
05E8    E4  05  C9
```



.... AND DIGITAL TO ANALOG INTERFACE



8x8 MATRIX

The possibilities and effects on a MATRIX layout are infinite. We will allocate the next few pages to showing some interesting visual effects.

Firstly we will show how each of the LEDs is accessed.

As with any matrixing system, each location has a set of co-ordinates. If we compare our display with the x and y axes in geometry, we find the x-axis has the lower output port number and the y-axis the higher number.

The output ports allocated to this display are 3 and 4 and this is determined by the chip access lines on the main board. Each line from the 74LS138 has a particular number and we have selected lines 3 and 4.

On the display board, each of the LEDs has a particular co-ordinate value which must be in the form of a Hex number. Each successive row or column has a hex number which is DOUBLE the previous number. The following diagram shows this:

The lowest priority LED has the value 01, 01 and the highest LED 80, 80. The value of each LED between these limits is also given, as well as the value for 4 individual LEDs, as a guide.

Placing these hex values into a simple program will illuminate any particular LED on the screen.

Here is the general program:

```
3E Hex value: ↔
D3 03
3E Hex value: ↕
D3 04
76
```

IF THE 8x8 MATRIX DOESN'T WORK

On P.28 of this issue we described the construction of the 8x8 matrix and presented 3 short programs to test the LEDs in the display.

Hopefully you will have put the project together by now and will be ready to explore its capabilities.

The main difference between this project and the display on the TEC-1 is not so much the number of LEDs, but the way in which they are arranged.

We have created a regular matrix of 8 LEDs by 8 LEDs and this produces a screen very similar to a window on a video display.

The most common fault will be one or two of the LEDs failing to illuminate when the whole screen is accessed.

If this is the case, or if one is dull, the fault will be a damaged LED. LEDs are temperature sensitive, and excess heat when soldering will damage them. On the other hand, it may be a poor quality LED in the batch.

If any of the LEDs are particularly dull, they should be replaced at this stage to produce a good display.

Here are some of the possible faults and their remedies:

If a row or column fails to light, the fault will be in one of the output lines of a latch or one of the driver transistors. Make sure it is not a dry joint or a missing link and then check the orientation of the transistors and the LEDs.

If a row and column is failing to illuminate, the fault will lie in a shorted LED at the intersection.

Remove the LED and turn on the remainder of the screen. If the remainder of the LEDs come on, the fault is a short.

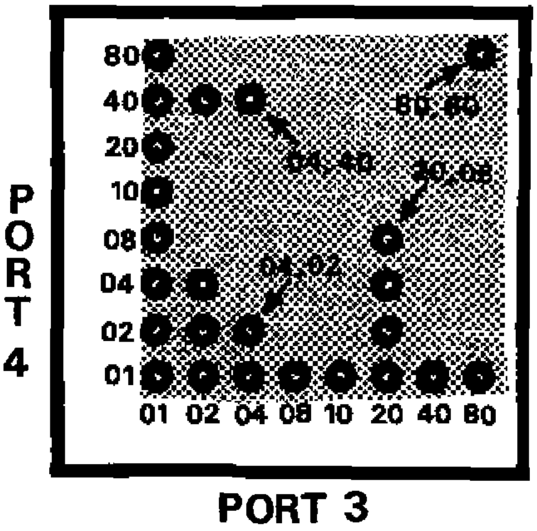
The only other fault we have seen is one row glowing brighter than the rest. This can be due to one of the transistors shorting between collector and emitter. A short to base may cause the row to be extinguished.

If all these suggestions fail to locate the fault, turn the TEC-1 off and re-program the set of instructions. Check to see that you have loaded FF into both port 3 and port 4.

Check both ends of the connecting leads and make sure they are connected correctly to the pins on the dip plug.

Since the expansion port socket is effectively in parallel with the other memory chips, it is very unlikely the the PC tracks will have shorts between them.

This means you should look mainly on the display board itself.



Diag 1: The ports and their Hex values.

If we take a particular case and load the co-ordinates 04, 02 into the program:

```
3E 04
D3 03
3E 02
D3 04
76
```

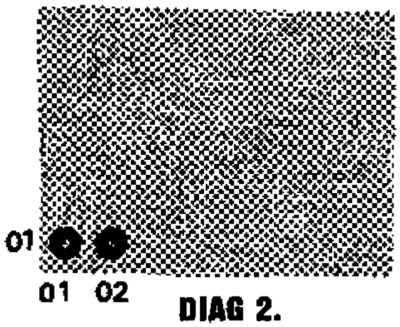
As you type the program, this is what you should be saying: Load the accumulator with 4, output it to port 3. Load the accumulator with 2 and output it to port 4. Halt.

Problems:

Illuminate 3 of the other LEDs by inserting the following data into the program:
1: 04,40
2: 20,08
3: 80,80.

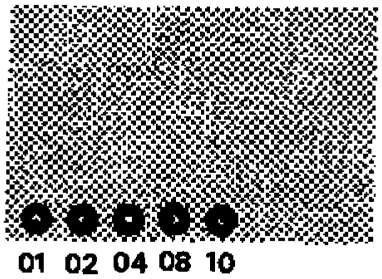
TWO OR MORE LEDs

More than one LED can be illuminated in any row or column by adding the Hex value of each LED. We will start with the simplest case but absolutely any LEDs in any row or column can be illuminated.



DIAG 2.

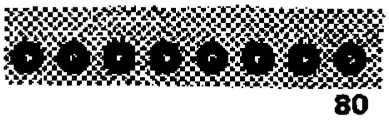
In diagram 2, two LEDs are shown illuminated. These have co-ordinates 01,01 and 01,02. To turn on both of these LEDs we add the bottom Hex numbers. The result is 03. Place this value into the program .at address 801.



Diag 3.

Diagram 3 shows five LEDs illuminated. Add the Hex numbers together and insert it into the program and see if you are correct.

Did you get 1F?



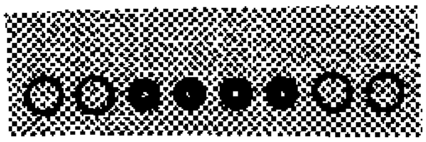
Diag 4.

The fourth diagram shows ALL the LEDs on the bottom row illuminated. What value must be placed in the program at 801 to access these LEDs?

The answer if FF. This is obtained by adding 01, 02, 04, 08 10, 20, 40, 80, this gives: 0F + F0
= *FF*

Problem:

Load the program with a hex value which will illuminate the four LEDs in the centre of the bottom row:

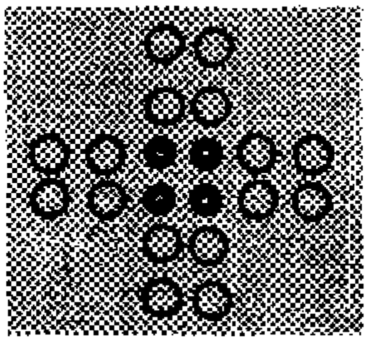


Diag 5.

Firstly look up which values are allocated to each LED then add these values.

Place this into the program and observe the result. You will be correct with the value 3C.

The program for accessing the LEDs in the 8X8 Marix is identical to that for the display on the TEC-1. The only difference is in appearance. A regular array makes the effect more dramatic and the overall possibilities are much greater.

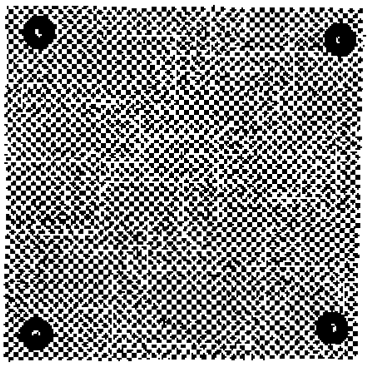


Diag 6.

To turn on the four centre LEDs we must insert the value 08 + 10 into the program for both outputs.

Problem:

What value must be inserted into the program to illuminate the four corner LEDs?

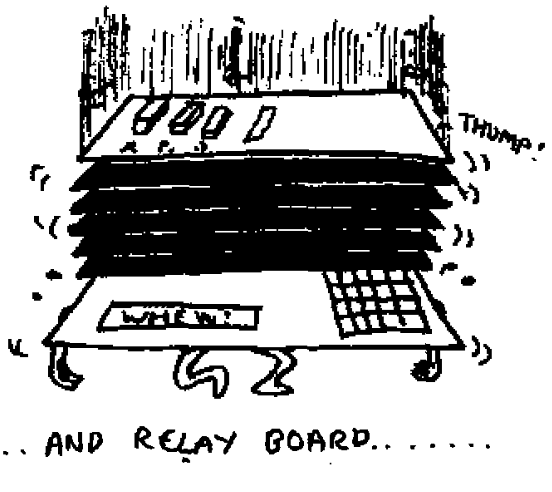


Diag 7.

It is now your turn to illuminate a LED. Select a LED on the matrix and mark it with a pen. Determine its co-ordinates and put them into the program. Execute the program and see it the marked LED comes on. Try two more of these routines and confirm the program by illuminating the LED.

Now illuminate two or three LEDs in any row or column by adding the relevant Hex values together and observe the LEDs on the display.

With this simple program it is not possible to illuminate any combination of LEDs on the whole screen because we are using the outputs in the static mode. To illustrate this, try to illuminate one column and one row at the same time. You know the Hex value for a complete row is FF. Place this into the program and see what happens. The result is a completely-filled screen. The closest effect to producing an intersecting row and column is a non-illuminated row and column produced by inserting a value such as EF into the program.



PROBLEMS:

Demonstrate your understanding of addressing the matrix display by solving the following:

1. Illuminate the whole screen.
2. Illuminate the whole screen except for the outer row and column of LEDs.
3. Illuminate the four centre LEDs as well as the next row and column on each side.
4. Illuminate any quarter of the display.
5. Leave the two centre rows and columns non-illuminated.
6. Place FF in port 3 and 00 in port 4. What appears on the screen? Why?

MAKING A FLASHING LED

We know the general formula for turning on a LED on the matrix:

```
3E (data) ← Single
D3 03       Byte.
3E (data)
D3 04
76
```

"FLASHING LED"

To FLASH the LOWEST priority LED we insert data into the program as follows:

```
LD A,01         800  3E 01
OUT (3),A       802  D3 03
LD A,01         804  3E 01
OUT (4),A       806  D3 04
CALL DELAY      808  CD 00 0A
LD A,00         80B  3E 00
OUT (3),A       80D  D3 03
LD A,00         80F  3E 00
OUT (4),A       811  D3 04
CALL DELAY      813  CD 00 0A
JP 0800         816  C3 00 08
```

DELAY ROUTINE AT 0A00:

```
11 FF 06
1B
7B
B2
C2 03 0A
C9
```

Press **RESET, GO** and the lowest LED will blink ON and OFF. The program is basically loading data into ports 3 and 4 then calling the delay so that the information will be displayed on the screen for a short period of time. The output latches are then loaded with 00 data which will produce a non-illuminated display and the delay routine is called. This produces the 'OFF' period. The program is cycled in an endless loop to produce the flashing.

With this program it is easy to flash any number of LEDs or even the whole screen.

TO BLINK THE WHOLE SCREEN

To blink the whole screen, change the data at addresses 801 and 805 to FF. This has the effect of filling the screen for one delay period and then non-illuminating the screen for one delay period.

To alternately blink the left-hand side of the screen and then the right-hand side:
Insert the following data:

at address:

801 insert FF
805 insert 0F
80C insert FF
810 insert F0

You can make the flash move in the up/down motion by programming:

801 insert 0F
805 insert FF
80C insert F0
810 insert FF

An overlap can be created by inserting the following data:

801 insert 1F
805 insert FF
80C insert F8
810 insert FF

You will notice the two centre rows remain ON for the whole period of time as shown by this table:

```
      ┌ 80
      │ 40
TOP   │ 20
      │ 10 ┐
      └  8 │
         4 │ BOTTOM
         2 │
         1 ┘
```

An interlocking effect can be created by programming the following:

801 insert AA
805 insert FF
80C insert 55
810 insert FF

To make a block of 4 LEDs jump diagonally and back again, the following information is inserted into the program:

change 801 to 0F
change 805 to 0F
change 80C to F0
change 810 to F0

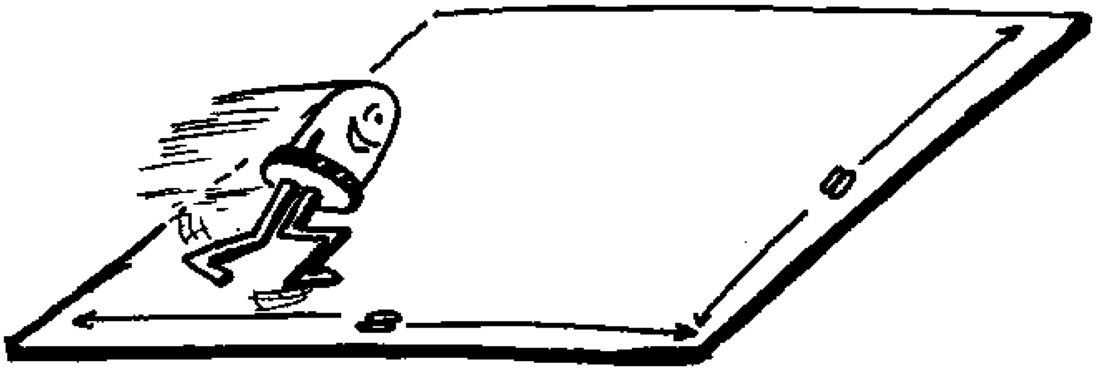
You can experiment with the length of the delay to produce a faster or slower flash rate.

For a slow flash insert: 11 FF 0A
Medium flash: **11 FF 08**
fast flash: **11 FF 06**

TO RUN A SINGLE LED ACROSS THE DISPLAY

This program will run a single LED across the bottom of the display, from left to right and HALT.

```
LD A,01         800  3E 01
OUT (4),A       802  D3 04
LD C,08         804  OE 08
LD A,01         806  3E 01
OUT (3),A       808  D3 03
LD B,A          809  47
CALL DELAY      80A  CD 00 0C
LD A,B          80D  78
RLC A           80E  CB 07
DEC C           810  OD
JP NZ LOOP      812  C2 08 08
HALT            815  76
```



"RUNNING LED ON AN 8X8 MATRIX"

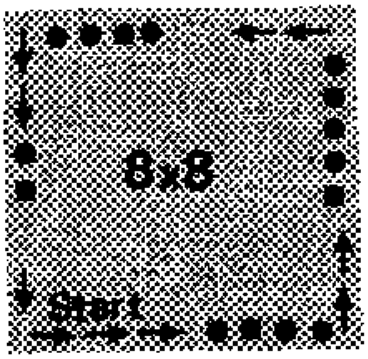
To regulate the speed at which the LED crosses the display, we need a delay routine. (Exactly the same as the previous delay routine.)

```
Delay routine        11 FF 06
at OCOO:             1B
                     7B
                     B2
                     C2 03 0C
                     C9
```

For a full column to move across the screen, change the data at **801** to **FF.**

To create a REPEAT, change the Halt at **815** to **C3 00 08.**

To make a single LED run around the perimeter of the display, we must create a program for each of the four sides. The program above is suitable for the first side and three more programs are needed. At location **815** we remove the **HALT** function (or the return function) and add the following:
Press RESET,ADdress 0815, +. *Now continue:*



```
LD A,80          815   3E 80
OUT (4),A        817   D3 04
LD C,07          819   OE 07
LD A,02          81B   3E 02
OUT (3),A        81D   D3 03
LD B,A           81F   47
CALL DELAY       820   CD 00 0C
LD A,B           823   78
RLC A            824   CB 07
DEC C            826   OD
JP NZ LOOP       827   C2 1D 08
HALT             82A   76
```

Press RESET, GO. The LED will travel along 2 sides of the display and Halt.

Program the third side as follows:
Press RESET, ADdress, 082A, +
Add the following:

```
LD A,80          82A   3E 80
OUT (3),A        82C   D3 03
LD C,07          82E   OE 07
LD A,40          830   3E 40
OUT (4),A        832   D3 04
LD B,A           834   47
CALL DELAY       835   CD 00 0C
LAD A,B          838   78
RRC A            839   CB 0F
DEC C            83B   OD
JP NZ LOOP       83C   C2 32 08
HALT             83F   76
```

Press RESET,GO and watch the LED travel the 3 sides of the display. If everything is correct, program the last side as follows:

```
LD A,01          83F   3E 01
OUT (4),A        841   D3 04
LD C,07          843   OE 07
LD A,40          845   3E 40
OUT (3),A        847   D3 03
LD B,A           849   47
CALL DELAY       84A   CD 00 0C
LAD A,B          94D   78
RRC A            84E   CB 0F
DEC C            850   OD
JP NZ LOOP       851   C2 47 08
JP 0800          854   C3 00 08
```

Two adjustments must be made to the first section of the program to eliminate the double exposure on the lowest priority LED. Change location **805** to **07** and **807** to **02.** The led will now travel evenly around the display.

To view the effect, press RESET, GO.

The previous program is long because each direction of travel must include the commencement location.

The next program is just as interesting but much shorter because it generates its own new set of values at the end of each cycle via the **INC H** operation.

It moves a LED across the screen and increases its value on each pass.

```
LD A,01          800   3E 01
LD H,01          802   26 01
LD A,H           804   7C
OUT (3),A        805   D3 03
LD C,08          807   OE 08
LD A,01          809   3E 01
OUT (4),A        80B   D3 04
LD B,A           80D   47
CALL DELAY       80E   CD 00 0C
LD A,B           811   78
RLC A            812   CB 07
DEC C            814   OD
JP NZ LOOP       815   C2 OB 08
INC H            818   24
JP 804           819   C3 04 08
```

```
At 0C00:
                       11 FF 06
                       1B
                       7B
                       B2
                       C2 03 0C
                       C9
```

At the beginning of the previous routine, the first instruction **LD A,01** is not needed as the second and third instruction performs this task. Your requirement is to re-write the whole listing, beginning at 0800, with this instruction removed. This requires the instruction at **819** to be changed to **C3 02 08** as all the instructions have been shifted two locations.

Run the new listing and make sure it works.

Increase the speed of the program by decreasing location **0C02** to **03.**

How can we make it run slower?

Ans: Insert FF into location 0C02 and reduce the CLOCK speed on the computer.

MAKING THE LEDS RUN FROM RIGHT-TO-LEFT

We can add an instruction to this program to make the LEDs run from right-to-left.

The two locations to change are:

change **809** to **3E 80**
change **812** to **CB OF**

Try these variations:

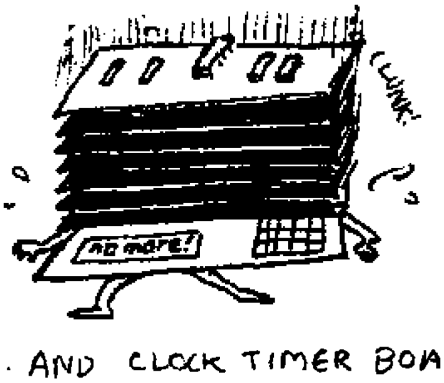
change **802** to **26 FF**
change **818** to **25**

To make the LEDs run from left to right and back again or from top to bottom and down again, requires the combining of a SHIFT-LEFT program with a SHIFT-RIGHT program.

Key in the following listing and push RESET, GO. Watch the effect.

Don't forget the delay routine at OCOO.

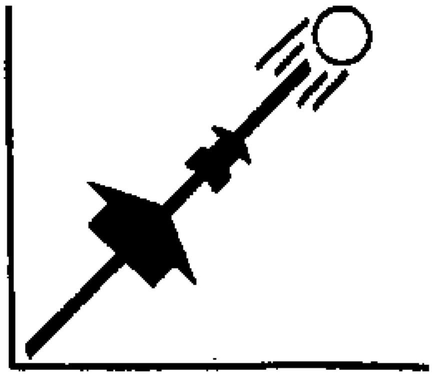
```
LD H,01          800   26 01
LD A,H           802   7C
OUT (3),A        803   D3 03
LD C,08          805   OE 08
LD A,01          807   3E 01
OUT (4),A        809   D3 04
LD B,A           80B   47
CALL DELAY       80C   CD 00 OC
LD A,B           80F   78
RLC A            810   CB 07
DEC C            812   OD
JP NZ LOOP       813   C2 09 08
LD C,08          816   OE 08
LD A,80          818   3E 80
OUT(4),A         81A   D3/04
LD B,A           81C   47
CALL DELAY       81D   CD 00 OC
LD A,B           820   78
RRC A            821   CB OF
DEC C            823   OD
JP NZ LOOP       824   C2 1A 08
INC H            827   24
JP 0802          828   C3 02 08
```



.... AND CLOCK TIMER BOARD....

"TAKE-OFF!"

This program produces a single LED which runs diagonally across the display. The angle at which the LED moves is the result of increasing the value of both outputs AT THE SAME TIME. This can lead to some interesting effects.



At 800:

```
LD A,01        800   3E 01
OUT (3),A      802   D3 03
OUT (4),A      804   D3 04
RRA            806   1F
PUSH AF        807   F5
CALL DELAY     808   CD 00 09
POP AF         80B   F1
JP 802         80C   C3 02 08
```

At 900:

```
                     11 OF 00
                     1B
                     7A
                     B3
                     C2 03 09
                     C9
```

At address **806** the instruction **17** will cause the LED to travel up the screen. If we insert the instruction **1F** the LED will travel down the screen.

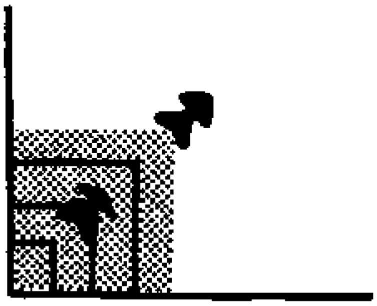
At location **801** insert the value 90. Try both directions of travel and watch the different effects.

Both ROTATE instructions **17** & **1F** cause the 'bits' in the accumulator to rotate through the 'carry' and this creates a 'hole' or zero in the output. This forms the non-illuminated band which passes across the screen.

At location 801, the value 01 can be replaced by 02, 4, 8, 10, 20, 40 or 80. These will not alter the effect on the screen as they will merely define the starting point for the program and it will run through its cycle in the normal manner.

"FAN - OUT"

This program is almost identical to the previous. But by adding one new instruction, we can change the effect on the display to produce a completely different effect.



```
LD A,01        3E 01
OUT (3),A      D3 03
OUT (4),A      D3 04
RLA            07
PUSH AF        F5
CALL DELAY     CD 00 09
POPAF          F1
INC A          3C
JP 802         C3 02 08
```

Delay at 900:

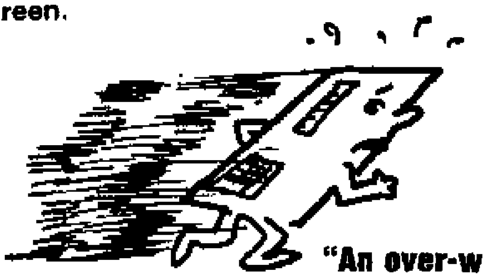
```
               11 FF 06
               1B
               7A
               B2
               C2 03 09
               C9
```

The new instruction is **INC A**. It makes the least significant bit HIGH. The result is to produce an increasing row of LEDs. This is how it happens:

Initially a HIGH is programmed as the Least Significant Bit. The operation **RLA** transfers this HIGH to the second location. When **INC A** is executed, a HIGH is placed in the lowest position. This gives two HIGHs in the register. These two HIGHs shift up the register when **RLA** is executed. **INCA** produces another HIGH in the lowest position and thus the whole register is gradually filled.

The program is producing its own NEW set of data each time the listing is cycled.

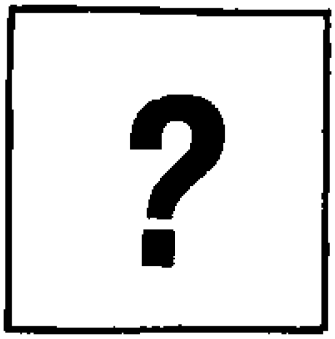
The final result is most impressive. The display fans out from the lower left-hand corner to fill the entire screen.



"An over-worked TEC-1"

OUR MYSTERY EFFECT

I call this our mystery effect as I have forgotten how it appears on the screen. All I remember is producing it. It took about an hour or so to get the program together and I will leave it for you to type into the TEC-1 and see what appears.



Here is the listing:

```
LD C,40        800   OE 40
LD A,01        802   3E 01
OUT (3),A      804   D3 03
LD A,01        806   3E 01
OUT (4),A      808   D3 04
RLCA           80A   07
CALL 900       80B   CD 00 09
DEC C          80E   OD
JP NZ 808      80F   C2 08 08
LD C,20        812   OE 20
LD A,01        814   3E 01
OUT (4),A      816   D3 04
LD A,01        818   3E 01
OUT (3),A      81A   D3 03
RLCA           81C   07
Call 900       81D   CD 00 09
DEC C          820   OD
JP NZ 824      821   C2 1A 08
LD C,40        824   OE 40
LD A,01        826   3E 01
OUT (3),A      828   D3 03
OUT (4),A      82A   D3 04
RLCA           82C   07
CALL 900       82D   CD 00 09
DEC C          830   OD
JP NZ 832      831   C2 28 08
JP 0800        834   C3 00 08
```

At 0900:

```
               900   F5
               901   CD 00 0C
               904   F1
               905   3C
               906   CB 47
               908   CA OC 09
               90B   C9
               90C   CD 00 0C
               90F   CD 00 0C
               912   C3 00 09
```

Delay at 0C00:

```
                     11 FF 06
                     1B
                     7A
                     B2
                     C2 03 0C
                     C9
```

USING THE KEYBOARD

The next area of learning is to include a keyboard input for the 8x8 matrix.

Whenever the HALT function is placed in a program, the Z80 stops the program and waits for an input via the interrupt line.

In our case, this comes from the keyboard and the non-maskable interrupt line is activated to allow the Z80 to accept the data from the keyboard encoder via the data bus.

This data is loaded into the accumulator and compared with a value in the program. If the two values are the same, the output is zero and the program advances.

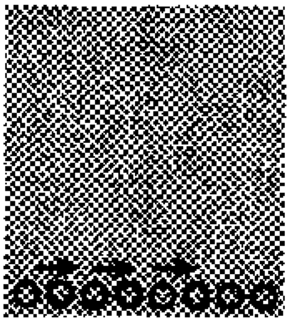
This is the basis of the next set of programs. The correct key must be pressed for the program to be executed. Otherwise the program will return to the HALT instruction and the outputs will not change.

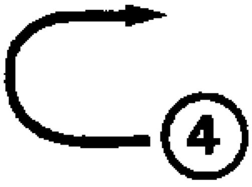
MOVING A LED VIA KEY '4'.

This program moves a LED across the bottom row. It advances one position each time the '4' key is pressed.

No delay routine is employed and the LED will shift at a speed determined by pressing the key.

When the LED reaches one side of the display it re-appears at the opposite side. This can be a distinct advantage when playing some of the games we have devised. At the moment the shift in this program is only left-to-right.





```
LD A,01       800    3E 01
OUT (4),A     802    D3 04
LD B,A        804    47
LD A,B        805    78
OUT (3),A     806    D3 03
HALT          808    76
LD A,I        809    ED 57
CP 04         80B    FE 04
JP NZ 808     80D    C2 08 08
RLC B         810    CB 00
JP 805        812    C3 05 08
```

Accumulator A is loaded with 01 and passed to segment port 4 where it is latched. The contents of A are loaded into register B so that it can be operated upon by the **ROTATE LEFT CIRCULAR** function and also be in a "safe" register, so it is not written over.

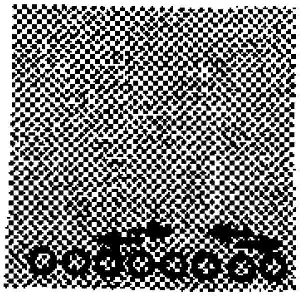
The program is HALTED at 808 and the Z80 waits for a keyboard instruction. When a key is pressed, the NMI line is activated and the data is sent to the Z80 and initializes the INTERRUPT VECTOR REGISTER 'I'. The keyboard data is placed in the accumulator register and compared with the value 04. If the answer is ZERO, the program is incremented to address 810, which instructs the Z80 to ROTATE REGISTER B LEFT. This causes the HIGH bit to shift from bit 0 position to bit 1 position and this will make the LED shift one place to the right on the display when operations at 81C, 81D, 81E, 805, 806, 807 and 808 have been performed.

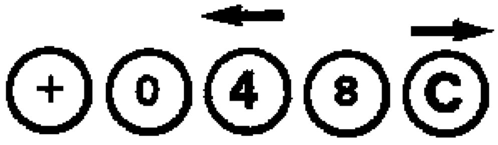
The new data-value in register B is loaded into register A at 805 and is passed to the display latch port 3 at 806 and 807.

The important feature of this program is the use of the interrupt vector register I to detect the input from the keyboard and to enable a compare function to be performed.

SHIFTING A LED ⟷

This program expands on the previous and adds a shift in the opposite direction. We now have a forward and reverse shift.





Key '4' shifts left and 'C' shifts the LED to the right.

The direction of shift is governed by **RLC B** and **RRC B** and these can be swapped to give the opposite effect.

If you require the LED to travel up and down the screen, the output ports 3 and 4 must be reversed in the program.

```
LD A,01       800    3E 01
OUT (4),A     802    D3 04
LD B,A        804    47
LD A,B        805    78
OUT (3),A     806    D3 03
HALT          808    76
LD A,I        809    ED 57
CP 04         80B    FE 04
JP NZ 815     80D    C2 15 08
RLC B         810    CB 00
JP 805        812    C3 05 08
CP OC         815    FE 0C
JP NZ 808     817    C2 08 08
RRC B         81A    CB 08
JP 805        81C    C3 05 08
```

This, and many other features can be altered to suit your own requirements. It is a matter of experimenting and determining which instruction should be altered. If you discover these changes yourself, you will have a much greater understanding of how the program is put together.

The values at 80C and 816 determine which buttons are operative. These can be changed to any pair you choose, simply by inserting the correct data into the program.

The data corresponds to the value which appears on the key, for 0 to F. Keys +, −, GO and AD have the values 10, 11, 12, and 13.

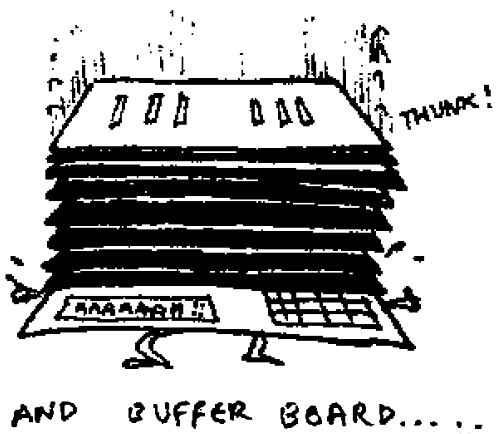
ADDING AUTO REPEAT

A simple addition to the previous program will enable the LED to run across the display in an auto repeat mode, when the correct key is pressed.

This repeat operation is not capable of detecting when the key has been released as the keyboard encoder contains a latch which retains the last value outputted from the key pad.

The NMI line operates a flip flop inside the Z80 which is edge triggered and this means that when it

cont. over . . .



AND BUFFER BOARD.....

cont. next issue!!

is reset, after dealing with the value from the keyboard encoder, it cannot be set again without physically pressing the key AGAIN.

Thus a key pressed for a long time can only be recorded ONCE.

The following program will detect key 4 and run the LED across the screen via a loop in the program and continue to do so until another key is pressed. This is the only way of halting the run.

```
LD A,01          800   3E 01
OUT (4),A        802   D3 04
LD A,01          804   3E 01
OUT (3),A        806   D3 03
LD B,01          808   06 01
HALT             80A   76
LD A,I           80B   ED 57
CP 04            80D   FE 04
JP NZ HALT       80F   C2 0A 08
RLC B            812   CB 00
LD A,B           814   78
OUT (3),A        816   D3 03
CALL DELAY       817   CD 00 0C
JP 80B           81A   C3 0B 08
```

At 0C00:

```
11 FF 0A
1B
7B
B2
C2 03 0C
C9
```

Press RESET, GO.
Press **Key 4** to shift LED.
Press any other key to **HALT** LED.

AUTO RETURN AND STOP

The following program detects 3 keys. The + key shifts the LED left, the 'O' key stops the LED and key '4' shifts it right.

The speed of travel across the display is controlled by the length of time of the DELAY ROUTINE.

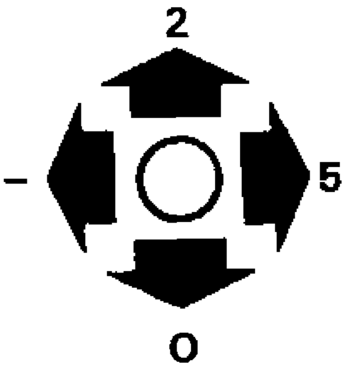
```
LD A,01          800   3E 01
OUT (4),A        802   D3 04
LD A,01          804   3E 01
OUT (3),A        806   D3 03
LD B,01          808   06 01
HALT             80A   76
LD A,I           80B   ED 57
CP 04            80D   FE 04
JP NZ 81A        80F   C2 1A 08
RLC B            812   CB 00
LD A,B           814   78
OUT (3),A        815   D3 03
CALL DELAY       817   CD 00 0C
CP 10            81A   FE 10
JP NZ HALT       81C   C2 0A 08
RRC B            81F   CB 08
JP 80B           821   C3 0B 08
```

At 0C00:

```
11 FF 0A
1B
7B
B2
C2 03 0C
C9
```

4-DIRECTION SHIFT

This program is an extension to the previous listing to obtain a 4-direction shift.



The four buttons we have chosen for controlling the LED are: –, 5, 2 and 0. There is no auto repeat feature in this listing and the LED can be moved around the entire display by using the keys mentioned.

```
LD A,01          800   3E 01
OUT (3),A        802   D3 03
LD B,A           804   47
LD A,01          805   3E 01
OUT (4),A        807   D3 04
LD C,A           809   4F
HALT             80A   76
LD A,I           80B   ED 57
CP 11            80D   FE 11
JP NZ 81A        80F   C2 1A 08
RRC B            812   CB 08
LD A,B           814   78
OUT (3),A        815   D3 03
JP 80A           817   C3 0A 08
CP 05            81A   FE 05
JP NZ 827        81C   C2 27 08
RLC B            81F   CB 00
LD A,B           821   78
OUT (3),A        822   D3 03
JP 80A           824   C3 0A 08
CP 02            827   FE 02
JP NZ 834        829   C2 34 08
RRC C            82C   CB 01
LD A,C           82E   79
OUT (4),A        82F   D3 04
JP 80A           831   C3 0A 08
CP 00            834   FE 00
JP NZ 80A        836   C2 0A 08
RLC C            839   CB 09
LD A,C           83B   79
OUT (4),A        83C   D3 04
JP 80A           83E   C3 0A 08
```

This program is the basis of a game we will be presenting in the next issue. Basically it is a **HUNT THE FOX** game in which a secret co-ordinate is selected and the object of the game is to locate the fox in the MINIMUM NUMBER OF MOVES. The LED is the pack of hounds and when they coincide with the fox, the screen will flash a victory or produce a hunting tune.

The completion of the game is up to you. Try your hand at writing a game along these lines and send it in for publishing in the next issue.

In the little space left I would like to include a program from one of our readers.

Inspired by the content of issue 9, a TEC-1 and a Z80 Machine Code book, he has written a sound effects program which will really amaze you. It is a complex sound generator which is fully programmable and it is only when you start to change some of the data bytes, that you will see how it goes together.

ALIENS ATTACK RUN

-by M J Allison, 3095

```
LD HL,0903       800   21 03 09
LD A,01          803   3E 01
LD HL,A          805   77
LD C,30          806   06 30
CALL 0903        808   CD 0E 09
INC (HL)         80B   34
DJNZ CALL        80C   10 FA
JP 0800          80E   C3 00 08
```

```
PUSH AF          900   F5
PUSH DE          901   D5
LD DE, 0020      902   11 20 00
DEC DE           905   1B
LD A,D           906   7A
OR A,E           907   B3
JP NZ 905        908   C2 05 09
POP DE           90B   D1
POP AF           90C   F1
RETURN           90D   C9
PUSH AF          90E   F5
PUSH BC          90F   C5
LD BC,00AA       910   01 AA 00
LD A,80          913   3E 80
OUT (1)A         915   D3 01
LD A,00          917   3E 00
OUT (1),A        919   D3 01
CALL 0900        91B   CD 00 09
DEC BC           91E   0B
LD A,B           91F   78
OR A,C           920   B1
JP NZ 913        921   C2 13 09
POP BC           924   C1
POP AF           925   F1
RETURN           926   C9
```

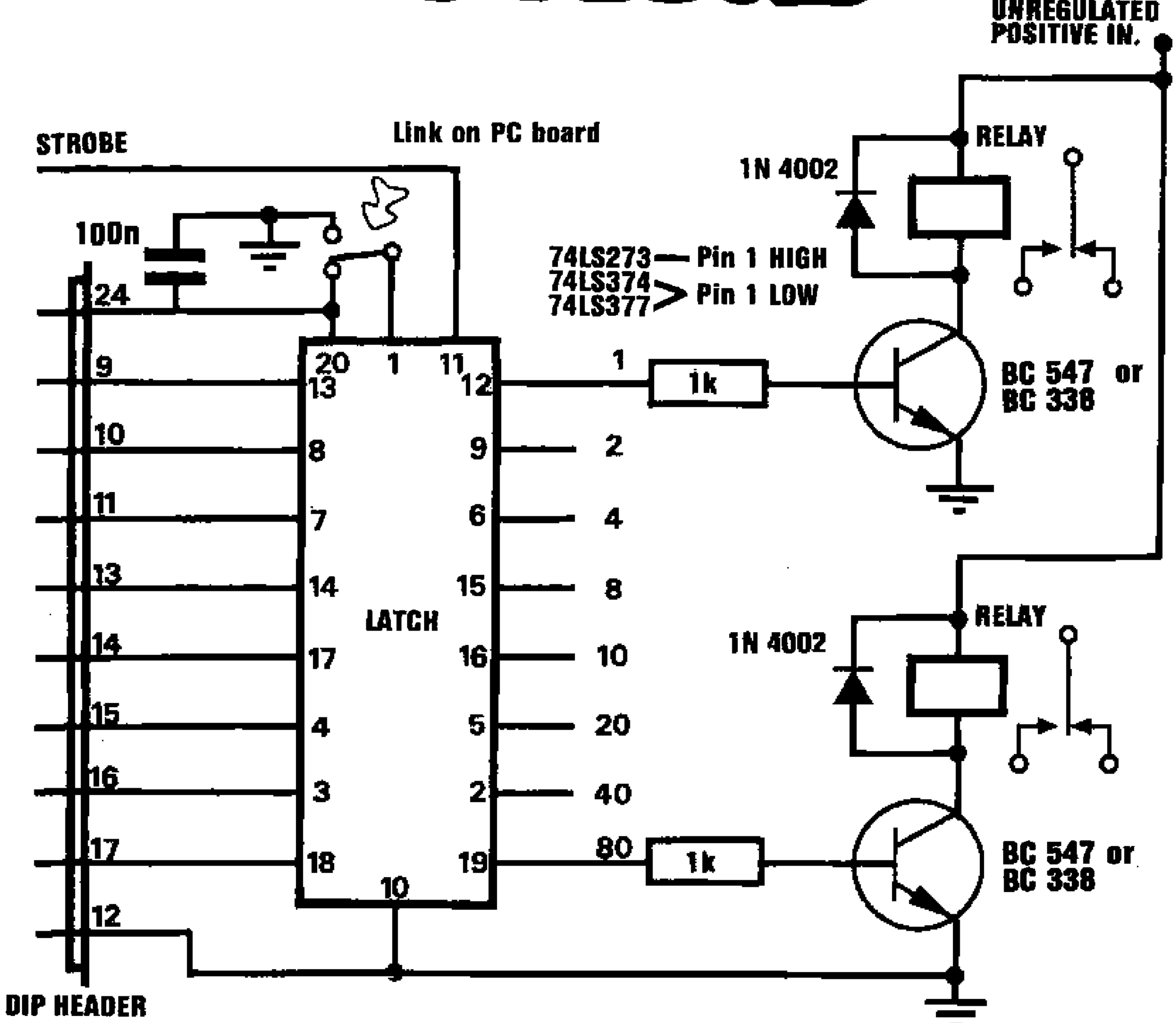
I have run out of room for this issue and still have lots more programs and ideas. Next issue will contain another 20 pages of programming and include 2 more programs from Mr Allison.

Turn to P.50 for 6 pages on the RELAY DRIVER BOARD project and type in the program for operating the relays.

The projects for next issue . . . . .I'll keep them a secret, but you'll be very pleased; I assure you.

RELAY DRIVER BOARD

$25.60 Complete



—TEC-1 RELAY BOARD—

The TEC-1 Relay Board has been designed to give the TEC-1 an interface with the real world.

It can be used in such applications as the control of model railways or complex light sequencers. It can also be used to implement timing and control functions for machinery.

The project will drive 8 on-board relays or external relays with suitable coil resistances and/or it is possible to drive small low-voltage lamps directly.

We do not recommend switching 240v via the relays on the PC board as the terminations are very near the other TEC-1 components.

If you require to switch 240v, the on-board relay can become a slave to power the higer-voltage relay.

Each of the relay contacts has been brought to the edge of the PC board where it is labelled 'NO' for Normally Open contact and 'NC' for Normally Closed contact. The centre 'C' is the Common contact.

The Hexadecimal value for each relay has also been printed on the board to make programming easier.

If relays are not required, small light bulbs can be connected between the two holes normally used for the relay coil.

CONSTRUCTION

Three types of latch chips can be used on the board. These are: 74LS273, 74LS374, or 74LS377.

Each chip is capable of performing the latch function and the only pin which requires a different logic level is pin 1.

On the 74LS273, pin 1 is master RESET, active LOW. So this pin must be tied HIGH to stop this chip from remaining in the reset condition.

On the 74LS374, pin 1 is an output enable, active LOW. When pin 1 is HIGH, the outputs will go to a high impedance state, effectively disconnecting them from circuit. When pin 1 is LOW, the outputs are normal, being either HIGH or LOW, depending on the contents of the latch.

Pin 1 on the 74LS377 is an ENABLE, active LOW. When this pin is HIGH, the contents of the latch cannot be changed by any of the input signals. When the pin is LOW, the latch operates as normal.

None of these special funtions are required on our project so pin 1 can be tied HIGH or LOW and remain in this condition.

For the 74LS273, pin 1 is tied HIGH and this requires a very short jumper as shown on the overlay.

For the 74LS374 and 74LS377, pin 1 is tied LOW and this requires a jumper link the full length of the IC as shown on the overlay.

Solder in the necessary link for pin 1 and another link on the other side of the IC.

Next fit the eight 1k resistors, the eight NPN transistors and eight protection diodes. A 100n decoupling capacitor fits above the input socket.

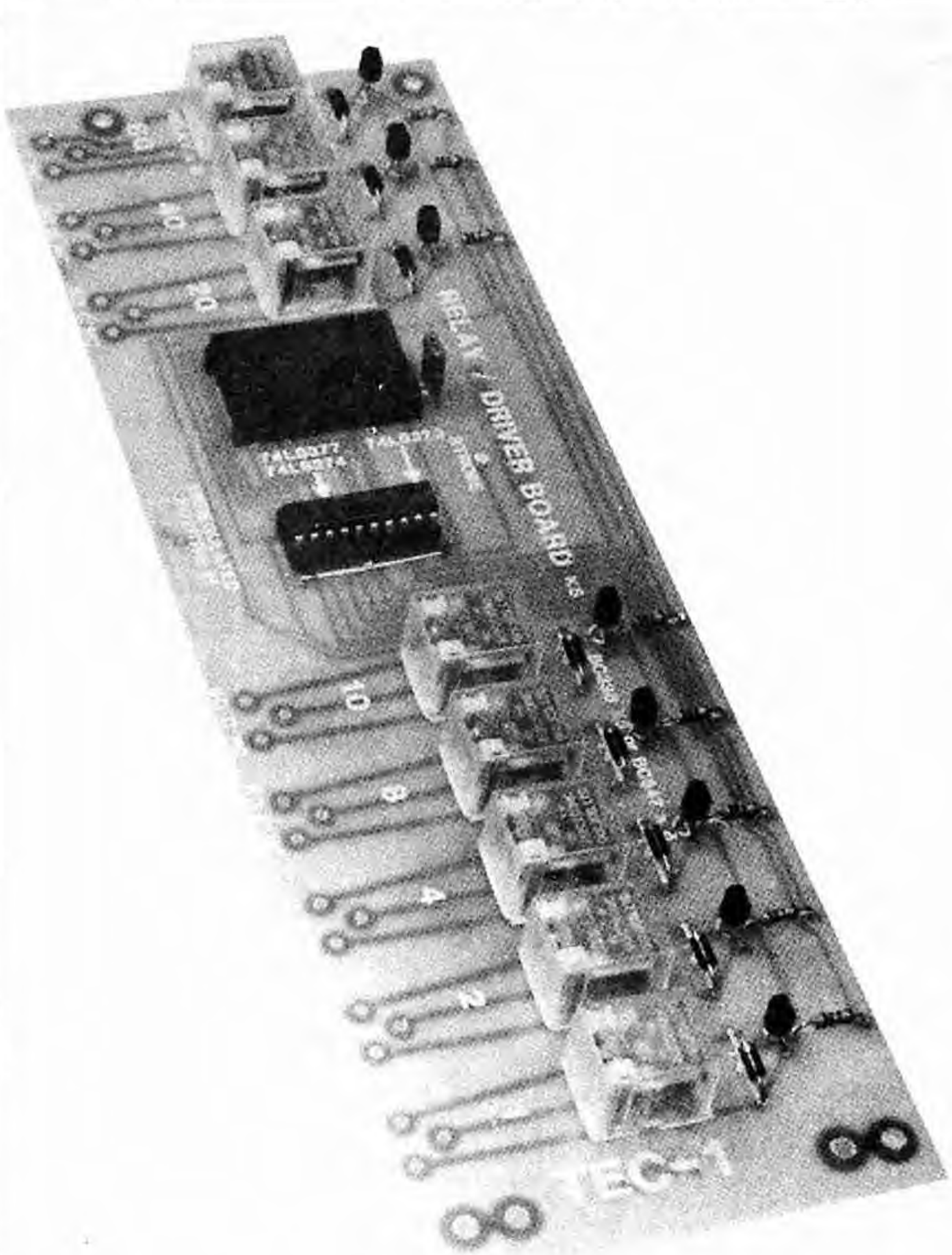
The board is designed to take two types of relays. They are both single pole double throw types and one has a contact rating of 1 amp, while the other has a 3amp contact rating. Select the type your require and install them.

This unit is designed to plug into the EXPANSION PORT SOCKET on the TEC-1 and there is two methods of connecting them together.

One is to individually wire the leads onto a dip header or dip plug and plug it into the expansions socket. The other is to use a wire wrap socket. The socket is inserted into the holes on the PC board and pushed home. The pins are now carefully soldered. Next a dip header is soldered to the end of the long leads of the wire-wrap socket. Solder the corner pins first, then align the rest of the pins and solder them. Care must be taken not to over-heat the dip header as the plastic melts easily.

Finally insert a 20 pin IC socket and fit the latch IC.

Two jumper wires are also needed to complete the wiring. These are taken from the 'STROBE' hole and the UNREGULATED POSITIVE IN hole.



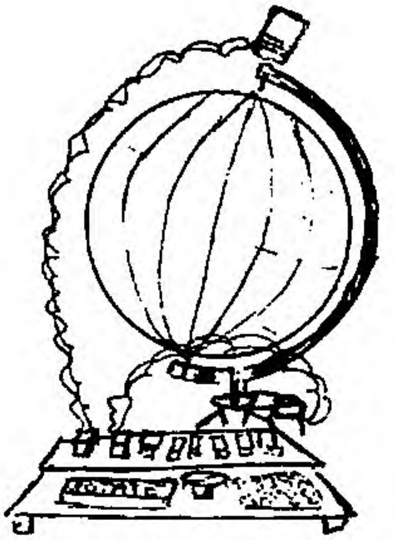
COMPLETED RELAY DRIVER BOARD

The Unregulated IN line goes to the INPUT terminal of the 7805 regulator as 12v is needed to operate the relays.

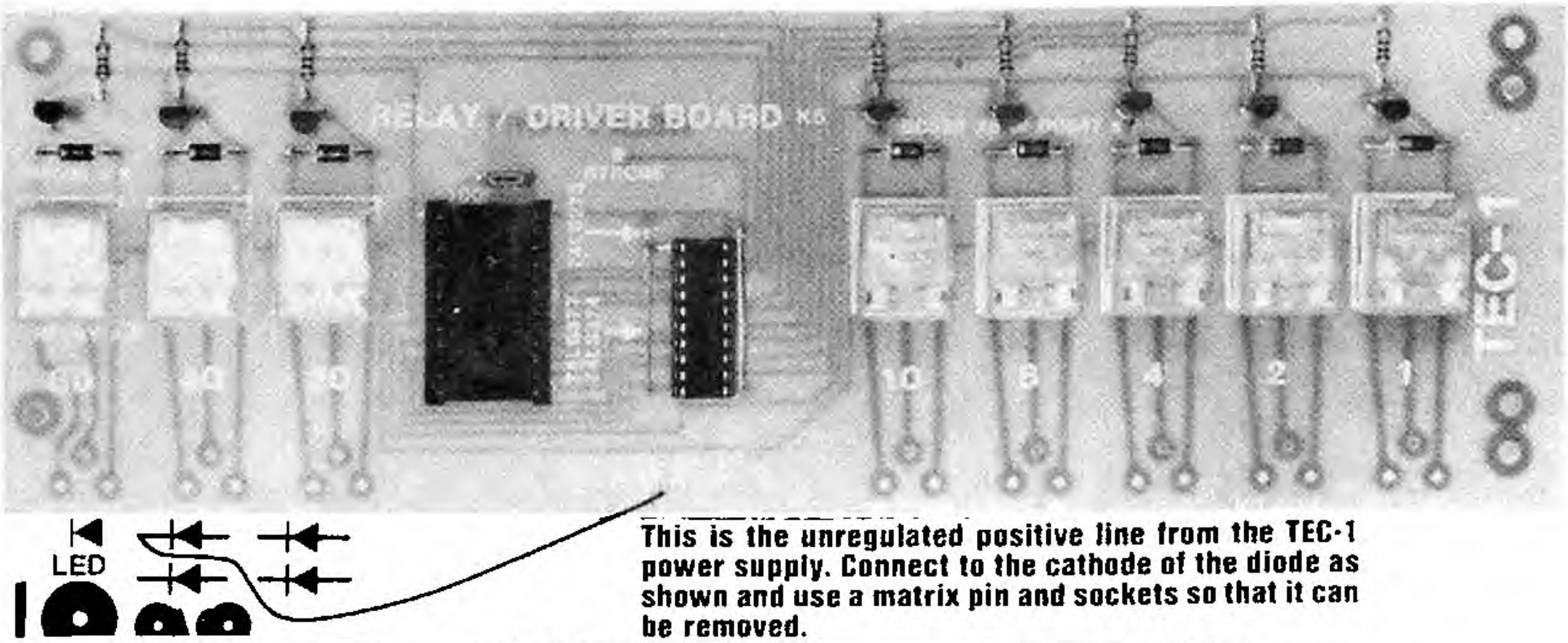
The other line, from the STROBE hole, is taken to pin 12 of the 74LS138 IN-OUT decoder nearest the 74C923 chip. This is output 03.

Solder a 24 pin IC socket to the EXPANSION PORT on the TEC-1 and plug the RELAY BOARD dip header socket into this expansion port socket. The board should fit neatly above the TEC-1. The printing on the RELAY board will be up-side-down because all wiring to the relays is done from the back.

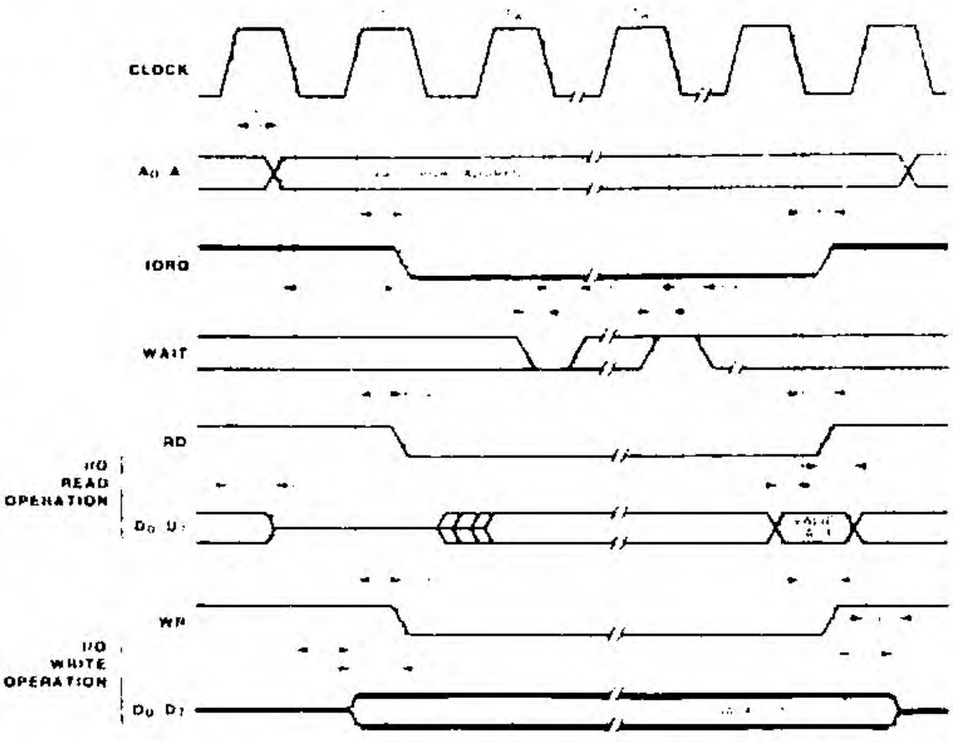
The unit is now ready for use.



I TOLD THEM I WANTED TO CONTROL THE WORLD!



This is the unregulated positive line from the TEC-1 power supply. Connect to the cathode of the diode as shown and use a matrix pin and sockets so that it can be removed.



SOURCE: SGS DATA BOOKS.

HOW IT WORKS

The latch board works in the same manner as the display latches and displays.

Look at the timing diagram for the Z80 INPUT/OUTPUT cycle.
Compare the IORQ signal to the DATA BUS signal. It can be seen that the DATA BUS stabilizes first, then the IORQ signal goes LOW and HIGH again.

During this the DATA BUS has remained stable and now that the cycle is complete, it is no longer required.

The latch used is a positive-edge-triggered device. It can be seen that the positive-going edge of the IORQ, which is passed to the latch via the 74LS138, comes late in the cycle, but as the data is still stable, the latch is able to record this information and energise a relay.

The port which is to receive the data is selected by the 74LS138, according to the information on the address bus. There are 256 ports addressable by the Z80, but only 8 are decoded on the TEC-1. This still gives us the availablility of using 40 relays!

Up to 5 boards can be added to the TEC-1. These are stacked on top of each other and accessed via ports 03, 04, 05, 06, and 07. Ports 00, 01 and 02 are used by the TEC-1 for the keyboard, and displays.

HOW TO USE THE RELAY BOARD

If you have a Mon 1A EPROM, try the following:

Start at address 0800, program the following: 01, 02, 04, 08, 10, 20, 40, 80, FF ADdress 05B0, GO, GO.

The relays will sequence in order.

If you have the MON-1 EPROM, enter the following at 0800: 3E, 01, D3, 03, C7. RESET, GO.

The first relay will switch on and the computer will reset. Substitute different values for 01 in the program and watch the operation of the relay or relays.

With this RELAY BOARD your TEC-1 can access the REAL WORLD.

We will now show you how to use it and produce delay routines of varying length so that you can turn lights, motors and pieces of equipment on and off.

PARTS LIST

8	-	1k ¼ watt
1	-	100n greencap
8	-	BC 547 or BC 338 transistors
8	-	1N 4002 diodes
8	-	relays SPDT type S 4060
1	-	74LS273 Latch IC
1	-	wire-wrap socket 24 pin
1	-	dip header 24 pin
1	-	20 pin IC socket

The RELAY BOARD takes up to 8 relays and these can be turned ON and OFF in absolutely ANY order, by the program we have developed.

The only limitations are very heavy currents and high voltages. The relays are somewhat exposed and are close to the components on the TEC-1, so we do not suggest 240v switching.

Secondly the relays are designed for 1 amp operation and should be limited to around this value for reliable operation.

We suggest only low voltage operation (12v) with currents up to 1 amp for the relays and transistors. We have a high current version (3 amp), S 4066 and this will enable loads up to 36 watts to be handled by each output. This is quite considerable in electronic terms.

Higher currents and voltages will have to be handled via a relay which is remotely positioned so that it can be provided with the insulation it needs. Don't forget, any relays switching high voltages can still have a 12v coil. This means that you can combine relays and transistors on the one RELAY 8OARD.

For fast sequencing, a set of transistors will be the best choice as relays have a low operating frequency.

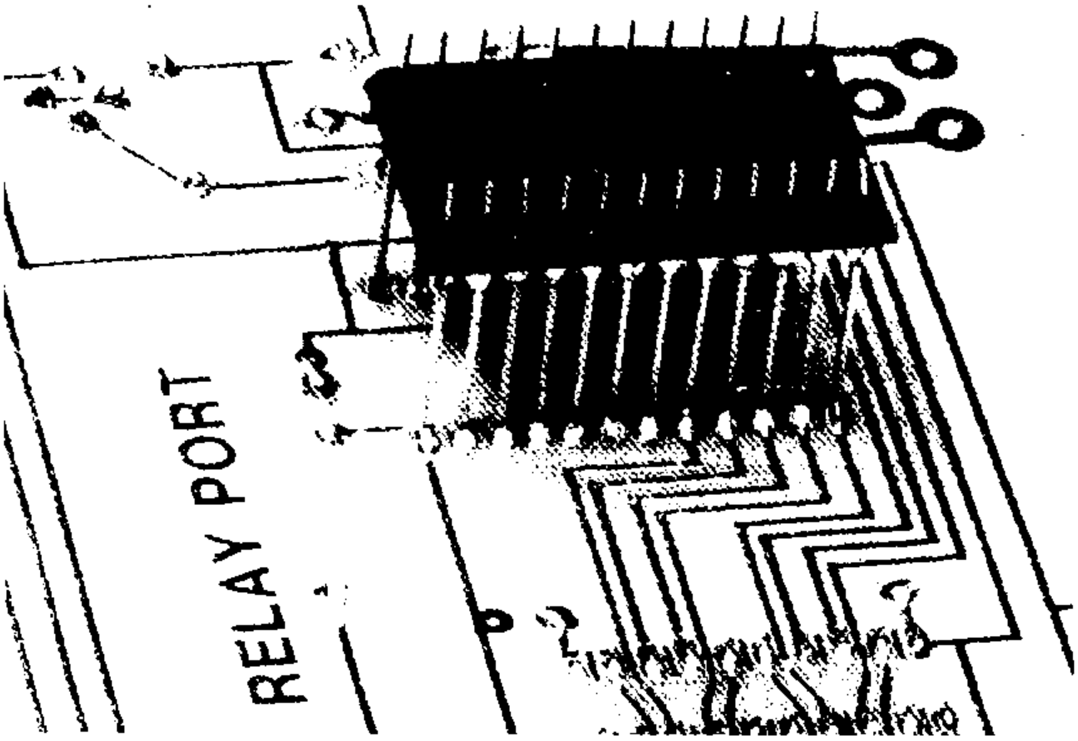
The choice of using a relay or transistor will depend on the type of load you are driving. You will need to know such factors as current requirements, voltage, speed of operation and if any spikes are present.

Our main concern at this stage is to test the RELAY 8OARD and get it working.

To do this you will will need some form of indication on each of the outputs to show when they are operating. This can be a miniature lamp or a light emitting diode. Fit 8 of these and keep them near each relay to show which relay they are indicating.

The program that we have developed for this project controls the 8 outputs and will switch them ON and OFF in any combination, as per the program.

It is merely a matter of changing the data at **0A00** to produce the required sequence. The data governs the time of the delay between each operation and this can be set for a millisecond delay or up to a 100-hour delay. To do this requires 4 bytes of information. The next byte governs how many relays are activated at the end of each delay. These 5 bytes form a 'UNIT' or 'STEP'.



The diagram shows a close-up of the plug which connects the relay board to the TEC-1. It is made up of a wire-wrap socket and a dip-header plug. The wire-wrap socket is mounted on the components side of the board with its long leads extending out the side shown in the photo. The dip plug is soldered onto these 'thick' leads, effectively turning them into thin pins suitable for inserting into the EXPANSION PORT socket.

THE PROGRAM:

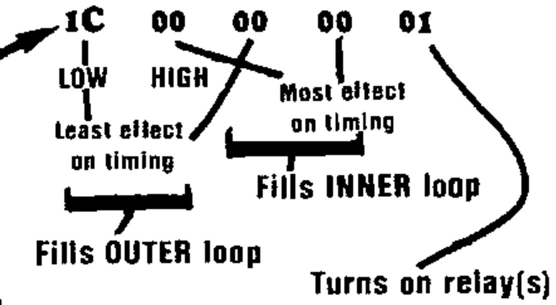
```
LD B,dd            800   06 08 (No of steps)
LD HL,A00          802   21 00 0A
LD IX,904          805   DD 21 04 09
LD A,(HL)          809   7E
LD (IX +00),A      80A   DD 77 00
INC HL             80D   23
LD A,(HL)          80E   7E
LD (IX + 01),A     80F   DD 77 01
INC HL             812   23
LD A,(HL)          813   7E
LD (IX+ 03),A      814   DD 77 03
INC HL             817   23
LD A,(HL)          818   7E
LD (IX + 04),A     819   DD 77 04
INC HL             81C   23
LD A,(HL)          81D   7E
INC HL             81E   23
OUT (03),A         81F   D3 03
CALL 0900          821   CD 00 09
DEC B              824   05
LD A,B             825   78
OR A               826   B7
JP NZ, 0809        827   C2 09 08
JP 0800            82A   C3 00 08
```

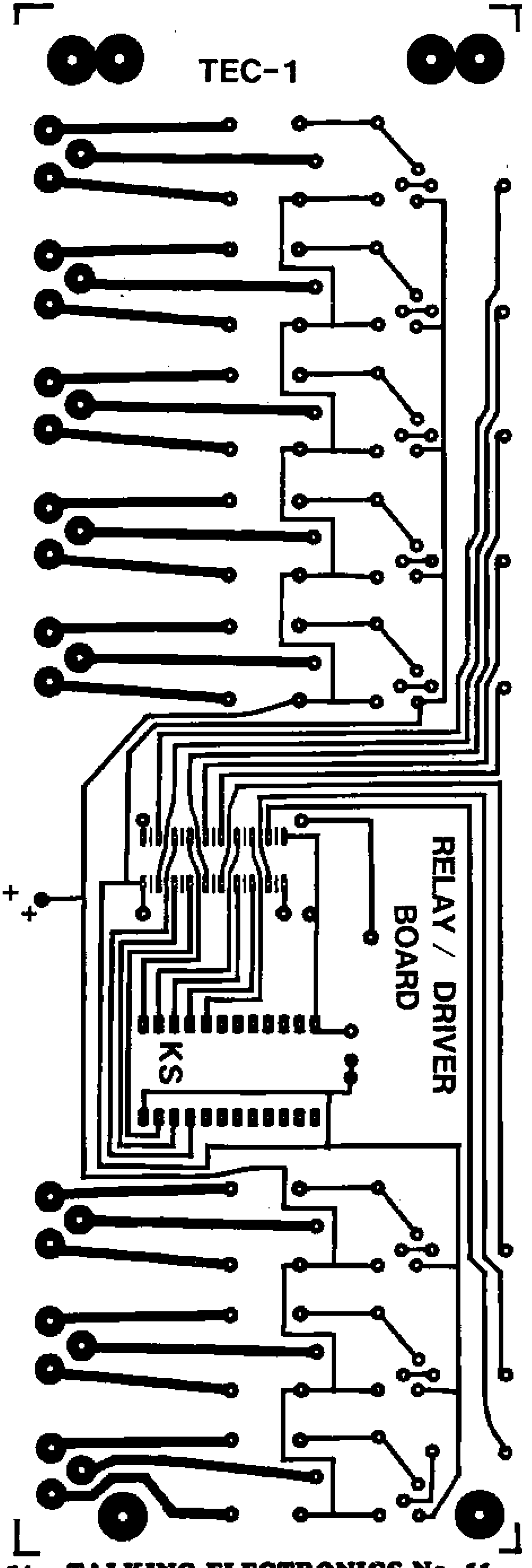
```
PUSH AF            900   F5
PUSH BC            901   C5
PUSH DE            902   D5
LD BC, FF FF       903   01 FF FF
LD DE, FF FF       906   11 FF FF
DEC DE             909   1B
LD A,D             90A   7A
OR E               90B   B3
JP NZ 909          90C   C2 09 09
DEC BC             90F   0B
LD A,B             910   78
OR C               911   B1
JP NZ, 906         912   C2 06 09
POP DE             915   D1
POP BC             916   C1
POP AF             917   F1
RET                918   C9

DATA               1C 00 00 00 01
at 0A00:           38 00 00 00 23
                   1C 00 00 00 86
                   1C 00 00 00 2C
                   38 00 00 00 38
                   1C 00 00 00 32
                   1C 00 00 00 62
                   1C 00 00 00 C0
```

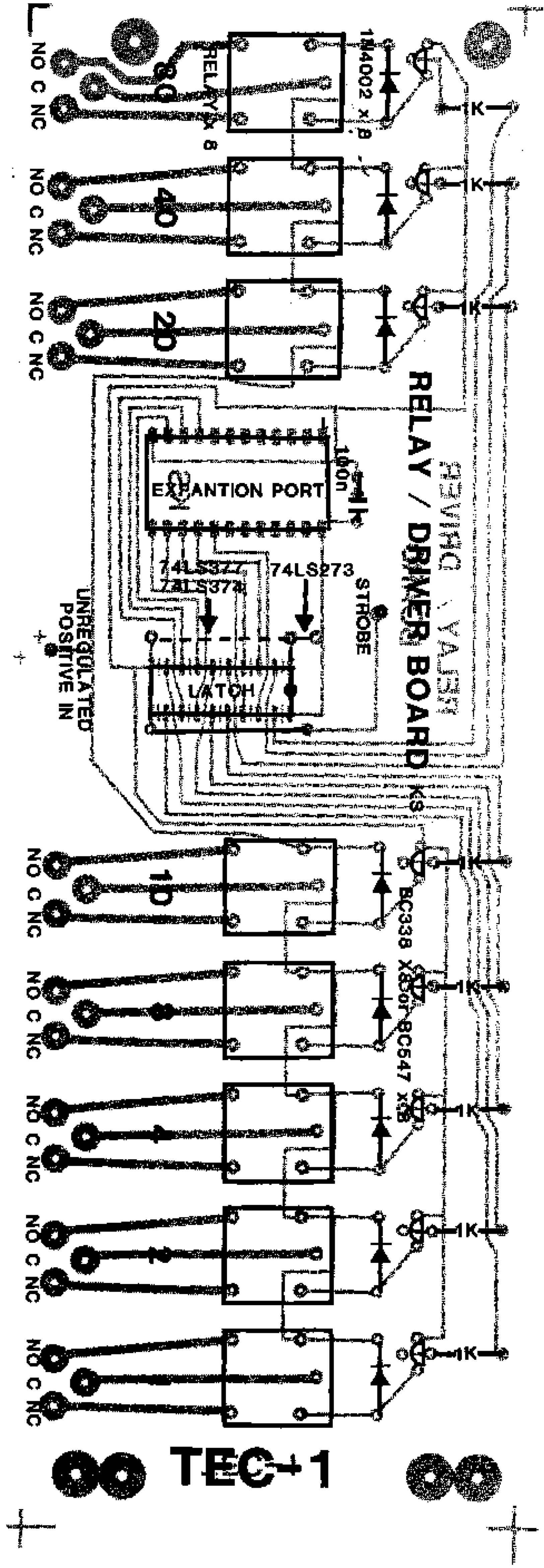
The theory of operation of this program is quite complex. It will take two or more chapters before we are up to describing its features. In the meantime we will describe it in a simplified way.

What each byte does:

At **0A00** the data bytes are taken in groups of 5:




TEC-1
RELAY / DRIVER BOARD
KS
+
+


RELAY x 8
1N4002 x 8
NO C NC
8D
4D
2D
1D
1K
RELAY / DRIVER BOARD
EXPANTION PORT
100n
74LS377
74LS374
74LS273
STROBE
UNREGULATED POSITIVE IN
+
LATCH
KS
BC338 X8 or BC547 x8
TEC-1

. . . . .from p. 53.

Here is an overall view of the program:

The program consists of 3 sections: The main program is at 800. This can be likened to a cook book. They both give an overall picture of what you are trying to achieve.

The second part of the program is the DELAY ROUTINE. This is at 900 and is similar to the mixing and cooking times for a recipe. They both involve TIME, and this is the important element of this section.

The third part is the DATA. This is similar to the ingredients: flour, milk, eggs, etc. The more flour you add, the longer you have to mix.

That's a simple analogy. This is how it works:

The main program (the procedure for the recipe. Such as getting out the bowls and equipment) contains all the 'setting-up' data and instructions with the first instruction, at 801, telling the computer how many cakes (steps) you require.

At 900 the length of time for the mixing will depend on quantity of flour and milk is going to be used.

0A00 contains the quantity of these ingredients.

The program can make up to 256 cakes (steps) before it starts to repeat.

The main program may look complex but most of it is a set-up routine, like getting everything together to make a cake.

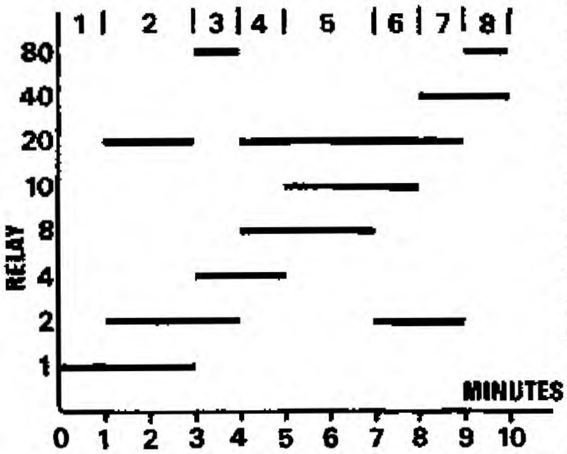
To extend this program is very easy, all you need do is add a 5-byte block of data to the end of the data at 0A00. Each block will produce one more delay and instruct a set of relays to come on at the end of the delay. The value of 'B' at 801 must correspond to the number of steps in the program otherwise it will be forgotten!

The first three bytes in the delay (at 0900) PUSH the registers onto the STACK making the delay 'transparent' to the main program.

At the end of the delay, the registers are POPPED off the stack. This means that they will contain the same value after the delay, as they had before it.

The main part of the delay program consists of two routines, one inside the other. The inside or 'nested' delay starts at 906 and uses the register pair DE. The delay-value is loaded into the register pair and decremented by one. The accumulator is then loaded with the contents of the D register and the E register is ORed with the accumulator. The result appear in the accumulator. The accumulator will be zero if both D and E are completely zero and this will set the zero flag.

The next instruction JP NZ 909 checks if the zero flag has been set and if it has, the computer will execute the next instruction. If it has not been set, the computer will jump to 909 and perform another DECrement instruction.



The graph of our program. It shows the operating time for each of the relays.

When the nested delay is zero, the computer will decrement the outer delay, check that it is not zero and proceed to decrement the inner delay. When the two delays are BOTH zero, the original values of the six registers are pulled (POPed) off the stack and the program returns to the main program.

DELAY LENGTHS

The values 00 00 in the data at **0A00** may look like a short delay length, but they are not. If fact we have tried to trick you! **00** is, in fact the longest delay you can get! If you load the register with **00 00** it is decremented by one to get **FF FF** and then checked to see if the value is zero. As you can see, it is not! The shortest delay is **01 00** as this is immediately decremented to **00 00**, then checked.

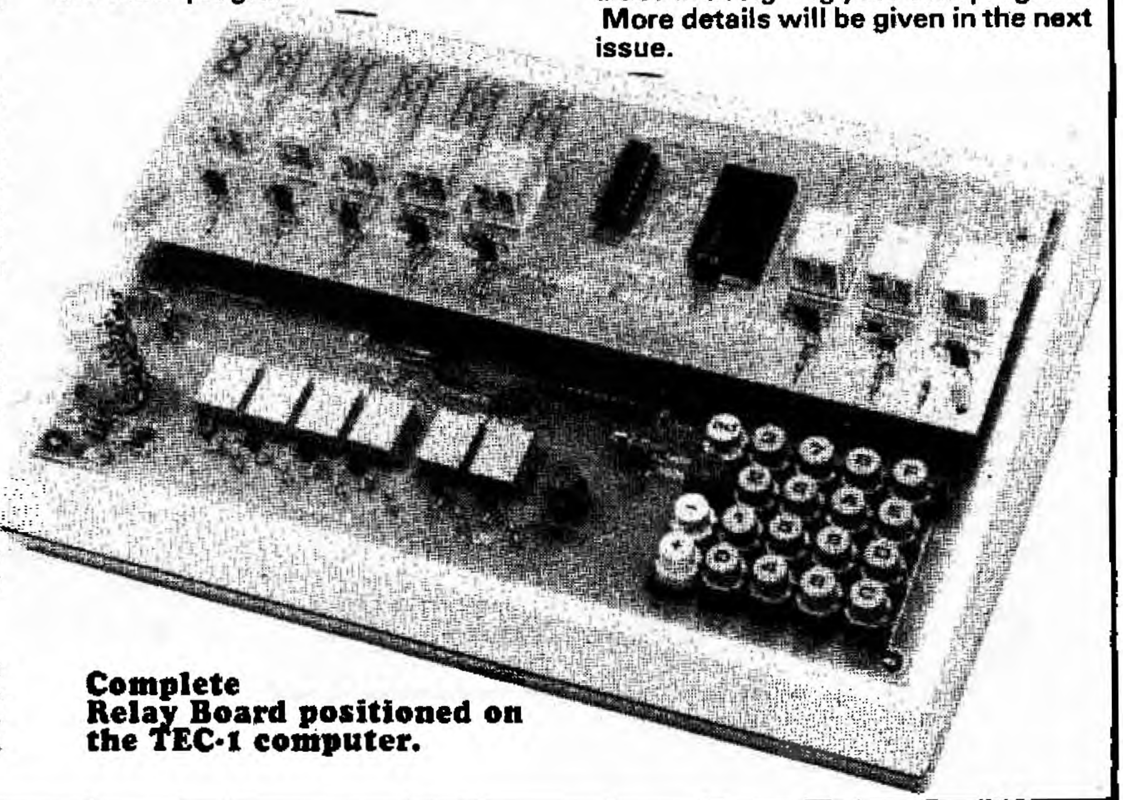
To give an example of the time taken to execute **00 00**, we place **00 00**, in the nested delay and **01 00** in the outer delay. This will produce a time delay of 2 seconds, when the computer is turned to the high position. The value **1C 00** for the outer delay will produce a time of about 1 minute.

If you require a delay of 10 minutes, the value of the outer delay is increased 10-fold by inserting the value 1C 0A. For a 100-minute delay, the value is 1C 64. See the HEX table on P.16 for additional values.

RUNNING THE ROUTINE

Enter the program into the TEC-1 and set it into operation. It will take 10 minutes to execute and has 8 steps. There are six 1-minute steps and two 2-minute steps.

The length of the delay for each step can be individually programmed as can the data for the number of relays which are to be activated. The accompanying diagram shows the various relays which will be activated and by referring to this, you will be aided in designing your own program. More details will be given in the next issue.



Complete Relay Board positioned on the TEC-1 computer.

TALKING ELECTRONICS®

$2.20 ★

$2.95 NZ

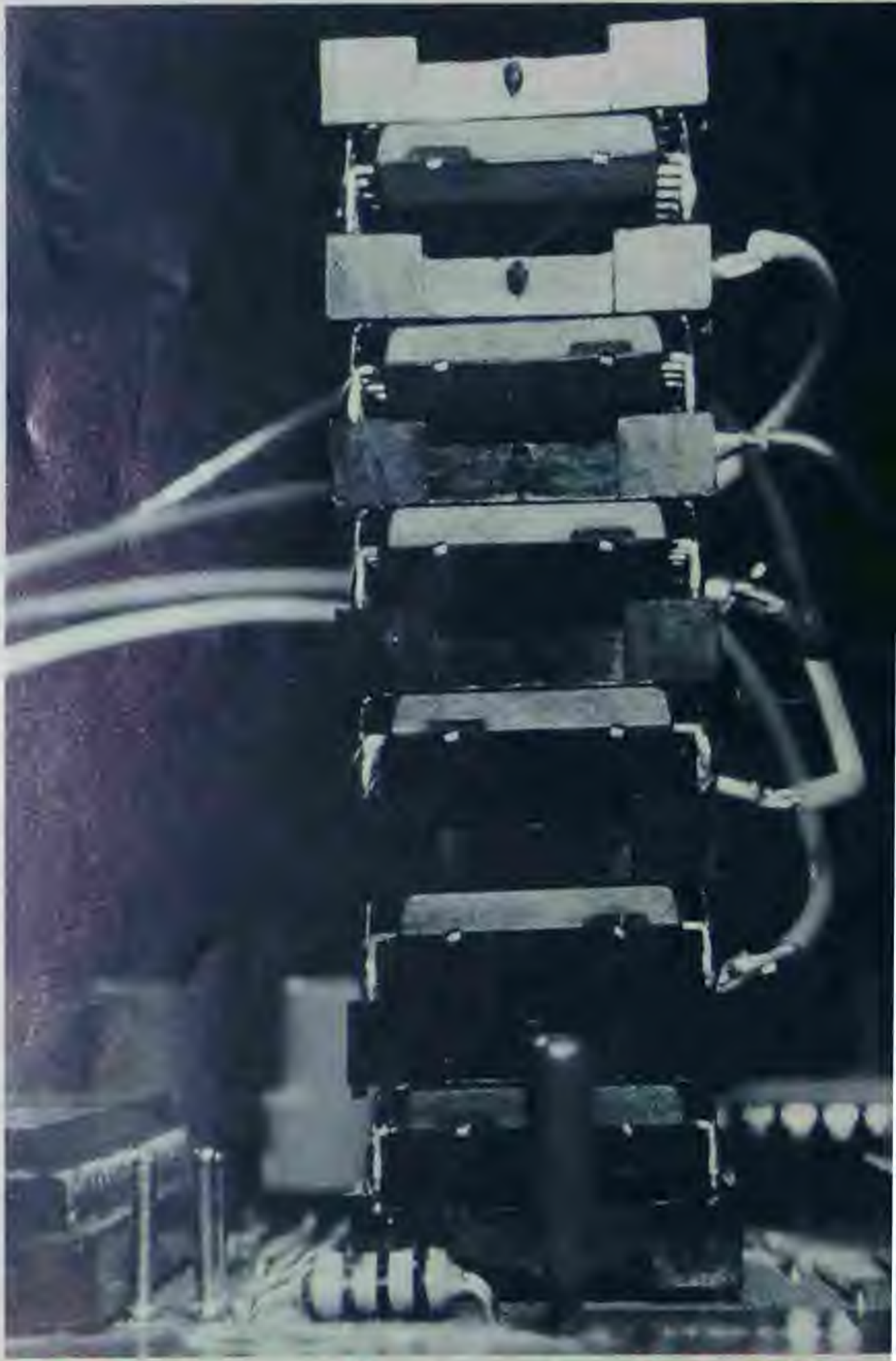
Issue No 12

HEADLIGHT

REMINDER

BIG EAR

TOUCH PUZZLE



2 'Add-ons' for the TEC

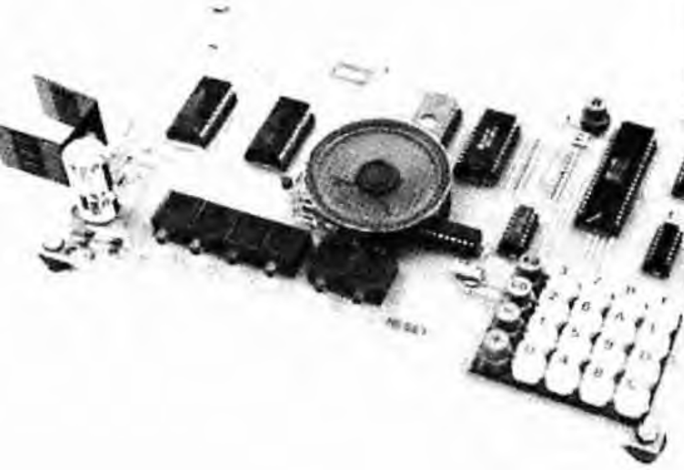
★RAM STACK

★PRINTER/PLOTTER

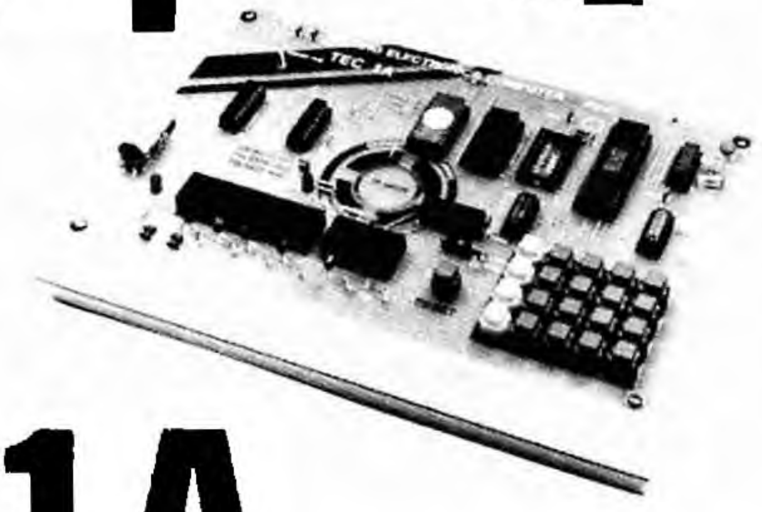
TALKING ELECTRONICS COMPUTER

TEC-1

PART III



&



TEC-1A

The TEC-1A is an update of the TEC-1. For all programming, both are the same. Only TEC-1A kits are available from the kit suppliers.

FEATURES IN THIS ISSUE:

- ★ **Programs for the 7-segment display.**
- ★ **Programs for SOUND and TONES**
- ★ **Programs for the 8x8**
- ★ **RAM STACK Project**
- ★ **Interface for PRINTER: PLOTTER**
  - **Text mode**
  - **Graphics mode**
  - **Computer Graphics.**

Parts: $74.00
PC board: $21.00
Case: $19.00
Post: $5.00 MAX.
'Add-ons' as required . . .

With over 1,000 TEC's in the field, we have had a lot of feedback on the success of this project.

Most of them worked first go and this was very encouraging, considering the complexity of the unit.

Five schools bought class sets and this included a University in New Zealand.

More schools and TAFE colleges will be including them in their microprocessor courses and we will see more of this type of demonstrator in the near future.

It's undeniable that the Z80 is one of the most brilliantly designed microprocessors and it is unfortunate that so little has been written about it.

This is most probably due to the designers and manufacturers of the chip. Until recently, the entire production has enjoyed advanced orders. They were extensively used in controlling applications, computers and intelligent games.

In fact the original manufacturer, Zilog, could not keep up with supplies for its own orders and franshised SGS to make the shortfall. With the down-turn of sales, we have seen some of the chips come on the market and in the initial stages, nobody knew what to do with them.

A few Z80 programming books came on the market but were generally hard to follow and severely lacking in information. Most of them didn't explain how the programming went together. This is why we have designed this series.

In the first two parts we covered some experiments to show how the display was accessed and how to create simple movement and sound effects.

In this part we are continuing along the same lines with slightly more advanced material and introduce a couple of 'add-ons' for those who are advancing faster than the midstream.

Try them all, in the order presented and you will see the concept of MACHINE CODE programming fall into place.

TEC-1A

After the first run of 1,000 boards we found the 8212 display driver chips were fairly difficult to get. So we decided to update the board and include a few small improvements at the same time. Both the TEC-1 and TEC-1A operate with the same software however the TEC-1A has 3 small changes.

The regulator is mounted under the PC board so that it cannot be bent over and broken off, the 2,200uf electrolytic has been changed to 1,000uf and the output latches have been changed to 74LS273 (or 74LS 374 or 74LS377). In all other respects, the boards are identical.

TEC CASE

Either board can be mounted on a **RETEX BOX,** size:RA-2 as shown on P.3 of issue 11. From that small photo we sold hundreds of cases, proving that the magazine is read from cover to cover.

These boxes not only neaten your project but strengthen the board near the keys. There is sufficient room inside the case to fit a power transformer however we suggest using an external plug pack connected to the computer via a plug and socket on the side of the case.

If you want a TEC-CASE, they are available from us for $19.50 including post and packing.

THE ADVENTURES OF Mr WRAGG:

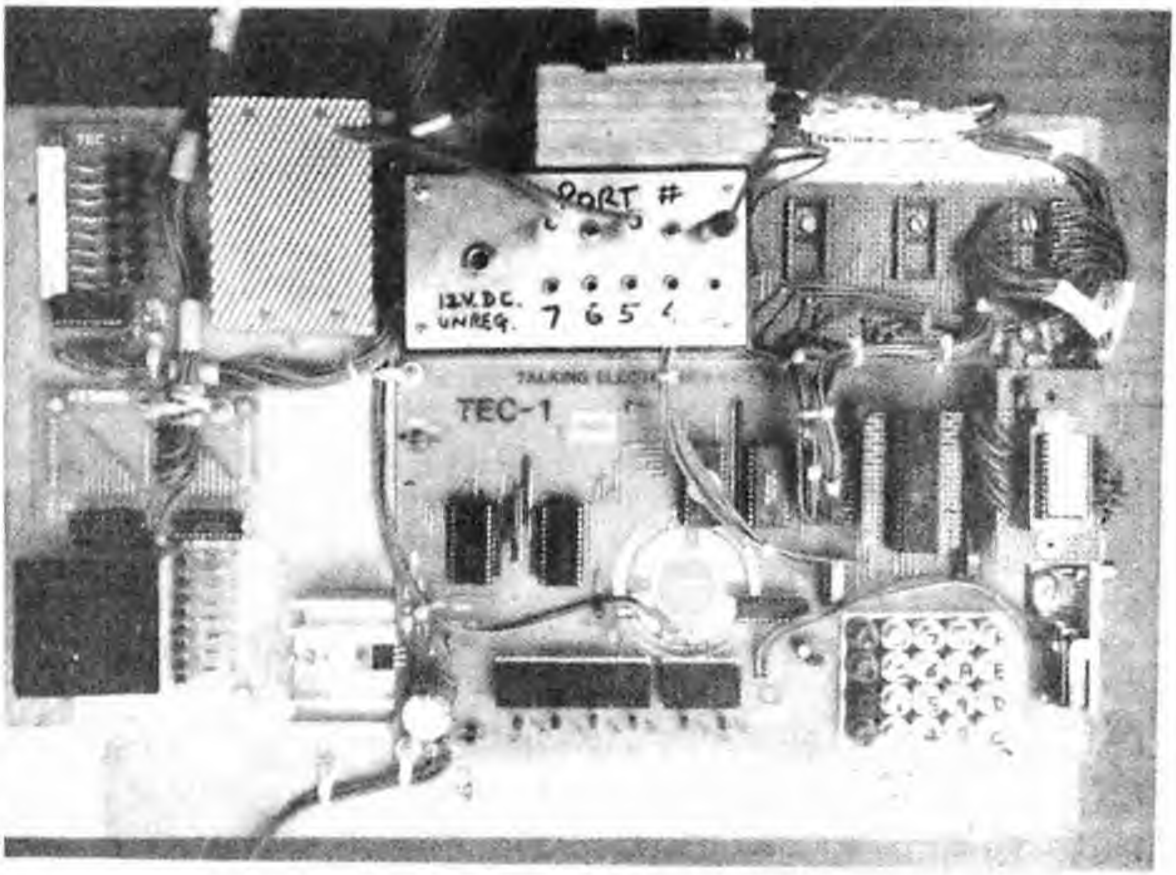
Some constructors have used their own method of mounting the PC board and others have strengthened the weaker parts of the design such as the PC board near the keys, the 7805 regulator mounting, the expansion socket etc.with a variety of ideas. But none have done more than a teacher from Brighton High School. Mr Wragg has made his computer STUDENT-PROOF. He has strengthened every lead and plug by using clips, clamps, thick hook-up wire and fasteners where ever possible to reduce the possibility of damage.

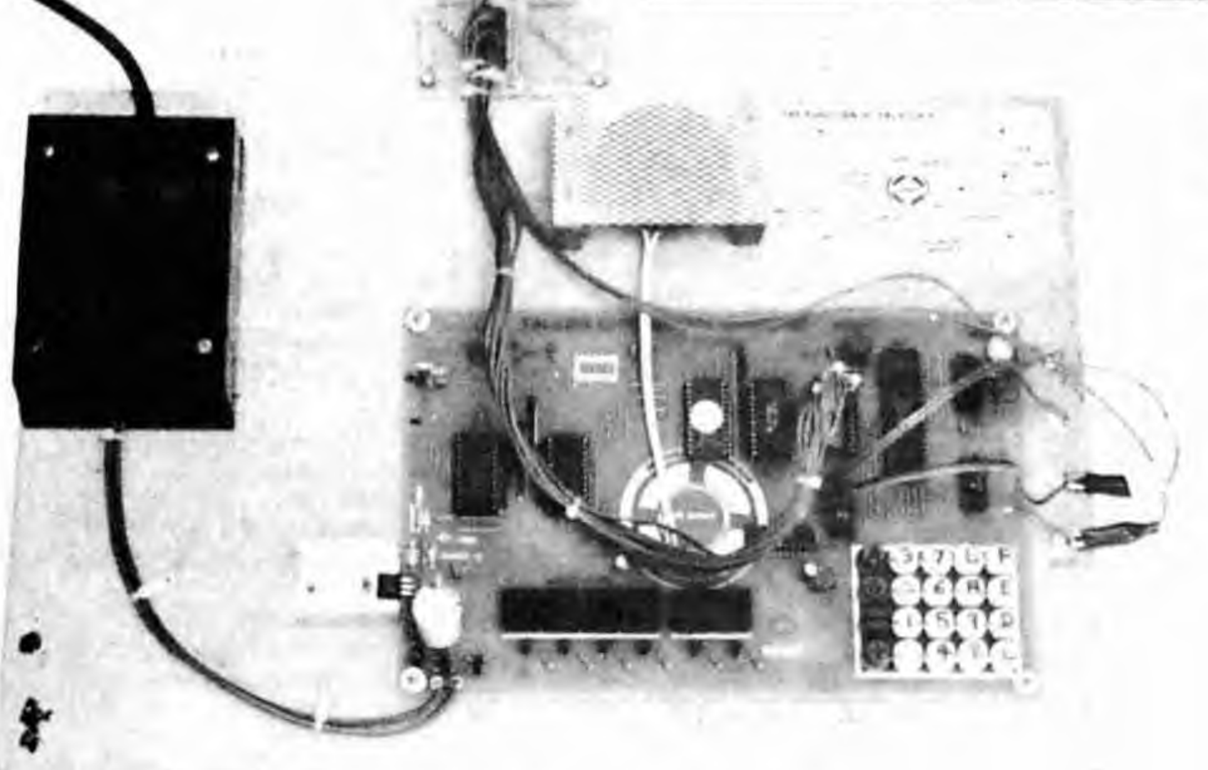
The results are shown in the photos. It looks like a before-and-after case.

The lower photo shows the TEC-1, screwed to a base-board, with the 8x8 matrix connected to the expansion socket - before he got the bug. The upper photo shows how things grew. It shows only 3 add-ons, out of the 8 he has designed and built.

Inspired by the TEC, he now holds early morning classes for electronics enthusiasts.

He covers both the practical and programming aspects of computers and some of his teaching aids will be presented in the next issue.





The beginning stages of Mr Wragg's TEC. He now has a whole batch of support devices including a Velocity-Acceleration recording interface & a programmed ROM.

We will be passing on may of these developments in future issues.

Programming on the 7-Segment Displays:

We continue from P.29 of issue 11 with more programs on the TEC display.

For this, you will need the TEC-1 or TEC-1A. No 'add-ons' are used at this stage. The displays on the PC provide the readout.

The next program, "**Back and Forth**" runs the 'g' segment across the 6 displays and back again without shifting to the 7th and 8th outputs. Thus the blank output and speaker are not activated.

The number of 'shifts' is determined by the value loaded into register C and 1, 2, 3, 4, 5, or 6 displays can be activated.

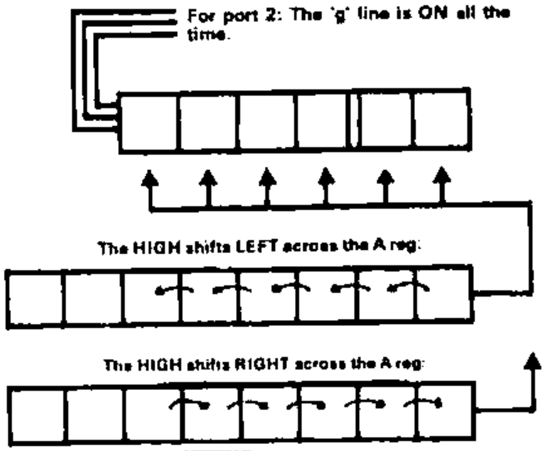
"BACK AND FORTH"

```
LD A,04 ('g' segment)       800   3E 04
OUT (2),A (seg. port)       802   D3 02
LD C,05                     804   0E 05
LD A,01 1st display         806   3E 01
OUT (1),A (cath. port)      808   D3 01
LD B,A                      80A   47
CALL DELAY                  80B   CD 00 09
LD B,A                      80E   78
RLC A                       80F   CB 07
DEC C                       811   0D
JP NZ LOOP                  812   C2 08 08
LD C,06                     815   0E 06
OUT (1),A                   817   D3 01
LD B,A                      819   47
CALL DELAY                  81A   CD 00 09
LD A,B                      81D   78
RRC A                       81E   CB 0F
DEC C                       820   0D
JP NZ LOOP 2                821   C2 17 08
JP START                    824   C3 00 08
```

Delay at 0900:

```
11 FF 07
1B
7B
B2
C2 03 09
C9
```

For Port 1, the cathode of the first display is activated, then the 2nd display, then the third display, etc.


For port 2: The 'g' line is ON all the time.
The HIGH shifts LEFT across the A reg:
The HIGH shifts RIGHT across the A reg:

The diagram shows the shifting of the HIGH across the accumulator (the A register). This is output to port 1 and will turn on one transistor at a time. These transistors feed the commen cathode lines of the displays.

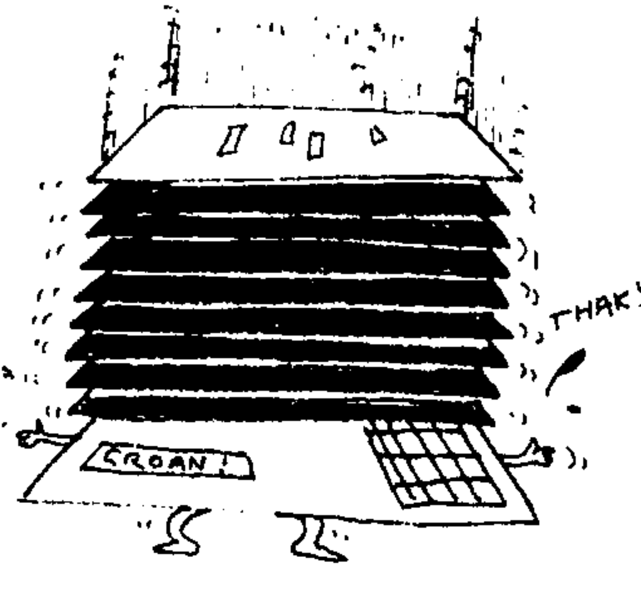
Keep this program in the computer for the next experiment.

QUESTIONS:

1. At 805, why do we insert 05 into register C? Try the value 06 and see what happens. (slow the clock rate to see the effect.)

2. At 805, insert the value 03. What happens?

3. At 805, insert the value 01. Can you see what is happening on the screen and in register C. (The bit is being shifted through the register via the speaker and is appearing on the left hand side of the display.)



AND OUTPUT LATCHES & DRI

Another interesting TEC-1 repair came in this week. The constructor had purchased a set of components from us in 'shart-form'. He had bought some of the items himself and the remainder he purchased from us - things like the EPROM, Z80 and display drivars.

But what was interesting, he had made the printed circuit board himself. Not being satisfied with photocopying the layout on the back of the magazine, he reproduced the entire artwork using tape and stick-on lands. It was copied so exactly that it took us a few minutes to realize the situation. And when it finally dawned, it was quite a shock! It was like looking at a forgery! The work entailed in its creation must have been enormous. And now we had to repair it.

Normally there is not a single firm under the sun which would entertain a repair which had not been made with components and PC board as supplied by the organisation. You can imagine the assortment of repairs which would be sent in.

But because we have had so much success with repairing the computer, we accepted it and went to work.

Theoretically, every track had to be inspected because the board was a 'one-off'.

After quite a few minutes, we noticed two tracks did not connect to the appropriate places. When we joined these with short lengths of tinned copper wire, all the chips received power, but still the computer failed to work.

Then we noticed the clock chip was a Fairchild 4049 and this was promptly replaced.

The display immediately lit up and on going through the keyboard, we found keys 4 and 7 did not give the correct display reading.

Back to more inspection. More detective-work located a track on the common line which should not have been there. When this was cut, the computer worked perfectly.

Here's the lesson:

If you are gaing to repraduce artwork, you must check, re-check and then triple chack the layout. The only effactive way to do this is to make the artwork identical in size and design so that the two layouts can be placed together and held up to the light. Looking from both sides, you will be able to detect any discrepancy in the wiring.

When the board is etched and drilled, go over the entire board with a multimeter and check each of the lines for continuity. Mark each line as you check it so that none are missed.

Only this way can you be sure every line goes to its correct destination.

I know it is fun to produce everything yourself, but for some items the result is not economical. Making PC boards is one of the most time-consuming and costly endevours. After its all finished, it will probably cost nearly as much as a bought one and won't have an overlay or solder mask.

If you have made your own PC board and it doesn't work, please don't send it in for repair. We have met the challenge and don't need it again. It will involve too much of our time and delays the production of the next issue of the magazine.

"ALL THE VALUES"

This program produces all the combinations for the 7-segment display. This will include many unusual effects as well as all tha known letters and numbers.

It is basically an extension to the **BACK and FORTH** program with an additional listing at **0A00** and a change at **824** and **801.**

Fill in this section:

```
3E 00
D3 02
0E 05
3E 01
D3 01
47
CD 00 09
78
CB 07
0D
C2 08 08
0E 06
D3 01
47
CD 00 09
78
CB 0F
0D
C2 17 08
C3 02 0A
```

At 0A00:

```
LD L,01    A00   2E 01
INC L      A02   2C
LD A,L     A03   7D
JP 0802    A04   C3 02 08
```

Delay at 0900:

```
11 FF 07
1B
7B
B2
C2 83 09
C9
```

Push RESET, GO:

Write the ASSEMBLY CODE for the program abova and also the address listings, starting at 0800.

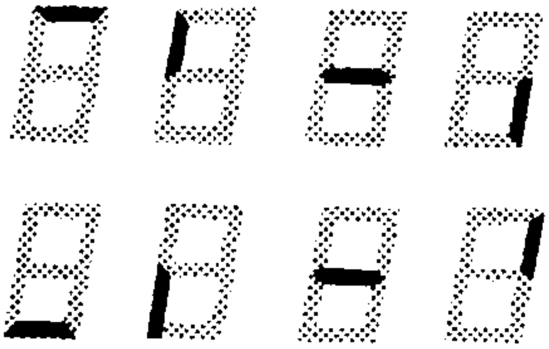
MOVEMENT AROUND A SINGLE 7-SEGMENT DISPLAY:

The following program produces movament around a singla 7-segment display which can be increased in spaad to produce a novel effect.

```
LD A,20      800   3E 20
OUT (1),A    802   D3 01
LD A,01      804   3E 01
CALL 0900    806   CD 00 09
LD A,02      809   3E 02
CALL 0900    80B   CD 00 09
LD A,04      80E   3E 04
CALL 0900    810   CD 00 09
LD A,20      813   3E 20
CALL 0900    815   CD 00 09
LD A,80      818   3E 80
CALL 0900    81A   CD 00 09
LD A,40      81D   3E 40
CALL 0900    81F   CD 00 09
LD A,04      822   3E 04
CALL 0900    824   CD 00 09
LD A,08      827   3E 08
CALL 0900    829   CD 00 09
JP 0800      82C   C3 00 08
```

at 0900:

```
OUT (2),A   900   D3 02
            902   11 FF 07
            905   1B
            906   7B
            907   B2
            908   C2 05 09
            90B   C9
```



This program has not been efficiently written. It contains a repetition of **LD A'** and **'CALL'** instructions. It should have a **BYTE TABLE.** This will be shown in a later program.

However it does contain one byta-saving faature. The statemant **OUT (2),A** is usad at each stage and has been placed in the **CALL ROUTINE.** This saves 14 bytes of program.

1. Replace **(07)** at **904** with **00** and watch the screen.

2. Replace **(FF)** at **903** with **0F.**

3. At address **801** replace the data with **01, 04,** and then **10.** What happens to the display?

4. Raplace data at **801** with **05, 0F, 2D.** What happens to the display in each of these cases?

THE DELAY ROUTINE

We have used a delay routine in many of the programs we have investigated.

We have also saen how it can be adjusted from a few milliseconds to a faw seconds in length.

For this time-delay to occur, many thousands of clock cycles must be involved in its exacution. In fact, up to one million or more cycles can be involved.

Let us look at how this comes about and how the delay operates.

The delay program we will ba investigating is:

```
1.   11 FF 07
2.   1B
3.   7B
4.   B2
5.   C2
```

The meaning of each line is:

1. **LD DE 07FF** FF is loaded into E and 07 into D
2. **DEC DE** Register E is decremented by one. If an **underflow** occurs, register D is dacremented
3. **LD A,E** Register E is loaded into the accumulator.
4. **OR D** The D register is OR-ed with the accumulator
5. **JP NZ** to: The program jumps to line 2 if the result is not zero.

The numbar of cyclas to perform each operation is as follows:

LD DE	**11 FF 07**	10 cycles
DEC DE	**1B**	6 cycles
LD A,E	**7B**	7 cycles
OR D	**B2**	7 cycles
JP NZ Line 2	**C2 03**	10 cycles

One loop consists of:

1B	- 6
7B	- 7
B2	- 7
C2	- 10

total: 30 cycles.

To produce a delay of 256 loops, the instruction is:

11 FF 00
FF is loaded into the E register and 00 into the D register.

E is decremented by one on each loop of the program and when it gets to 00, the result of OR-ing the accumulator (which will contain tha valua of the D ragister - 00) will be zero and the microprocessor will jump out of the dalay routine and back to the main program.

Total clock time for the 256 loops is:
256 x 30
= 7680 cycles.

If the D register is loaded with FF the delay time will be:
7680 x 256
= 1,966,000 cycles.
This is about 2 million cycles!

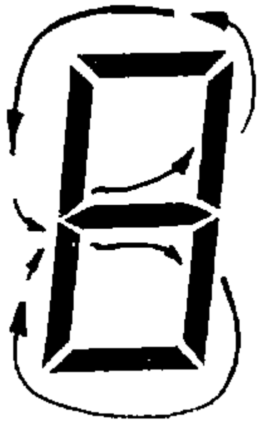
When deciding upon a delay of suitable length, try various values in this location:

11 FF 07

This location has only a small effect on the delay time and can be considered to have a 'trimming effect'.

FIGURE 8's ACROSS THE SCREEN:

The next program introduces a table of values which the program 'looks up' during the execution of each cycle. These are called **'data bytes'** or bytes of data, which are used one at a time.



The LED turn-on sequence around the display.

```
LD C,20         800     0E 20
LD A,C          802     79
OUT (1),A       803     D3 01
CALL 0900       805     CD 00 09
RRC C           808     CB 09
BIT 7, C        80A     CB 79
JP Z LOOP       80C     CA 02 08
JP 800          80F     C3 00 08
```

```
LD HL,0B00      900     21 00 0B
LD B,9          903     06 09
LD A(HL)        905     7E
OUT (2)A        906     D3 02
CALL DELAY      908     CD 00 0A
INC HL          90B     23
DEC B           90C     05
JP NZ 905       90D     C2 05 09
RET             90F     C9
```

At 0A00

```
A00     11 FF 07
A03     1B
A04     7B
A05     B2
A06     C2 03 0A
A09     C9
```

at 0B00:

```
B00     01
B01     02
B02     04
B03     20
B04     80
B05     40
B06     04
B07     08
B08     01
```

On the first pass, the program places 0B00 into a register pair such that the 0B goes into the H register (meaning the High order byte register) and 00 in to the L register (Low order byte register).

At 905 the contents of the byte at 0B00 will be loaded into the Accumulator because this is the address specified by the HL instruction.

On each subsequent pass, the HL register pair in incremented by ONE. Since the value 0B00 is contained in this pair, the result will be to add 1 to 00 to get 01, 02 etc. Thus the program will look up 0B01, 0B02 etc and finds the relevant byte of data.

The second interesting part of this program is the counting of the **DATA BYTE** table. The computer must know how many data bytes are to be accessed.

Thus it is given an intial value of **09** at **904** and decremented the value by one on each pass. When the result is zero, the program jumps to **0800** and starts again.

The segment can be made to travel across the screen in the opposite direction by changing 3 values:

The starting value at 801 must be changed to 01. RRC must be changed to RLC **(CB 01)** and bit 6 must be tested for zero **(CB 71)**

The changes are:

```
800  0E 01
808  CB 01
80A  CB 71
```

CONTROL VIA THE KEYBOARD

Movement on the screen can be controlled by the keyboard by introducing a **HALT** or wait function. This causes the program to halt and wait for an input via the interrupt line.

When a key is pressed, the non-maskable interrupt line is activated and allows the Z80 to accept data from the keyboard encoder via the data bus.

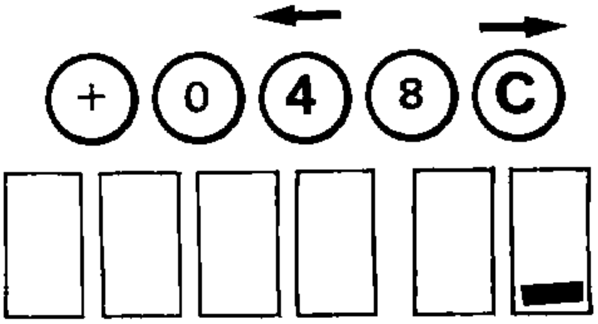
The data is loaded into the accumulator and compared with a value in the program. If the two values are the same, the output is zero and as determined by the next instruction, the program advances.

This program moves a LED across the bottom row.

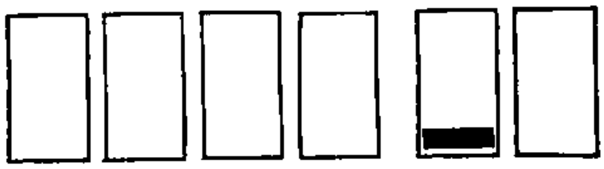
Key '4' shifts the LED left and 'C' shifts it right.

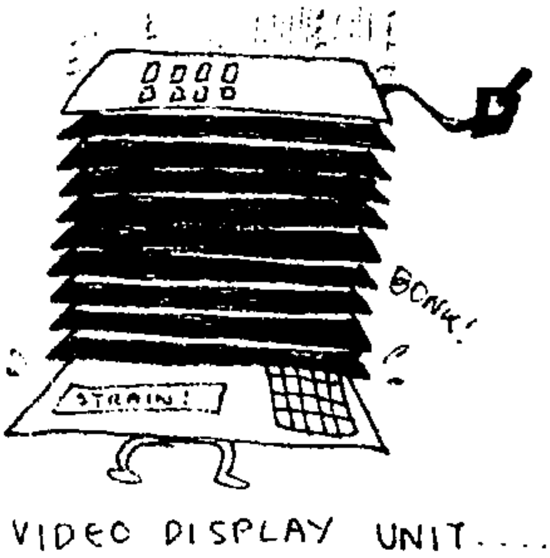
The direction of shift is determined by **RLC B** and **RRC B.** Each press of a button moves the LED one place. No delay routine is required in this program.

```
LD A,04         800     3E 04
OUT (2),A       802     D3 02
LD B,A          804     47
LD A,B          805     78
OUT (1),A       806     D3 01
HALT            808     76
LD A,01         809     ED 57
CP 04           80B     FE 04
JP NZ 815       80D     C2 15 08
RLC B           810     CB 00
JP 805          812     C3 05 08
CP 0C           815     FE 0C
JP NZ 808       817     C2 08 08
RRC B           81A     CB 08
JP 805          81C     C3 05 08
```



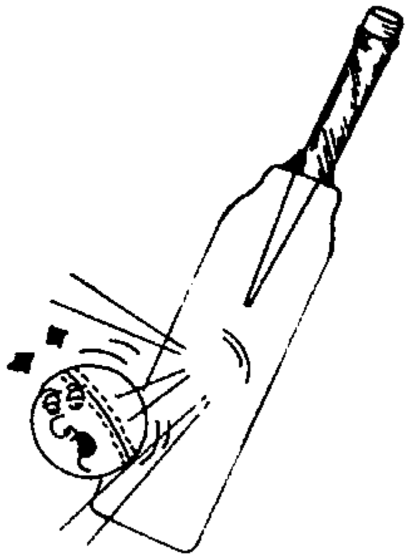
The 'd' segment shifts across the display. The direction is determined by buttons '4' and 'C'.





CREATING A BAT

This program produces a **2-segment** bat capable of travelling across the lower segment of the display. The '+' key moves the bat to the left and the 'C' key moves it to the right.



```
LD A,80        800     3E 80
OUT (2),A      802     D3 02
LD B,03        804     06 03
LD A,B         806     78
OUT (1),A      807     D3 01
HALT           809     76
LD A,I         80A     ED 57
CP 10          80C     FE 10
JP NZ 0816     80E     C2 16 08
RLC B          811     CB 00
JP 806         813     C3 06 08
CP C           816     FE 0C
JP NZ 809      818     C2 09 08
RRC B          81B     CB 08
JP 806         81D     C3 06 08
```

WRITING A PROGRAM

This is a written exercise requiring YOU to write a program. Our aim will be to write a BAT program exactly like the previous program and you can refer to it if a problem arises.

For each line, the MACHINE-CODE value should be obtained from the Z80 CODE SHEET on the back page of issue 11. It should then be placed in the space provided.

1. Load the accumulator with the value 80. Answer: ________
2. Output the contents of the accumulator to port 2. ________
3. Load register B with the value 3: ________
4. Load register B into the accumulator: ________
5. Output the accumulator to port 1: ________
6. Halt the program: ________

The Z80 is now waiting for an interrupt.

7. Load the index register into the accumulator: ________
8. Compare the accumulator with the value 10: ________
9. Jump to 'COMPARE C' (below) if the answer to line 8 is NOT zero: ________
10. Rotata register B LEFT CIRCULAR: ________
11. Jump to the address which states: Load B into A (above): ________
12. Compare the accumulator with the value 'C': ________
13. Jump to HALT (above) if NOT zero: ________
14. Rotate register C Right Circular: ________
15. Jump to: Load register B into A (above): ________

Complete the following listing by adding the values you have obtained from the statements above:

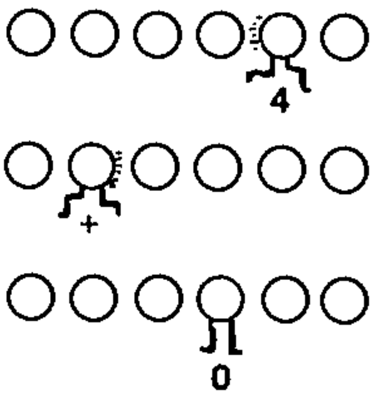
```
800
802
804
806
808
80A
80B
80D
80F
812
814
815
817
81A
81D
81F
822
824
825
827
82A
```

AUTO MOVEMENT & HALT

S Riley,
Guildford, 2161.

The following program detects 3 keys. The '+' key shifts the LED left, the '0' key stops the LED and key '4' shifts it right.

The speed of travel across the display is controlled by the DELAY ROUTINE.



```
LD A,01        800     3E 01
OUT (2),A      802     D3 02
LD A,01        804     3E 01
OUT (1),A      806     D3 01
LD B,01        808     06 01
HALT           80A     76
LD A,I         80B     ED 57
CP '+'         80D     FE 10
JP NZ 081D     80F     C2 1D 08
RLC B          812     CB 00
LD A,B         814     78
OUT (1),A      815     D3 01
CALL DELAY     817     CD 00 0C
JP 80B         81A     C3 0B 08
CP 04          81D     FE 04
JP NZ 80D      81F     C2 0D 08
RRC B          822     CB 08
LD A,B         824     78
OUT (1),A      825     D3 01
CALL DELAY     827     CD 00 0C
JP 808         82A     C3 0B 08

   at 0C00:            11 FF 0A
                       1B
                       7B
                       B2
                       C2 03 0C
                       C9
```

So far we have turned on one segment or LED at a time in the display or more than one segment or LED within the same digit. But not 2 LEDs in different displays, in different positions.

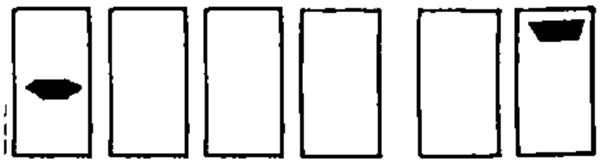
This seems impossible but by using a clever pulsing technique we can alternately access one then the other to produce the effect of both being on at the same time.

In this program we will alternately access segment 'g' in the first display and segment 'a' in the 6th display to give the appearance that they are both on at the same time.

SWITCHING 2 PIXELS INDEPENDENTLY:

Run the following program and observe the effect:

```
LD A,20        800     3E 20
OUT (1),A      802     D3 01
LD A,01        804     3E 04
OUT (2),A      806     D3 02
CALL DELAY     808     CD 00 0B
JP 0A00        80B     C3 00 0A
```



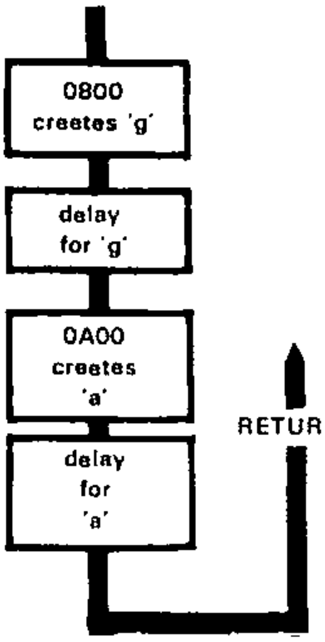
```
LD A,01        A00     3E 01
OUT (1),A      A02     D3 01
LD A,01        A04     3E 01
OUT (2),A      A06     D3 02
CALL DELAY     A08     CD 00 0B
JP 0800        A0B     C3 00 08
```

at 0B00:

```
                    11 FF 00
                    1B
                    7B
                    B2
                    C2 03 0B
                    C9
```

Turn the speed control up and the effect is two different LEDs being lit at the same time. Turn the speed control down and the alternating effect becomes more noticeable.

The flow diagram for this is:



Insert **05**into the delay routine at **0B02** and watch the display. The alternating effect is more obvious. This is the basis for all the letters and writing on the display. Each digit is being turned on and off very quickly.

PROBLEMS:

Turn the speed control up and keep the delay routine short for the following problems:

1. Change values in the program to turn on segment 'd' in the first display and 'a' in the sixth display.

2. Create the figure 1 in the first display and '0' in the last display. Which locations in the program must be altered to achieve this?

TO CONTROL 2 PIXELS. One with movement.

This program produces two pixels. One is fixed and the other moves up and down.

In this experiment, the main program is at **0A00** and it calls the delay at **0B00** and a short routine at**0800**. When the program has been entered, push, RESET, ADdress, 0A00, GO, GO to execute the program. The main task with this experiment will be to rewrite the main program so that it appears at 0800. This will involve changing a number of machine code values to suit the new location.

```
LD C,BB (or any value)   0A00   0E BB
LD A,01                  0A02   3E 01
OUT (1),A                0A04   D3 01
LD A,01                  0A06   3E 01
OUT (2),A                0A08   D3 02
CALL DELAY               0A0A   CD 00 0B
CALL 0800                0A0D   CD 00 08
DEC C                    0A10   0D
JP NZ 0A02               0A11   C2 02 0A
LD C,BB                  0A14   0E BB
LD A,01                  0A16   3E 01
OUT (1),A                0A18   D3 01
LD A,04                  0A1A   3E 04
OUT (2),A                0A1C   D3 02
CALL DELAY               0A1E   CD 00 0B
CALL 0800                0A21   CD 00 08
DEC C                    0A24   0D
JP NZ 0A16               0A25   C2 16 0A
LD C,BB                  0A28   0E BB
LD A,01                  0A2A   3E 01
OUT (1),A                0A2C   D3 01
LD A,80                  0A2E   3E 80
OUT (2),A                0A30   D3 02
CALL DELAY               0A32   CD 00 0B
CALL 0800                0A35   CD 00 08
DEC C                    0A38   0D
JP NZ 0A2A               0A39   C2 2A 0A
JP 0A00                  0A3C   C3 00 0A
```

Delay at 0B00:

```
                                11 FF 00
                                1B
                                7B
                                B2
                                C2 03 0B
                                C9
```

```
LD A,20        800   3E 20
OUT (1),A      802   D3 01
LD A,01        804   3E 80
OUT (2),A      806   D3 02
CALL DELAY     808   CD 00 0B
RETURN         80B   C9
```

Problem:

1. Rewrite the MAIN PROGRAM to start at 0800:

```
LD C,BB        800
LD A,01        802
OUT (1),A      804
LD A,01        806
OUT (2),A      808
CALL DELAY     80A
CALL 0A00      80D
DEC C          810
JP NZ 0802     811
LD C,BB        814
LD A,01        816
OUT (1),A      818
LD A,04        81A
OUT (2),A      81C
CALL DELAY     81E
CALL 0A00      821
DEC C          824
JP NZ 0816     825
LD C,BB        828
LD A,01        82A
OUT (1),A      82C
LD A,80        82E
OUT (2),A      830
CALL DELAY     832
CALL 0A00      835
DEC C          838
JP NZ 082A     839
JP 0800        83C
```

Fill in the MACHINE CODE values:

at 0B00:
Same Machine code values for delay.

```
                    0B00
                    0B03
                    0B04
                    0B05
                    0B06
                    0B09
```

at 0A00:

```
LD A,20        A00
OUT (1),A      A02
LD A,01        A04
OUT (2),A      A06
CALL DELAY     A08
RETURN         A0B
```

Run the program by pressing RESET, GO. Does it work? (It should)

2. Insert the following data into the program you have written, to produce the name of a semiconductor:

at 0806: 3E 47

at 081A: 3E C7

at 082E: 3E C6

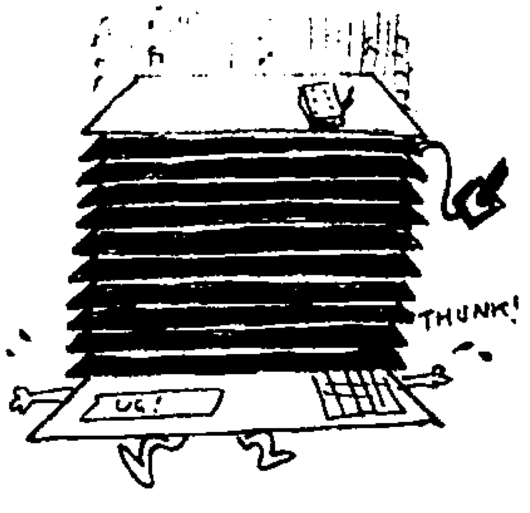
What is the name of the device?

3. Create the name of another semiconductor device by inserting the following information into the program:

```
802     3E 10
806     3E 4F
816     3E 08
81A     3E EA
82A     3E 04
82E     3E C6
```

Remove the value **20** at **0A01.**
Push RESET, GO.
What is the name of the device?
Reduce the delay of **BB** in the MAIN PROGRAM (at 3 locations) to **05.**.

The result will be your first readable multiplexed word.



AND AN EPROM BURNER..

JUMPS AND CALL INSTRUCTIONS

JUMP and CALL instructions are called **BRANCH INSTRUCTIONS.**

They cause the program to branch to another location in memory and execute the instruction contained at that location.

The 6 instructions we will investigate are:

JP Address	**C3 XX XX**
JP NZ Address	**C2 XX XX**
JR dis	**18 XX**
JR NZ dis	**20 XX**
CALL Address	**CD XX XX**
CALL NZ Address	**C4 XX XX**

The meaning of each instruction is as follows:

JP Address. This is an unconditional instruction . It means **Jump: Address.** The program will jump to a new address as determined by the next two bytes XX XX.

JP NZ Address. This means **Jump, non-zero: address.** The program will only jump to a new address if the result of the previous instruction is **NOT** zero. (If the result is zero, the program will neglect this 3-byte instruction and advance to the next instruction).

JR dis. This is an unconditional statement. It means: **Jump relative: displacement.**

In simple terms a relative jump means the program will jump to an address of plus 129 bytes or minus 126 bytes of the address of the JR op-code byte.

For instance, the value **FB** will cause a jump to **1B** in the following program.

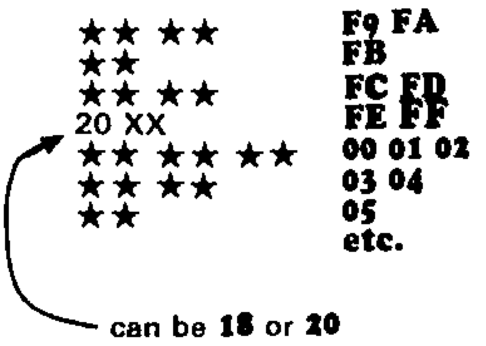
```
11 FF 07
1B
7B
B2
18 FB
C9
```

For a forward jump, **03**will cause the program to jump to **D3** in the following:

```
3E 01
18 03
3E 20
D3 01
3E 28
```

JR NZ dis. This is a conditional statement. It means **Jump relative, non-zero: displacement.** The displacement is given by a hex value such as **D7, EE, F8** for a backward jump or **07, 18, 44, 76,** for a forward jump.

When determining the displacement value, this is an easy method:



can be **18** or **20**

CALL Address. This is an unconditional instruction. It means **CALL the address given by the next two bytes XX XX.**

When using this instruction, it must be the intention of the programmer to call a sub-routine and then return to the instruction which immediately follows, as this is the requirement of the microprocessor.

For this reason, the sub-routine must conclude with a ratum instruction **C9.** The address of the byte immediately following **CD XX XX** will be saved in the stack. At the conclusion of the sub-routine it will ba **popped** off the stack, looked at, and cause the program to return to the instruction after **CD XX XX.**

CALL NZ Address. This is a conditional instruction and will only be executed if the result of the previous instruction is **NOT** zero. All other features of this instruction as per CALL Address above.

The main differences between these three sets of jump instructions are:

A **JP** instruction causes the program to go to a sub-routine but does not call it back again.

A **JP** instruction can make the program go to any location in memory. It is not restricted to a displacement value.

A **JP** instruction cannot be re-located without changing or looking at the two-byte jump address to see if the sub-routine is still at the same address.

A **JR** can only operate within +127 and -128 bytes (approx.)

JR can be easily re-located as it relates only to relative memory. This type of instruction is idaal when large portions of a program need to be shifted.

CALL instructions are used when a sub-routine is required to be executed (such as a delay) followed by a return to the main program.

QUESTIONS

1. Write the meaning of these, in words:

(a) **JP**
(b) **JP NZ**
(c) **JR dis**
(d) **JR NZ dis**
(e) **CALL**
(f) **CALL NZ**

2. Which instruction would you use for the following:
(a) You require to go to a sub-routine and then return to the main program.
(b) You require to go to another routine if the answer to the previous line is NOT zero.
(c) You require to go to the beginning of the program.
(d) You require to go to a locetion about 15 bytes further down the program.
(e) You require to go to a sub-routine on the condition NON-zero, and return.
(f) You require to go to a location back 8 bytes.

3. Give one advantage of a **JUMP RELATIVE** instruction, compared to a **JUMP** instruction.

4. To produce a loop in a program, which of the following should be used: **JR dis or JR NZ dis.**

5. At the end of a program, which instruction should be used: **CALL, JR NZ, JP.**

6. What is the difference betwaen **CALL** and **JUMP**?

Answers:

1. Jump.
Jump Non-zero.
Jump Relative displacement.
Jump relative non-zero displacement
Call.
Call Non-zero.

2. (a) **CALL**
(b) **JP NZ**
(c) **JP**
(d) JR 14
(e) **CALL NZ**
(f) **JR F6**

3. The program can be transferred to another location without affecting the **JUMP RELATIVE** instruction.

4. It must have a non-zero condition.

5. It must be a **JUMP** instruction with no other conditions.

6. **CALL** transfers the program to another location and requires that it be returned to the next instruction after the sub-routine has been performed. **JP** transfers the program to another location without any return requirement.

USING JR's

To show how we can substitute a **JR** instruction for a **JUMP** instruction, we will consider a simple program containing a delay routine.

We will choose the program: **RUNNING SEGMENT 'a' ACROSS THE SCREEN**. This can be found on P. 26 of issue 11 and is repeated here:

Type this into the computer and **RUN**.

at 0800:

```
LD A,01        800  3E 01
OUT (2),A      802  D3 02
LD B,01        804  06 01
LD A,B         806  78
OUT (1),A      807  D3 01
CALL DELAY     809  CD 00 0A
RLC B          80C  CB 00
JP LOOP        80E  C3 06 08

          0A00     11 FF FF
                   1B
                   7B
                   B2
                   C2 03 0A
                   C9
```

1. We will change the instruction at **80E** to **JR 806** . Change the address values to **18 F6.**

Place this into the program and **RUN**. Is any difference observed? (There should be no difference).

2. The delay routine at **0A00** can also be changed to include a **JR** instruction.

at **A06:**
change **C2 03 0A** to **20 FB 00**

Run the program and note the result. No difference should be detected. Both instruction perform the same in this case.

3. The **DELAY PROGRAM** can be placed immediately below the main program so that a **JR** instruction can be used at **809** and also at the end of the delay.

at **809:** insert **JR 820.**

Start the delay routine at **0820.**

At the end of the delay routine, insert: **JR 80C.**

The displacement values will have to be worked out by you. Follow through the steps as shown and write the complete program. Use the TEC as a counter to work out the displacement values (by pushing +,+,+,+,+, etc.)

Do not look at the answer at the bottom of the next column until you have finished.

Next issue we will give a **JR table** and explain how it is used.

QUICK DRAW

In this final exercise we will change a number of **JUMP** instructions to **JR** instructions. See the **QUICK DRAW** program on P. 13 of issue 11.

This is how to change the program:

1. Copy all the assembly code, replacing **JP** with **JR.**

2. Copy the machine code listing, remembering that the 3-byte **JP** instructions will become 2-byte **JR** instructions. At this stage do not insert the displacement values - this will be the final job.

3. Insert the displacement values for each of the **JR** instructions.

4. Fill in the memory locations, starting at **0800**.

Push RESET, GO and play the Quick Draw game. Does everything work correctly? It should.

We have learnt the major advantage of a **JR** instruction. It enables a program to be transferred to another location without having to alter any of the data.

See the effectiveness of this. Move the whole Quick Draw game to **0900** or **0A00** making sure you wipe the program at**0800** before starting at the new location.

To start the game, push RESET, GO, GO. Is it a success?

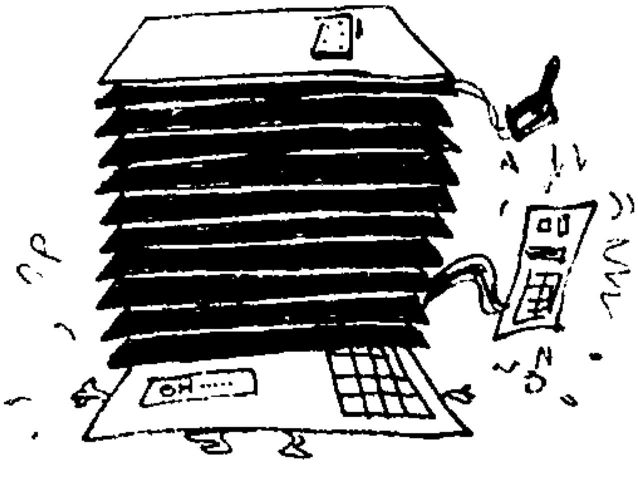
This is what your program should look like:

QUICK DRAW

```
        LD A,00         800  3E 00
        OUT (1),A       802  D3 01
START   LD DE,00        804  11 00 00
DELAY   DEC DE          807  1B
        LD A,D          808  7A
        OR E            809  B3
        JR NZ Delay     80A  20 FB
        LD A,E3         80C  3E E3
        OUT (2),A       80E  D3 02
        LD A,08         810  3E 08
        OUT (1),A       812  D3 01
LOOP 1  HALT            814  76
        AND OF          815  E6 0F
        CP 0C           817  Fe 0C
        JR Z,Right      819  28 05
        OR A            81B  B7
        JR Z,Left       81C  28 06
        JR Loop 1       81E  18 F4
RIGHT   LD A,01         820  3E 01
        JR Finish       822  18 02
        LD A,20         824  3E 20
        OUT (1),A       826  D3 01
LEFT    LD A,20         828  3E 28
FINISH  OUT (2),A       82A  D3 02
        HALT            82C  76
        JR Start        82D  18 D1
```

Your final program will look like this:

```
LD A,01        800     3E 01
OUT (2),A      802     D3 02
LD B,01        804     06 01
LD A,B         806     78
OUT (1),A      807     D3 01
JR 820         809     18 14
RLC B          80B     CB 00
JR 806         80D     18 F7


               820     11 FF FF
               823     1B
               824     7B
               825     B2
 JR NZ 823     826     20 FB
 JR 80B        828     18 E0
```



...AND REMOTE CONTROL....

TONES & TUNES

Here is e selection of tones and tunes for the computer. These have been submitted by readers and are presented in various formats to get you acquainted with the different ways of presenting a program.

OSCILLATOR

by Peter Alekseievs

```
800   3E  LD A,80
801   80
802   D3  OUT (01),A
803   01
804   3E  LD A,00
805   00
806   D3  OUT (01),A
807   01
808   C3  JP 800
809   00
80A   08
```

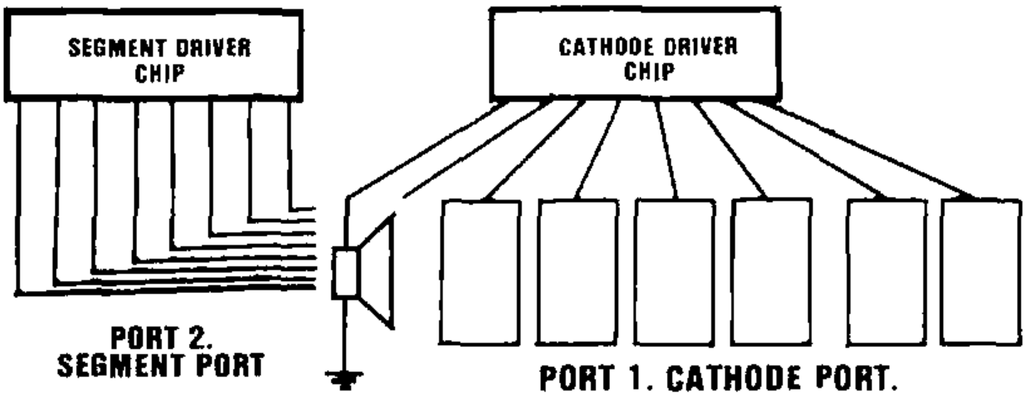
The principle of operation of this program can be seen in the diagram. We are accessing the speaker via port 1 and this is the 8th line of the driver chip. Thus the value 80 is inserted in the program.

We load a HIGH into this line for a number of clock cycles and then a LOW. This produces a CLICK which sounds like an oscillator when the speed control is increased.

TOCCATA

-by Stephen Clarke, 2774.

```
800: 00 00 3E 00 32 00 08 3E 09 32 01 08 CD B0 01 CD
810: B0 01 3E 21 32 00 08 CD B0 01 3E 62 32 00 08 CD
820: B0 01 C3 02 08

900: 0A 0F 11 12 0F 11 12 14 11 12 14 16 12 14 16 17
910: 14 16 12 14 11 12 0F 11 0E 0F 0A 0B 08 0A 0A 00
920: 1F 00 17 17 17 17 16 16 16 16 14 14 14 14 12 12
930: 12 12 11 11 11 11 0F 0F 0F 0F 0E 0E 0E 0E 0E 0E
940: 0E 0E 0B 0B 0B 0B 0A 0A 0A 0A 08 08 08 08 06 06
950: 06 06 05 05 05 05 03 03 03 03 02 02 02 02 02 02
960: 00 1F 0A 0F 11 12 0F 11 12 14 11 12 14 16 12 14
970: 16 17 14 16 12 14 11 12 0F 11 0E 0F 0A 0B 08 0A
980: 08 06 05 03 03 03 03 03 03 03 03 03 00 00 00 00
990: 00 00 00 00 00 00 1F
```



PHONE RING

By Cris Cogdon.

This program generates a ring similar to that of a new phone. It would make an ideal trick if you have one of these phones!

```
START  CALL RING      800   CD 14 08
       LD HL,1000     803   21 00 10
       CALL PAUSE     806   CD 1E 08
       CALL RING      809   CD 14 08
       LD HL,8000     80C   21 00 80
       CALL PAUSE     80F   CD 1E 08
       JR START       812   18 EC

RING   LD B,10        814   06 10
XRING  PUSH BC        816   C5
       CALL 081E      817   CD 8E 01
       POP BC         81A   C1
       DJ NZ XRING    81B   10 F9
       RETURN         81D   C9

PAUSE  DEC HL         81E   2B
       LD A,H         81F   7C
       OR L           820   B5
       RET Z          821   C8
       JR PAUSE       822   18 FA
```

FREQUENCY SWEEP

by Peter Alekseievs

This program gives an effect similar to a phaser gun. By changing the value of the second byte, different effects can be generated.

This program can be placed anywhere in memory as it consists entirely of **JR** instructions.

```
LD H,FF      26 FF
LD B,H       44
LD A,00      3E 00
OUT (1),A    D3 01
LD A,80      3E 80
OUT (1),A    D3 01
LD A,B       78
DEC A        3D
JR NZ FD     20 FD
DJNZ F2      10 F2
LD B,00      06 00
LD A,00      3E 00
OUT (1),A    D3 01
LD A,80      3E 80
OUT (1),A    D3 01
INC B        04
LD A,B       78
DEC A        3D
JR NZ FD     20 FD
LD A,H       7C
SUB B        90
JR NZ EF     20 EF
JP DA        18 DA
```

THE STRIPPER

```
800: 00 00 3E 00 32 00 08 3E 09 32 01 08 CD B0 01 C3
810: 02 08

900: 01 01 03 03 03 06 06 06 06 06 06 0A 08 08 06 06
910: 06 02 02 02 02 02 02 01 01 03 03 03 06 06 06 06
920: 06 06 0A 0D 0D 0C 0C 0C 0B 0B 0B 0B 0B 0B 0B 0A
930: 0A 01 01 01 01 0A 0A 01 01 01 01 0A 0A 09 0A 0A
940: 0A 0B 0B 0A 0B 0B 0C 0D 0D 0C 0D 0D 0D 0E 0E 0D
950: 0E 0E 0E 0F 0F 0E 0F 0F 11 11 11 0F 11 11 12 12
960: 12 12 12 12 12 00 00 00 00 00 00 00 00 00 00 1F
```

TEC CLOCK

This program will allow the TEC to be used as a CLOCK. The display is used as the readout and the time can be sat as shown opposite.

This is a 24 hour clock and its accuracy depends on the setting of the SPEED CONTROL. In a future issue we will present a crystal oscillator to take the place of the 4049 to turn the TEC into an accurate time-piece.

To set CLOCK:

at 989: insert saconds
at 98A: insert minutes.
at 98B: insert hours.

Example: 7:45:32

989: 32 98A: 45 98B: 07

```
START   LD IY, Clock Buffer        900   FD 21 89 09   Load pointer to clock counting buffer
        LD B,2                     904   06 02         load number of 60's to be tested
        LD A,(IY +0)               906   FD 7E 00      Read first clock buffer value
        ADD A,01                   909   C6 01         add 1 to the value
        DAA                        90B   27            decimal adjust the accumulator
        CP 60                      90C   FE 60         TEST A=60  sec/min
        JR NZ,DISP                 90E   20 13         GOTO 'DSP' if not equal
        XOR A                      910   AF            ZERO the accumulator
        LD (IY+0),A                911   FD 77 00      Store A in clock buffer
        INC IY                     914   FD 23         Advance pointer
        DJNZ EE                    916   10 EE         complete LOOP if B is not zero
        LD A,(IY +0)               918   FD 7E 00      Read hours value
        ADD A,01                   91B   C6 01         Increment hours value
        DAA                        91D   27            Decimal adjust the eccumulator
        CP 24H                     91E   FE 24         TEST hours =24
        JR NZ,DISP                 920   20 01         If not GOTO 'DSP'
        XOR A                      922   AF            ZERO A
DISP    LD (IY + 0),A              923   FD 77 00      Store hours in clock buffer
        LD B,03                    926   06 03         Load number of bytes to be convertad
        LD HL,DISP BUF +6          918   21 91 09      Load pointer to display buffer
        LD IX,CLK BUF              92B   DD 21 89 09   Load pointed to clock buffer
LOOP 1  LD A,(IX + 0)              92F   DD 7E 00      Read CLOCK BUFFER value
        INC IX                     932   DD 23         Advance pointer by 1
        PUSH BC                    934   C5            Save BC contants
        PUSH AF                    935   F5            Sava contents of A
        AND 0F                     936   E6 0F         Get laast significant 4 bits
        LD B,A                     938   47            Transfer A to B
        CALL LOOK                  939   CD 73 09      Get pattern for B
        POP AF                     93C   F1            Restore AF
        SRL A                      93D   CB 3F         Shift A one place to the right
        SRL A                      93F   CB 3F
        SRL A                      941   CB 3F
        SRL A                      943   CB 3F
        LD B,A                     945   47            Load A into B
        CALL LOOK                  946   CD 73 09      Get bit pattarn for B
        POP BC                     949   C1            Restore BC
        DJNZ LOOP                  94A   10 E3         Complete LOOP if B is not zero.
        LD B,0FFH                  94C   06 FF         Load LOOP value
LOOP 2  LD IX,DISP BUF             94E   DD 21 8C 09   Load pointer to digit patterns
        PUSH BC                    952   C5            Save BC contents
        LD B,07                    953   06 07         Load number of digits to be displayed
        LD C,40H                   955   0E 40         Load bit pattern for displey cathodes
        LD A,(IX +0)               957   DD 7E 00      Read display pattern
        OUT (2),A                  95A   D3 02         Output pattern to port 2
        LD A,C                     95C   79            Load C into A
        OUT (1),A                  95D   D3 01         Output cathode pattern to port 1
        SRL C                      95F   CB 39         Move cathode bit one place for MUX effect
        XOR A                      961   AF            Clear A
        LD E,10H                   962   1E 10         Load TIME DELAY value
        DEC E                      964   1D            Decrement E
        JR NZ FD                   965   20 FD         LOOP if not equal to zero
        OUT (1),A                  967   D3 01         Turn off anode bits
        INC IX                     969   DD 23         Advance to next pattern
        DJNZ loop 2                96B   10 EA         LOOP if not zero
        POP BC                     96D   C1            Restore BC
        DJNZ LOOP 2                96E   10 DE         LOOP if all digits not displayed
        JP START                   970   C3 00 09      Jump to START

LOOK    LD DE, DISP                973   11 7F 09      Load DE with display pattern
        PUSH AF                    976   F5            Save AF
        LD A,E                     977   7B            Load E into A
        ADD A,B                    978   80            Calculate pattern address
        LD E,A                     979   5F            Load A into E
        LD A,(DE)                  97A   1A            Read pattern
        LD (HL),A                  97B   77            Store pattern in display buffer
        DEC HL                     97C   2B            Decrement HL
        POP AF                     97D   F1            Restore AF
        RETURN                     97E   C9            End of sub-routine

        DISP PATTERN: EB, 28, CD,  97F
          AD, 2E, A7, E7, 29, EF, AF.
        CLOCK BUFFER               989
        DISP BUFFER                98C
```

SPIROID ALIENS

·by M Allison, 3095

This is quite a long program and shows the length of listing required to achieve a degree of realism. The game uses all of page **0800** and portions of **0900, 0A00, 0B00** and **0D00.**

The main program is at **0800** with calls at the other pages.

The game consists of unusual-shaped aliens passing across the display. Each game consists of 16 passes and you must shoot down the arrivals by pressing buttons 1, 2 or 3. To win, you must shoot down at least 11.

In the initial stages of the game, you must acquaint yourself with the connection betweeen the spiroid shapes and buttons 1, 2, 3. After this you will be ready to launch an attack.

Here's the listing:

```
Reserved for
message.          800  00 08
Blank             802  00
LD HL,903         803  21 03 09
LD A,80           806  3E 80
LD (HL),A         808  77
INC HL            809  23
LD A,00           80A  3E 00
LD C,A            80C  4F
LD (HL),A         80D  77
LD HL,0911        80E  21 11 09
LD A,20           811  3E 20
LD (HL),A         813  77
INC HL            814  23
LD A,00           815  3E 00
LD (HL),A         817  77
LD HL, 0849       818  21 49 08
LD A,01           81B  3E 01
LD (HL),A         81D  77
LD B,10           81E  06 10
LD D,00           820  16 00
LD IY,0865        822  FD 21 65 08
CALL 0D00         826  CD 00 0D
LD A,(0865)       829  3A 65 08
CP C              82C  B9
JR Z, 0826        82D  28 F7
LD HL,0849        82F  21 49 08
                  832  00
CP 01             833  FE 01
JR Z,083F         835  28 08
CP 02             837  FE 02
JR Z,0843         839  28 08
LD A,61           83B  3E 61
JR, 0845          83D  18 06
LD A,0F           83F  3E 0F
JR 0845           841  18 02
LD A,26           843  3E 26
LD (084D),A       845  32 4D 08
LD A( )           848  3E ( )
OUT A,(01)        84A  D3 01
LD A,(SYMBOL)     84C  3E ( )
OUT A,(02)        84E  D3 02
CALL 0900         850  CD 00 09
CALL 090E         853  CD 0E 09
XOR,A             856  AF
IN A(00)          857  DB 00
```

```
LD E,A            859  5F
LD A,72           85A  3E 72
CP E              85C  BB
JR Z,0861         85D  20 02
LD E,00           85F  1E 00
LD A,E            861  7B
AND 03            862  E6 03
CP (CODE)         864  FE ( )
JR Z,0871         866  28 09
SLA (HL)          868  CB 26
LD A,40           86A  3E 40
CP (HL)           86C  BE
JR Z,0878         86D  28 09
JR 0848           86F  18 D7
LD C,A            871  4F
INC D             872  14
CALL 0B00         873  CD 00 0B
JR 087D           876  18 05
LD C,00           878  0E 00
CALL 0A00         87A  CD 00 0A
LD A,01           87D  3E 01
LD (HL),A         87F  77
DJNZ 0822         880  10 A2
LD HL,0807        882  21 07 08
LD A,0B           885  3E 0B
CP D              887  BA
JR C,08A5         888  38 1B
LD A,(HL)         88A  7E
CP F0             88B  FE F0
JR Z,0842         88D  28 03
ADD A,10          88F  C6 10
LD (HL),A         891  77
LD HL,0800        892  21 00 08
LD A,F8           895  3E F8
LD (HL),A         897  77
PUSH HL           898  E5
CALL 01B0         899  CD B0 01
POP HL            89C  E1
LD A,E8           89D  3E E8
LD (HL),A         89F  77
CALL 0270         8A0  CD 70 02
JR 08BE           8A3  18 19
LD A,(HL)         8A5  7E
CP 10             8A6  FE 10
JR Z,08AD         8A8  28 03
SUB 10            8AA  D6 10
LD (HL),A         8Ac  77
LD HL,0800        8AD  21 00 08
LD A,DE           8B0  3E DE
LD (HL),A         8B2  77
PUSH HL           8B3  E5
CALL 01B0         8B4  CD B0 01
POP HL            8B7  E1
LD A,CA           8B8  3E CA
LD (HL),A         8BA  77
CALL 0270         8BB  CD 70 02
LD A,3F           8BE  3E 3F
OUT A,(01)        8C0  D3 01
LD A,8A           8C2  3E 8A
OUT A,(02)        8C4  D3 02
HALT              8C6  76
JP 0802           8C7  C3 02 08
                  8CA  00
Messages:   ↓     8CB  01
                  8CC  0C
                  8CD  09
                  8CE  05
                  8CF  0D
                  8D0  12
                  8D1  00
                  8D2  04
                  8D3  05
                  8D4  12
                  8D5  13
                  8D6  11
                  8D7  0E
```

```
                  8D8  16
                  8D9  05
                  8DA  04
                  8DB  1A
                  8DC  00
                  8DD  1F
                  8DE  04
                  8DF  00
                  8E0  04
                  8E1  00
                  8E2  04
                  8E3  00
                  8E4  01
                  8E5  01
                  8E6  01
                  8E7  1F
                  8E8  00
                  8E9  05
                  8EA  0D
                  8EB  04
                  8EC  00
                  8ED  0E
                  8EE  06
                  8EF  00
                  8F0  05
                  8F1  01
                  8F2  11
                  8F3  13
                  8F4  08
                  8F5  1A
                  8F6  00
                  8F7  1F
                  8F8  01
                  8F9  1A
                  8FA  01
                  8FB  1A
                  8FC  01
                  8FD  1A
                  8FE  1F

PUSH AF           B00  F5
PUSH BC           B01  C5
PUSH DE           B02  D5
PUSH HL           B03  E5
LD HL(0903)       B04  2A 03 09
PUSH HL           B07  E5
LD HL(0911)       B08  2A 11 09
                  B0B  E5
LD HL,0912        B0C  21 12 09
LD A,00           B0F  3E 00
LD(HL),A          B11  77
DEC HL            B12  2B
LD A,20           B13  3E 20
ld (HL),A         B15  77
LD HL,0904        B16  21 04 09
LD A,00           B19  3E 00
LD(HL),A          B1B  77
DEC HL            B1C  2B
LD A,24           B1D  3E 24
LD (HL),A         B1F  77
LD HL,0B35        B20  21 35 0B
LD A,01           B23  3E 01
LD (HL),A         B25  77
EXX               B26  D9
LD DE,0904        B27  11 04 09
LD C,00           B2A  0E 00
LD HL,0B68        B2C  21 68 0B
LD B,06           B2F  06 06
LD A,01           B31  3E 01
LD (DE),A         B33  12
LD A,01           B34  3E 01
OUT (01),A        B36  D3 01
LD A,(HL)         B38  7E
OUT (02),A        B39  D3 02
```

```
CALL 0900       B3B   CD 00 09
LD A,00         B3E   3E 00
LD (DE),A       B40   12
CALL 090E       B41   CD 0E 09
INC HL          B44   23
DEC DE          B45   1B
EX DE,HL        B46   EB
DEC (HL)        B47   35
EX DE,HL        B48   EB
INC DE          B49   13
DJNZ 0B31       B4A   10 E5
EXX             B4C   D9
SLA (HL)        B4D   CB 26
EXX             B4F   D9
INC C           B50   0C
LD A,06         B51   3E 06
CP C            B53   B9
JP Z,0B5A       B54   CA 5A 0B
JP 082C         B57   C3 2C 0B
Exx             B5A   D9
POP HL          B5B   E1
LD (0911),HL    B5C   22 11 09
POP HL          B5F   E1
LD (0903),HL    B60   22 03 09
POP HL          B63   E1
POPDE           B64   D1
POPBC           B65   C1
POPAF           B66   F1
RETURN          B67   C9
                B68   01
   LOOK-UP      B69   09
   TABLE        B6A   29
   FOR          B6B   A9
   SPIRAL       B6C   E9
                B6D   EB


PUSH AF         A00   F5
PUSH BC         A01   C5
PUSH HL         A02   E5
LD HL(0903)     A03   2A 03 09
PUSH HL         A06   E5
LD HL(0911)     A07   2A 11 09
PUSH HL         A0A   E5
LD B,09         A0B   06 09
LD HL,0911      A0D   21 11 09
LD A,05         A10   3E 05
LD (HL),A       A12   77
INC HL          A13   23
LD A,00         A14   3E 00
LD (HL),A       A16   77
LD HL,0903      A17   21 03 09
LD A,1F         A1A   3E 1F
LD (HL),A       A1C   77
INC HL          A1D   23
LD A,00         A1E   3E 00
LD (HL),A       A20   77
DEC HL          A21   2B
CALL 090E       A22   CD 0E 09
DEC (HL)        A25   35
LD A,01         A26   3E 01
CP (HL)         A28   BE
JP Z 0A2F       A29   CA 2F 0A
JP 0A22         A2C   C3 22 0A
DJNZ 0A1A       A2F   10 E9
POP HL          A31   E1
LD (0911),HL    A32   22 11 09
POP HL          A35   E1
LD (0903),HL    A36   22 03 09
POP HL          A39   E1
POP BC          A3A   C1
POP AF          A3B   F1
RETURN          A3C   C9
```

```
LD A,R          C00   ED 5F
CALL 03B5       C02   CD B5 03
AND 03          C05   E6 03
LD E,A          C07   5F
LD A,00         C08   3E 00
CP E            C0A   BB
JR Z 0C00       C0B   28 F3
LD A,E          C0D   7B
CP C            C0E   B9
JR Z 0C00       C0F   28 EF
RETURN          C11   C9

PUSH HL         D00   E5
PUSH BC         D01   C5
LD HL,0D06      D02   21 06 0D
LD B,01         D05   06 01
LD A,R          D07   ED 5F
DJNZ, 0D07      D09   10 FC
AND 08          D0B   E6 08
PUSH HL         D0D   E5
LD LH,0D33      D0E   21 33 0D
ADD A,L         D11   85
LD L,A          D12   6F
LD E,HL         D13   5E
LD HL,0D33      D14   21 33 0D
LD B,08         D17   06 08
LD C(HL)        D19   4E
INC HL          D1A   23
LD A,(HL)       D1B   7E
DEC HL          D1C   2B
LD (HL),A       D1D   77
INC HL          D1E   23
DJNZ,0D1A       D1F   10 F9
LD (HL),C       D21   71
POP HL          D22   E1
INC (HL)        D23   34
LD A,20         D24   3E 20
CP (HL)         D26   BE
JR Z,0D2F       D27   28 06
                D29   FD 73 00
                D2C   C1
   LOOK-UP      D2D   E1
   TABLE        D2E   C9
   FOR          D2F   36 01
   RANDOM       D31   18 F6
   NUMBERS      D33   01
                D34   02
                D35   03
                D36   01
                D37   02
                D38   03
                D39   01
                D3A   02
                D3B   03
```

Finally, the listing at **0900** must be inserted. This listing can be found in issue 11 P 36, under the heading **ALIENS ATTACK RUN.** This will provide the sound for the game.

This completes the listing. Before pushing **RESET, GO**, it is a very good idea to go through the complete listing again and double-check each of the machine code values. The reason for this is to prevent the program **SELF DESTRUCTING.** This could happen if you placed the wrong value in one of the locations which caused the computer to write over some of the contents of the program.

PUSH & POP

PUSH and **POP** are very much like **PUSH** and **PULL**. They are operations which transfer the contents of a register-pair to a holding area so that the registers can be used for other operations. This holding area is called the **STACK.**

We say register PAIR because the operations **PUSH** and **POP** require that 2 registers be specified. Thus, if the accumulator (Register A) is required to be pushed onto the stack, we combine it with the FLAGS register to get the register pair: AF.

There are a few technical complications concerning the placement of bytes onto the stack but these will not concern us at this stage. It is sufficient to say that the stack is located at the top end of the RAM, (about 8 - 10 bytes from the top)and as each new set of bytes is placed on the stack, the pile grows DOWNWARDS, towards the program we are executing.

We have already seen the effect of placing (PUSHING) more and more bytes onto the stack (issue 11, P. 12) and for this reason we must use the stack very carefully. Otherwise it will increase downwards and and crash into our program!

Basically we PUSH one pair of bytes onto the stack (from say register-pair AF) then push another pair of bytes onto the stack from say register pair HL. This will leave the accumulator and HL registers free for other operations.

If we want to get the 2 bytes of AF from the stack, we must firstly POP the two bytes from HL and then we can get the AF pair. It is a simple principle of **LAST ON, FIRST OFF.**

Pushing and popping are very handy instructions. By using a PUSH instruction at the start of a routine and a POP at the end, we can place a routine such as a delay routine, which will not affect the registers at all. This routine is said to be TRANSPARENT.

PUSHING and POPPING can take place between the stack and register pairs including the index registers. This group consists of the following: AF, BC, DE, HL,IX and IY.

It is interesting to note that the bytes are pushed onto the stack HIGH BYTE first, then LOW BYTE. They come off the stack LOW BYTE then HIGH BYTE. But because the stack is increasing DOWNWARDS, each byte placed onto the stack will have a lower address!

In the programs we have presented you can see PUSH and POP in operation. The stack is a temporary holding area and only the top pair can be accessed.

MORE PROGRAMS FOR THE 8x8 DISPLAY:

The 8x8 matrix was a very popular 'add-on', with nearly every TEC owner building up a display.

Here are some more programs for the matrix, commencing with a simple routine similar to the **FAN OUT** on P.34 of issue 11.

FAN OUT MK II

```
LD A,01         800    3E 01
OUT (3),A       802    D3 03
OUT (4),A       804    D3 04
RLA             806    07
PUSH AF         807    F5
CALL DELAY      808    CD 00 09
POP AF          80B    F1
INC A           80C    3C
JP NZ 802       80D    C2 02 08
LD A,FE         810    3E FE
OUT (3),A       812    D3 03
OUT (4),A       814    D3 04
RLA             816    07
PUSH AF         817    F5
CALL DELAY      818    CD 00 09
POP AF          81B    F1
DEC A           81C    3D
JP NZ 812       81D    C2 12 08
JP 802          820    C3 02 08
```

Delay at 0900:

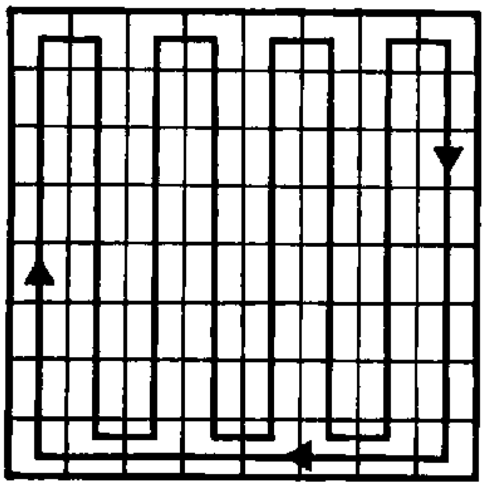
```
11 FF 06
1B 7B B2
C2 03 09
C9
```

BOUNCING BALL

by G L Dunt, 3219.

Bouncing Ball is an extension of **'AROUND THE DISPLAY'** (issue 11, P.29).

The diagram below shows the effect produced by this program and by varying the delay, it will appear as if two or more LEDs are circulating the display.



```
800    3E 01
802    D3 03
804    0E 08
806    3E 01
808    D3 04
80A    47
80B    CD 00 0C
80E    78
80F    CB 07
811    0D
812    C2 08 08
```

DELAY AT 0C00:

```
11 FF 06
1B
7B
B2
C2 03 0C
C9
```

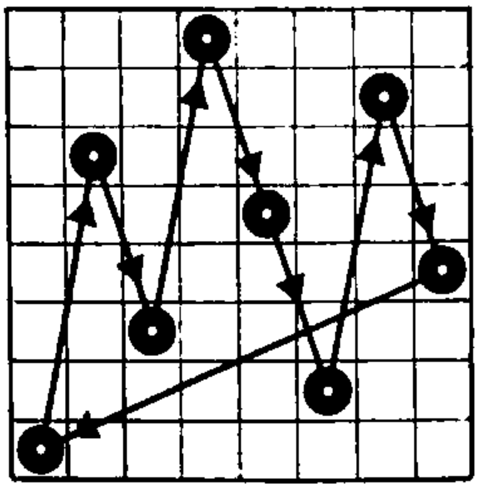
Type the first section into the TEC and **RUN**. This will check the code-values and prevent a major mistake. Type the second stage and **RUN**. Continue this way until the whola program has been inserted.

```
815    3E 02
817    D3 03
819    0E 08
81B    3E 80
81D    D3 04
81F    47
820    CD 00 0C
823    78
824    CB 0F
826    0D
827    C2 1D 08

82A    3E 04
82C    D3 03
82E    0E 08
830    3E 01
832    D3 04
834    47
835    CD 00 0C
838    78
839    CB 07
83B    0D
83C    C2 32 08

83F    3E 08
841    D3 03
843    0E 08
845    3E 80
847    D3 04
849    47
84A    CD 00 0C
84D    78
84E    CB 0F
850    0D
851    C2 47 08

854    3E 10
856    D3 03
858    0E 08
85A    3E 01
85C    D3 04
85E    47
95F    CD 00 0C
862    78
863    CB 07
865    0D
866    C2 5C 08

869    3E 20
86B    D3 03
86D    0E 08
86F    3E 80
871    D3 04
873    47
874    CD 00 0C
877    78
878    CB 0F
87A    0D
87B    C2 71 08

87E    3E 40
880    D3 03
882    0E 08
884    3E 01
886    D3 04
888    47
889    CD 00 0C
88C    78
88D    CB 07
88F    0D
890    C2 86 08

893    3E 80
895    D3 03
897    0E 08
899    3E 80
89B    D3 04
89D    47
89E    CD 00 0C
8A1    78
8A2    CB 0F
8A4    0D
8A5    C2 9B 08

8A8    3E 01
8AA    D3 04
8AC    0E 06
8AE    3E 40
8B0    D3 03
8B2    47
8B3    CD 00 0C
8B6    78
8B7    CB 0F
8B9    0D
8BA    C2 B0 08
8BD    C3 00 08
```

JUMPING LEDs.

· by G L Dunt, 3219.

This program demonstrates multiplexing in an easily understood manner.



By adjusting the SPEED CONTROL, the flickering effect of each LED will be speeded-up to give a steady pattern.

```
LD A,01         800   3E 01
OUT (3),A       802   D3 03
OUT (4),A       804   D3 04
CALL DELAY      806   CD 00 0C
LD A,02         809   3E 02
OUT (3),A       80B   D3 03
LD A,20         80D   3E 20
OUT (4),A       80F   D3 04
CALL DELAY      811   CD 00 0C
LD A,04         814   3E 04
OUT (3),A       816   D3 03
OUT (4),A       818   D3 04
CALL DELAY      81A   CD 00 0C
LD A,08         81D   3E 08
OUT (3),A       81F   D3 03
LD A,80         821   3E 80
OUT (4),A       823   D3 04
CALL DELAY      825   CD 00 0C
LD A,10         828   3E 10
OUT (3),A       82A   D3 03
OUT (4),A       82C   D3 04
CALL DELAY      82E   CD 00 0C
LD A,20         831   3E 20
OUT (3),A       833   D3 03
LD A,02         835   3E 02
OUT (4),A       837   D3 04
CALL DELAY      839   CD 00 0C
LD A,40         83C   3E 40
OUT (3),A       83E   D3 03
OUT (4),A       840   D3 04
CALL DELAY      842   CD 00 0C
LD A,80         845   3E 80
OUT (3),A       847   D3 03
LD A,08         849   3E 08
OUT (4),A       84B   D3 04
CALL DELAY      84D   CD 00 0C
JP 0800         850   C3 00 08
```

DELAY at 0C00:

```
11 0F 0F
1B
7B
B2
C2 03 0C
C9
```

Change delay to these values to create the full multiplexing effect.

at 0C00:

```
11 0D 01
11 6F 00
```

PRODUCING A LETTER

This extension to **JUMPING LEDs** program produces a letter of the alphabet. It will show the flexibility of multiplexing. Any figure or shape can be created on the screen.

The letter we will produce is the letter 'A'. This will be somewhat dimmer than when displaying one or two LEDs due to the current limitation of the latch at port 3. It cannot supply sufficient current to turn on 8 LEDs at the same time. A set of emitter-follower transistors would cure the problem.

```
LD B,08         800   06 08
LD A,01         802   3E 01
LD HL,0B00      804   21 00 0B
OUT (3),A       807   D3 03
PUSH AF         809   F5
LD A,(HL)       90A   7E
OUT(4),A        80B   D3 04
CALL DELAY      80D   CD 00 0A
INC HL          810   23
POP AF          811   F1
RLC A           812   CB 07
DEC B           814   05
JP NZ 807       815   C2 07 08
JP 800          818   C3 00 08
```

Delay at 0A00:

```
11 0F 01
1B
7B
B2
C2 03 0A
C9
```

at 0B00:

BYTE TABLE for letter A:

```
00
1F
3F
64
64
3F
1F
00
```

PODUCING A SHORT DELAY

When running the letter program above, you will find a disturbing flickering produced by the scan routine. This is basically due to the number of operations which must be carried out by the Z80 for each complete cycle of the program.

This takes a lot of clock cycles and the scan speed cannot be increased without increasing the clock frequency.

The solution is to provide a delay routine which requires less clock cycles for each loop.

This can be done by using the B register and an auto decrement function **DJNZ.** This will automatically decrement register B until it becomes zero.

At **0A00** the following delay routine is inserted:

```
PUSH BC         C5
LD B,FF         06 FF
DJNZ            10 FE
POP BC          C1
Return          C9
```

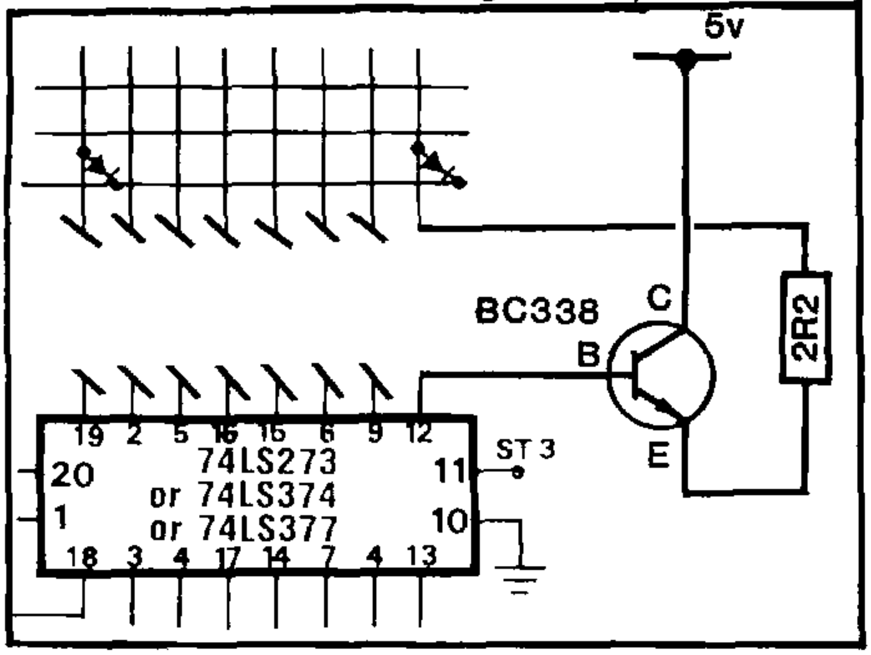
Note: The B register must be pushed onto the stack before it can be used as a decrementing register as it is alreay used in the main program to count the number of DATA BYTES.

To reduce the flicker even more, change the value of B for **FF** to **50**(or similar value). If the display is too dim, try our next modificetion:

INCREASING THE BRIGHTNESS OF THE 8x8

The brightness of the 8x8 can be dramatically improved by sourcing the display with a set of transistors.

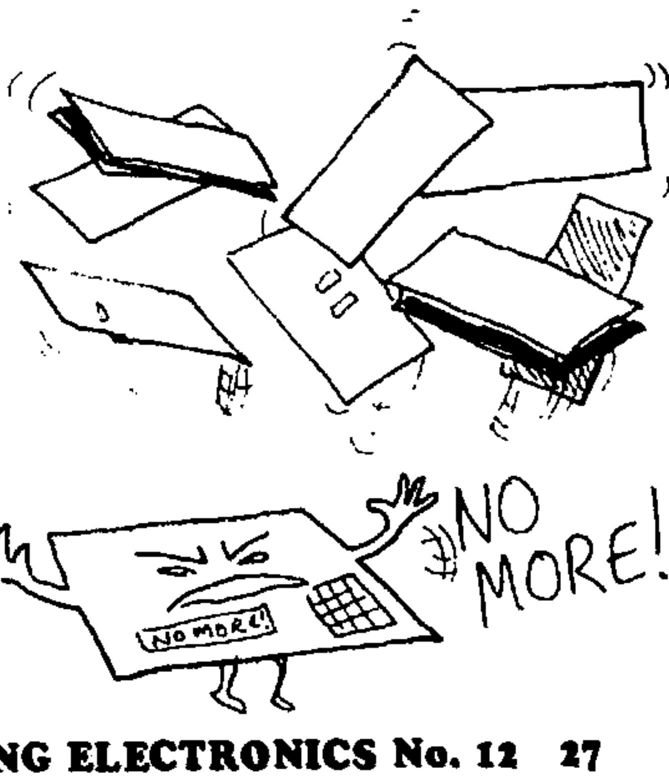
These are soldered under the PC in a row similar to the 8 sinking transistors. Don't forget to cut the PC tracks to each of the columns of LEDs before starting assembly.



This will enable you to start experimenting with different letters and shapes on the display and allow you to see them in a brightly lit room.

We will continue next issue with running these letters across the display in a similar manner to the running signs in shop windows etc.

AND NEXT MONTH.....



PRESENTING:

$9.70 per 2k
No PC board required.

OUR RAM STACK

ADD 12K TO THE TEC. OR ANY NUMBER OF 2K BLOCKS FOR RUNNING LARGER PROGRAMS.

Some of the best ideas are discovered by accident while others are over-looked for years because of their sheer simplicity.

Such is the case with our RAM STACK.

We have been thinking of a RAM PACK for a long time but never came up with an idea we really liked. Most ideas revolved around a PC board and trying to simplify the accompanying complexity of track-work.

Due to the parallel wiring requirement of memory chips, it is necessary to have PC tracks running between each of the pins.

This produces a very FINE set of tracks and consequently a number of problems arise. The most troublesome of these is the chance of a land being cut-in-two when the holes are being drilled. This causes a fine break in the track-work which must be repaired with solder when the components are being mounted on the board. Failure to see any of these breaks will render some of the chips inoperative.

In addition, the closeness of the tracks highlights the need for a solder mask, all contributing to increasing the cost of the project.

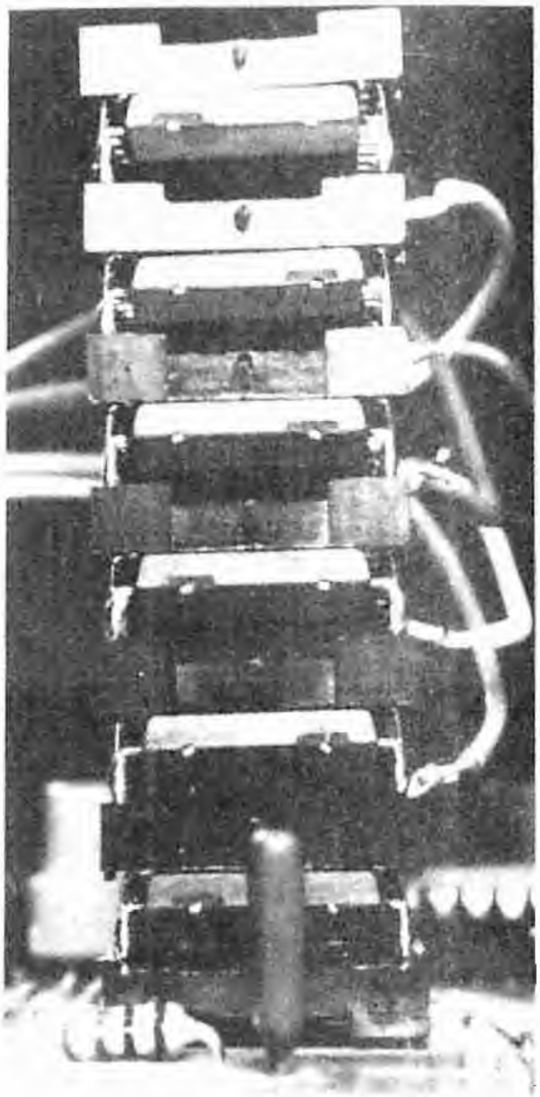
But a PC board can actually be eliminated.

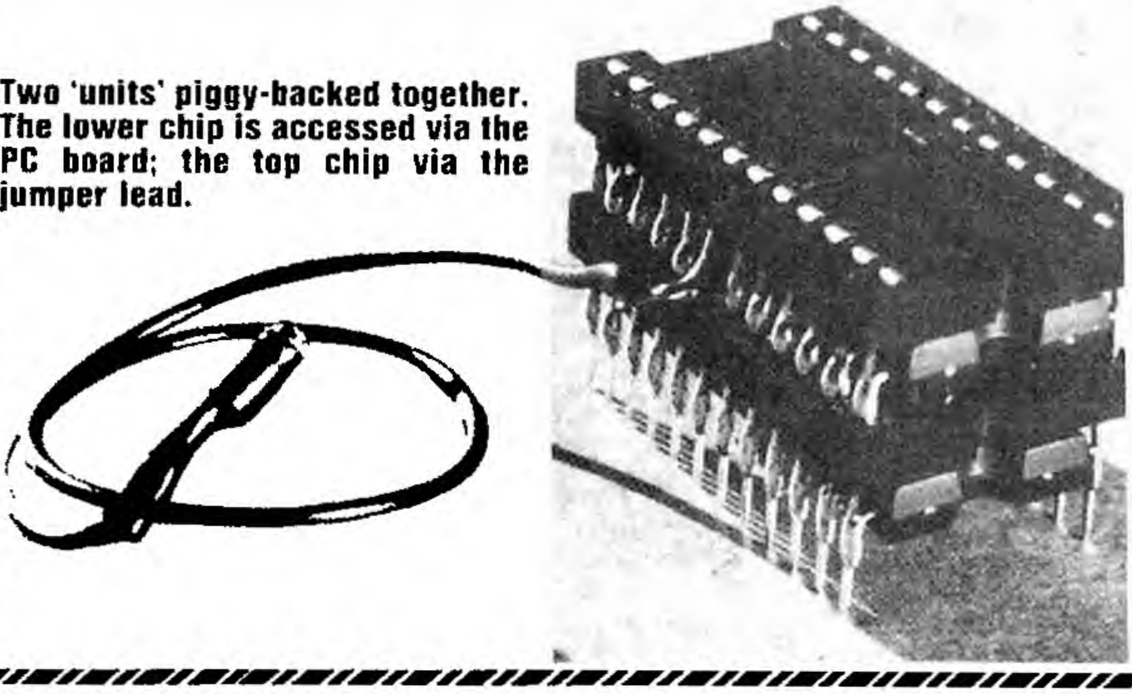
The simplest and best design for a RAM pack requires no more than a set of IC's and sockets.

And that's when we struck upon our brilliant idea - a RAM STACK.

Not only is this design the cheapest arrangement possible, but it also incorporates a number of advantages. The best of these is memory can be increased or decreased in blocks of 2k for little more than the cost of an IC and socket. This will enable 2k or 4k to be an economical addition.

With our design, if a fault develops, each chip can be tested individually and removed if found to be defective.





Two 'units' piggy-backed together. The lower chip is accessed via the PC board; the top chip via the jumper lead.

PARTS LIST:

for each 2k:

1	-	6116 or equiv.
1	-	24 pin IC socket
1	-	length of hook-up flex
1	-	matrix pin & connector.

Putting all these advantages together you can see why we are pleased with this design. The accompanying photos shows how it goes together.

There isn't much to explain about construction. It's just a matter of placing an IC socket over a RAM chip and soldering each of the chip-pins to the socker pins.

Make sure the solder does not flow down any of the IC's pins otherwise you will not be able to insert the chip into a socket when putting the whole thing together.

The only other connection to each of the RAM chips is at pin 18. This is the **CHIP ENABLE** pin and an individual line is taken from one of the outputs of the 74LS 138 (near the oscillator chip), to this pin.

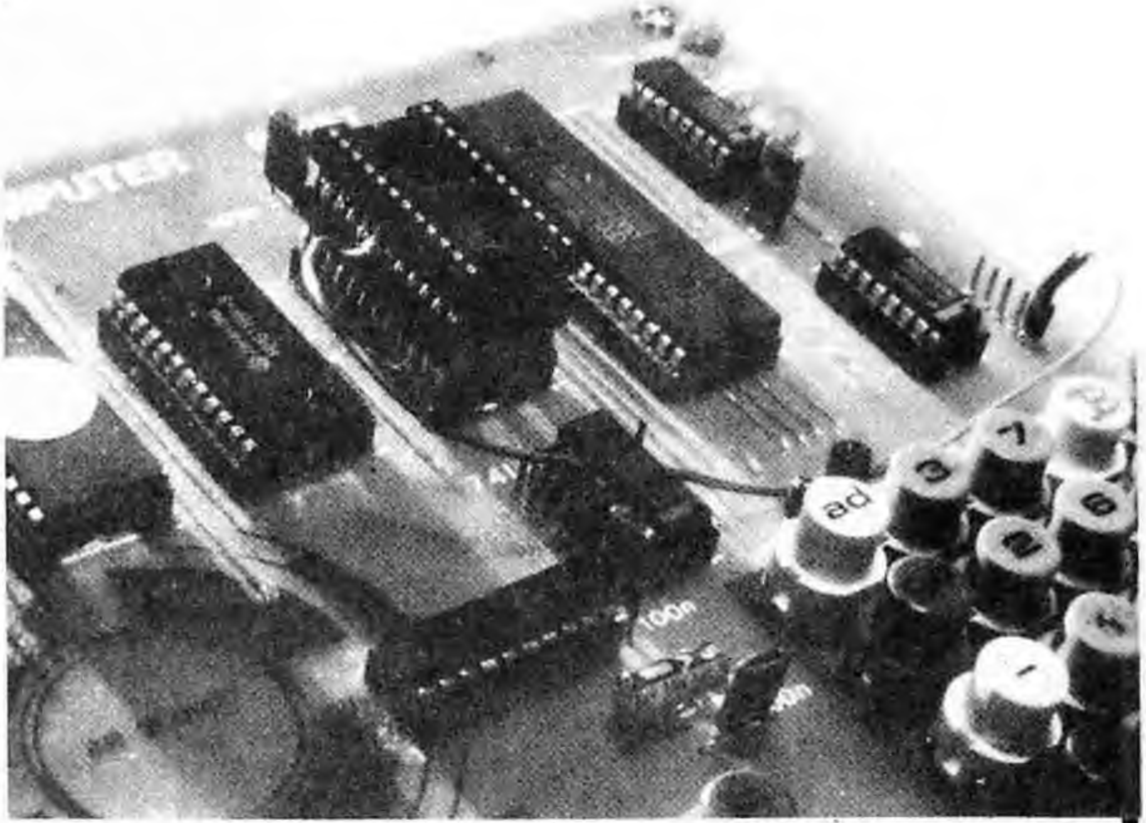
Each pin 18 of a memory chip must be kept separated from the others so that any chip can be individually selected.

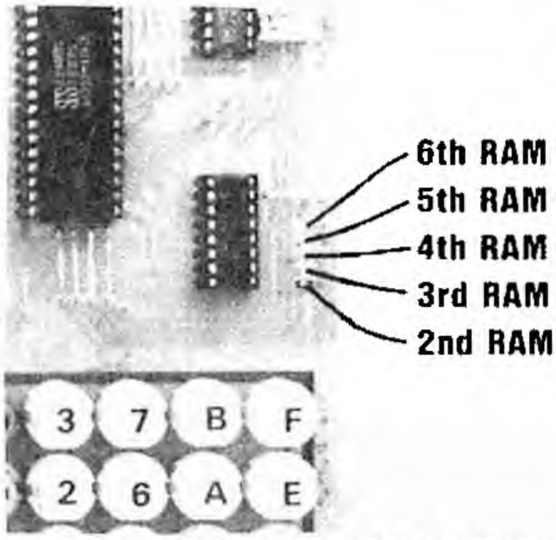
The close-up photos show how this pin is bent away from the rest so that it does not make contact with the lower IC socket.

Only the lowest RAM chip in the stack is selected by the track under the PC board. All others are connected via jumper leads, directly to the relevant output of the 74LS138 mentioned before.

Without any additional decoding, we can add a stack of 6 chips to the EXPANSION PORT SOCKET making a total of 14k for the TEC.

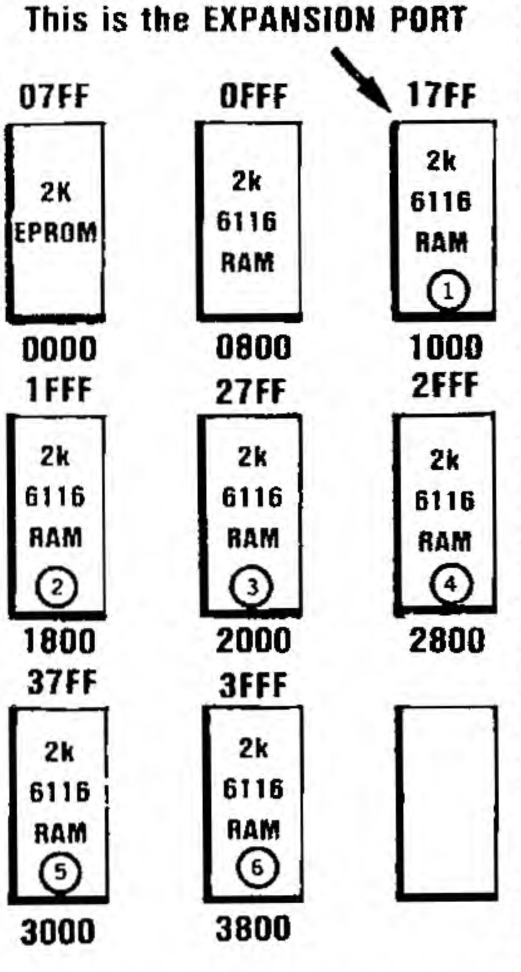
The lowest chip will have address values starting at 1000H to 17FFH. The others will have values as shown in the diagram below.



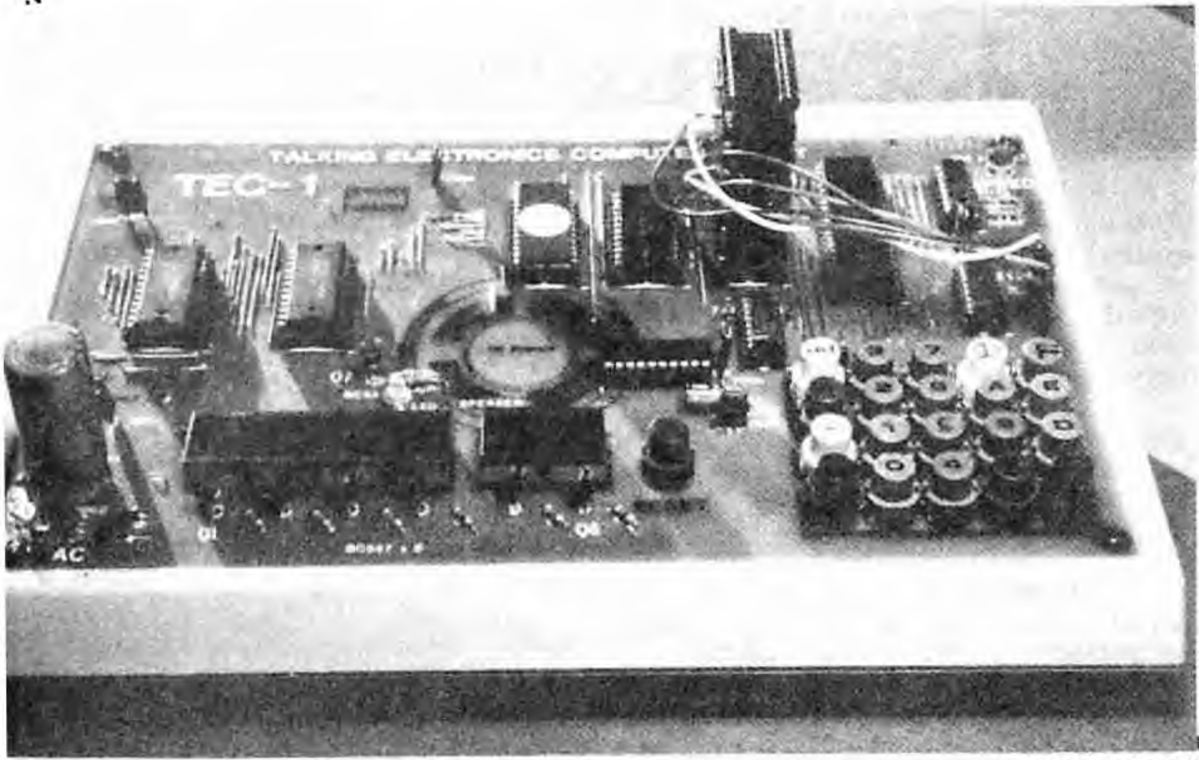


The first chip in the stack is enabled via the Expansion Port socket wiring. The other chips are enabled by connecting a jumper lead from pin 18 to one of the pins as shown above.

Of course you can **ENABLE** the chips 'out-of-order', by mixing up the jumper leads. This may fill the bottom chip, then the top chip, then number 3, then the fifth etc. No damage will result, it's just not a systematic way to do it.



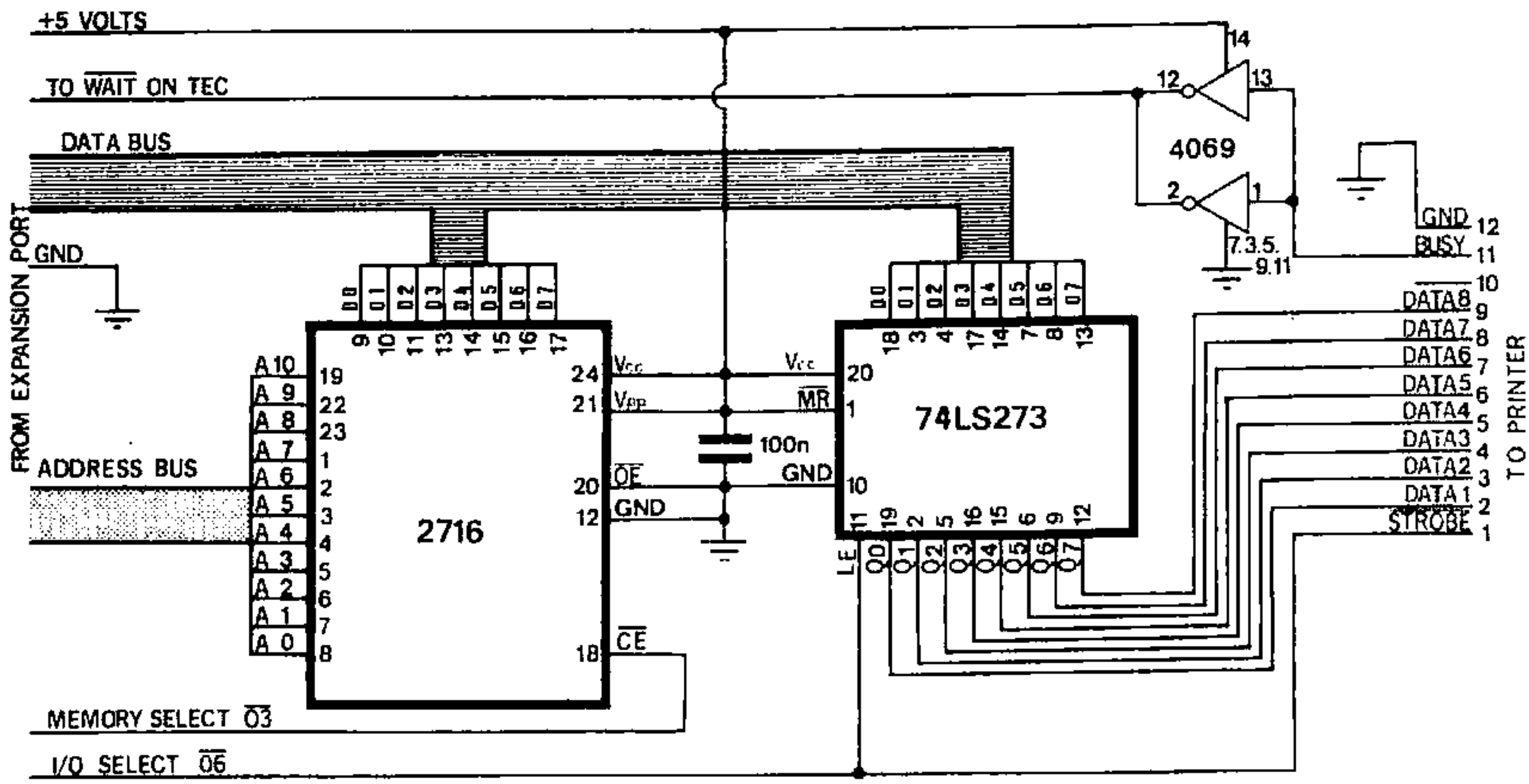
The start and finish address for the first 6 RAM chips.



Connecting the TEC to a:

Parts: $24.80 PC board: $3.80

PRINTER/PLOTTER



Printer/Plotter circuit

Buy a Dick Smith VZ 200 Printer-Plotter.

This project explains how to directly access (talk to) a PRINTER/PLOTTER. We have used the most readily available printer/plotter as it is not only the cheapest, but can be obtained from a number of suppliers.

Talking to one of these clever little performers is not very involved when you know how. But without the correct information it will remain completely DEAD. When you know how to supply it with the right stuff, it will do practically anything bar talk to you.

Actually you only have to sand it the necessary codes to produce the character, all the creation of the shape of the symbol is done by the chip within the printer.

Not only do these printer/plotters accept instructions to produce numbers letters and' symbols, but they can also be told to rotate, plot, vary the size of the characters and move in almost any diraction.

There is a two-way interaction between printer and computer. Data is sent to the printer faster than it can be executed and to save holding up the computer, it is deposited in a FIFO register in blocks of about 4 bytes (in our case). Larger computers can be instructed to go away and execute other work while the FIFO register empties.

Bursts of date are transmitted like this until the whole program is executed.

As we have used a standard printer, it is obvious that it has been designed to connect to any computer which has a normal, full-size, key-board so that each key will produce the corresponding letter on the paper.

But this luxury is not absolutely necessary as the computer merely produces a code number which is sent to the printer.

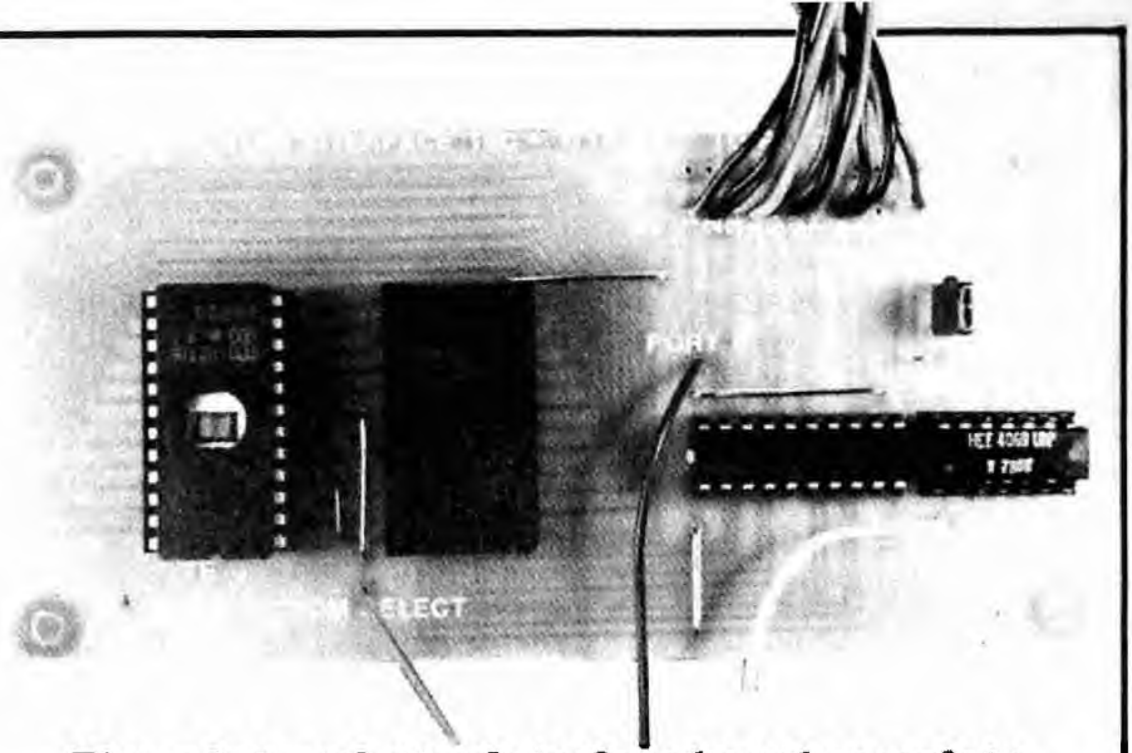
The code number (or value) is called an ASCII number or ASCII CODE and fortunately is identical for all types and models of personal computers.

The secret to getting the printer to work on the TEC is the latch chip. It holds the data long enough for the printer to read it.

PARTS LIST

1 ·	100n mono block
1 -	4049
1 ·	74LS273
1 ·	2716 (programmed)
1 -	14 pin IC socket
1 -	20 pin IC socket
1 -	24 pin IC socket
1 -	24 pin wire-wrap socket
1 -	24 pin DIP HEADER
1 ·	36 pin Centronics type plug.
	tinned copper wire
	hook-up flex
3 ·	'quick connect' pins and sockets

PRINTER INTERFACE PC BOARD



The printer/plotter interface board complete and ready for plugging into the TEC & printer.

This means all we have to do is produce the same set of ASCII numbers (or codes) and the printer will produce the correct set of shapes on the paper.

Thus we don't need a full-size computer at all.

It may be a bit slow pressing the keys on the TEC, but all the printing capabilities will be possible, and that's all we want.

In this series of articles, we will explore the functions of the printer/plotter and create some amazing effects.

The most important aspect of this is realizing you can create a CONTROL PROGRAM with machine code listings and thus fill the minimum amount of memory for any given effect.

In this way you can produce you own system and expand it as much as you like without having to resort to buying a ready-made console. This will produce a cheaper and more compact system and will gain you much more respect from your boss or customer.

The first part of this project requires assembly of the printer interface board. This board contains a latch and EPROM (filled with a number of handy programs). This will give you a run-up program to test the interface board and provide instant transfer of data from computer to paper to reduce the amount of button-pushing.

The other chip on the board provides an inverted WAIT signal to halt the Z80. This basically keeps the two units in synchronisation.

Set out all the parts on your bench and check everything. Solder the sockets, cap and 6 jumper links to the board. Mount the wire-wrap socket through the board so that the long pins act as 'stand-offs' for the component header plug. See the RELAY DRIVER BOARD article and photos for details of how this is done.

The final task involves connecting the board to the 36 pin Centronics plug.

WIRING THE PLUG

Wiring the Centronics-types plug to the printer interface is very easy. On the printer interface PC board there are 24 holes. Twelve of these are numbered. These numbers correspond to the numbers on the Centronics plug. Solder a length of hook-up wire between each hole and a corresponding hole on the connector plug. Pin 10 is not used, so no lead is needed. It is not necessary to use special connecting flex such as twisted pairs or screened lead. Our prototype worked perfectly with ordinary hook-up flex. It's best to use different coloured flex for each line to make tracing easier. These leads can be about 50 cm long and kept together with ties or tape at regular intervals.

The pens use water-based ink and tend to dry-out fairly easily. If they fail to start: open them up, add a drop of water, heat them up and fit them back into the printer.

We used a VZ 200 printer/plotter but there are other units with the same internal workings on the market. But they may not have the same input instruction set.

A close-up of the 4-pen print head.

The only remaining wires left are the 3 control lines. These are:

Memory select 03,
I/O select 06, and
WAIT.

These are fitted with 'quick connect' terminals which push onto matrix pins on the main PC board. Heat-shrink tubing can be placed over the terminals to strengthen the solder joint and make them easier to handle.

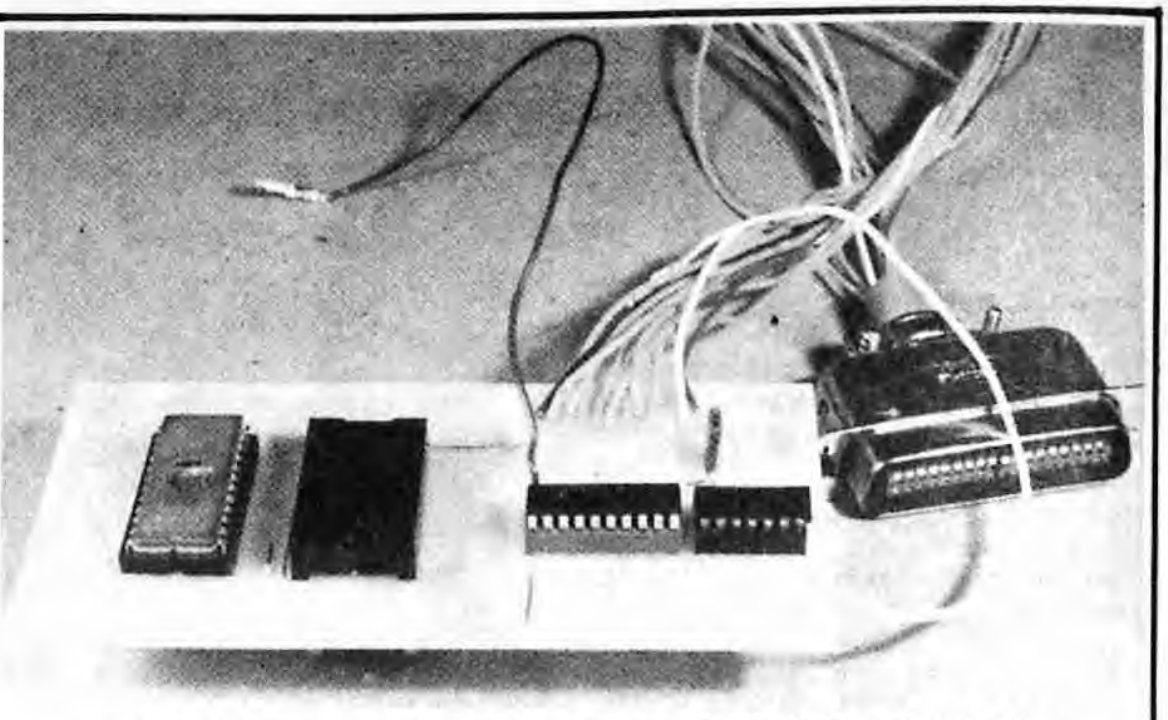
When the printer is first turned on it runs through an initial program (from its internal memory) which feeds the paper, sets the pen colours and starts the ink flowing by producing a box with each pen.

After this, there is very little else you can do via the buttons on the unit, except forward feed, change the colour of the pens and/or remove them.

All the rest of the action must come in the form of data from an outside source.

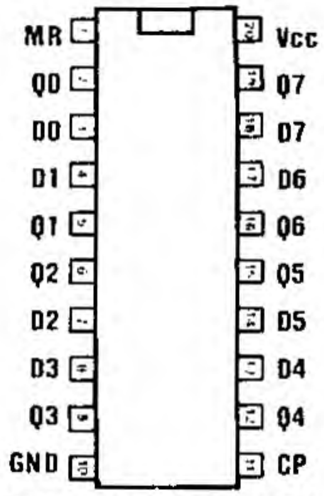
This is why we need the TEC. It supplies data at high speed to get the print-head moving.

Connect the centronics plug into the rear of the printer and fit the PRINTER INTERFACE PC BOARD to the computer. Connect the 3 flying leads as shown in the diagram:



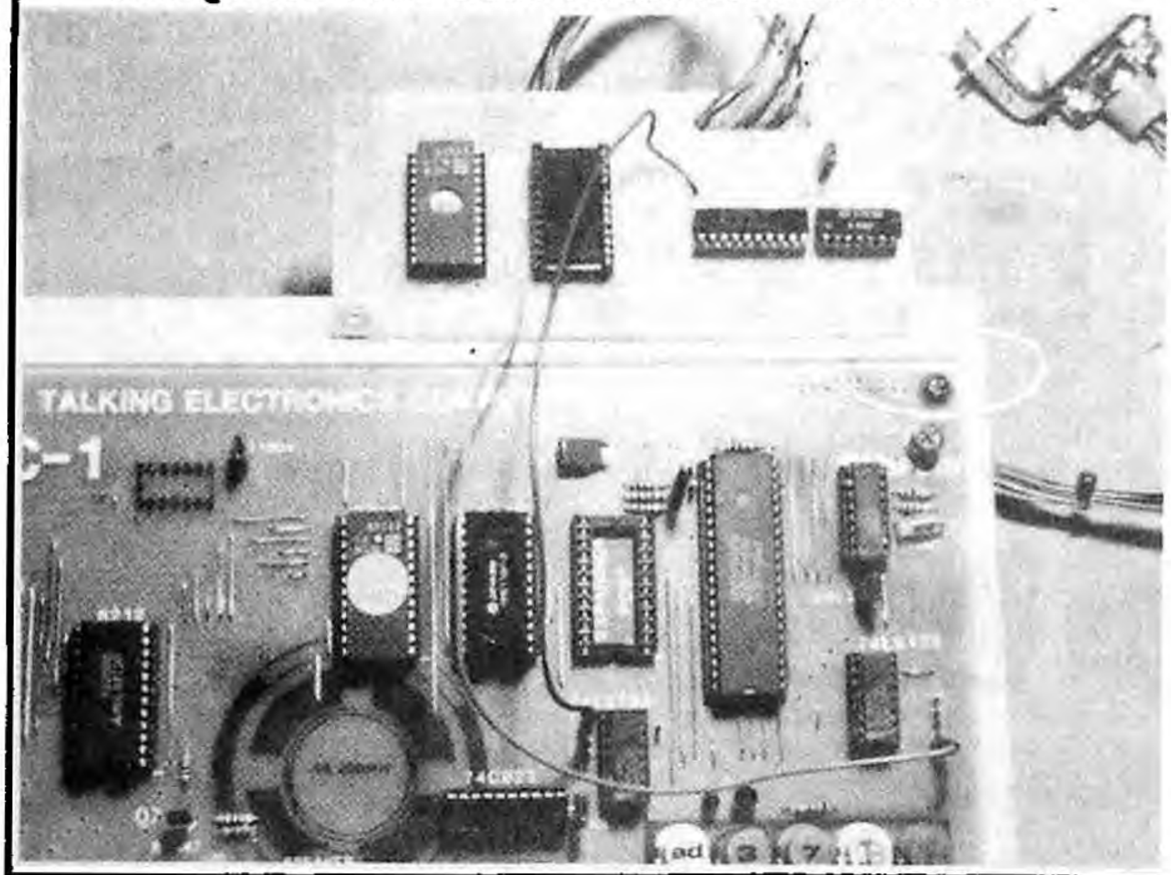
All the parts shown are included in the kit.

74LS 273



Pin-out for 74LS273

This photo shows the connections to the TEC.



To get something interesting out of the printer you will need to send it a program. The first of these is:

KEN's START-UP PROGRAM:

Make sure the print-head is to the left of the printer as when the printer has been switched on.

Push **ADdress 18A0 GO GO.**

Watch the result.

This type of program is beyond us at the moment but you will be capable of similar effects after reading this article.

For now, the next step is to be able to get letters and characters onto the paper.

PRODUCING LETTERS etc. . .

All information is fed to the printer in ASCII code. If you want a particular character, the correct code must be sent to the printer. Even if you want to send a number to the printer, such as 150, you must send it in the form of ASCII. This means 150 translates to 31 35 30, as you will see later from the table.

A small program is required to interpret your button pushing and send it to an output port. This is similar to making a segment on the display illuminate and the program for this is contained in the PRINTER/PLOTTER EPROM at 1980.

To use this program:

Press **ADdress 1980 GO GO**

The display will go blank and the TEC will be ready for conveying your keyboard instructions directly to the printer.

Each of the letters, numbers and symbols is shown in the table below and the corresponding hex value must be used for the symbol to appear on the paper.

Try obtaining all the letters, numbers and characters by following through the table.

PRINTER/PLOTTER ASCII VALUES:

	SPACE 20	0 30	@ 40	P 50	` 60	p 70
BACK SPACE 08	! 21	1 31	A 41	Q 51	a 61	q 71
LINE FEED 0A	" 22	2 32	B 42	R 52	b 62	r 72
CR 0D	# 23	3 33	C 43	S 53	c 63	s 73
DC1 11	$ 24	4 34	D 44	T 54	d 64	t 74
DC2 12	% 25	5 35	E 45	U 55	e 65	u 75
NEW COLOUR 1D	& 26	6 36	F 46	V 56	f 66	v 76
	' 27	7 37	G 47	W 57	g 67	w 77
	(28	8 38	H 48	X 58	h 68	x 78
	) 29	9 39	I 49	Y 59	i 69	y 79
	* 2A	: 3A	J 4A	Z 5A	j 6A	z 7A
LINE DEFORE 0B	+ 2B	; 3B	K 4B	[5B	k 6B	{ 7B
	, 2C	3C	L 4C	\ 5C	l 6C	¦ 7C
	- 2D	= 30	M 4D	] 5D	m 6D	} 7D
	. 2E	> 3E	N 4E	^ 5E	n 6E	~ 7E
	/ 2F	? 3F	O 4F	_ 5F	o 6F	⊠ 7F

Try the following sequence and you will see a word appear:
49 4E 43 52 45 44 49 42 4C 45

For the hex value **49**, the letter I will be printed. Press each number only ONCE. The first press will appear to have no effect, but as soon as the second button is pressed, the letter I will be printed.

Be very careful not to press button-sequence **11** or **12** as this will cause the mode to change and everything will appear to 'lock-out'.

Try writing a sentence using the hex key pad. It's slow but eventually gets you there. A space between words is created by typing **20.**

Any sentence you send to the printer via the keyboard can be re-presented again and again if placed into memory before-hand. It can also be corrected and adjusted (within limits). To do this, place the data at **0800** and call a program at **1880.**

Insert the following at **0800:**

49 4E 43 52 45 44 49 42 4C 45 20 20 48 55 4C 4B 0D 0A 1D FF.

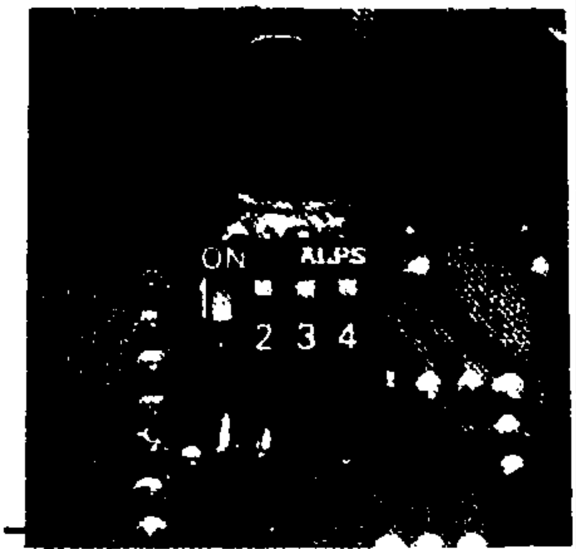
Push **ADdress 1880 GO GO.**

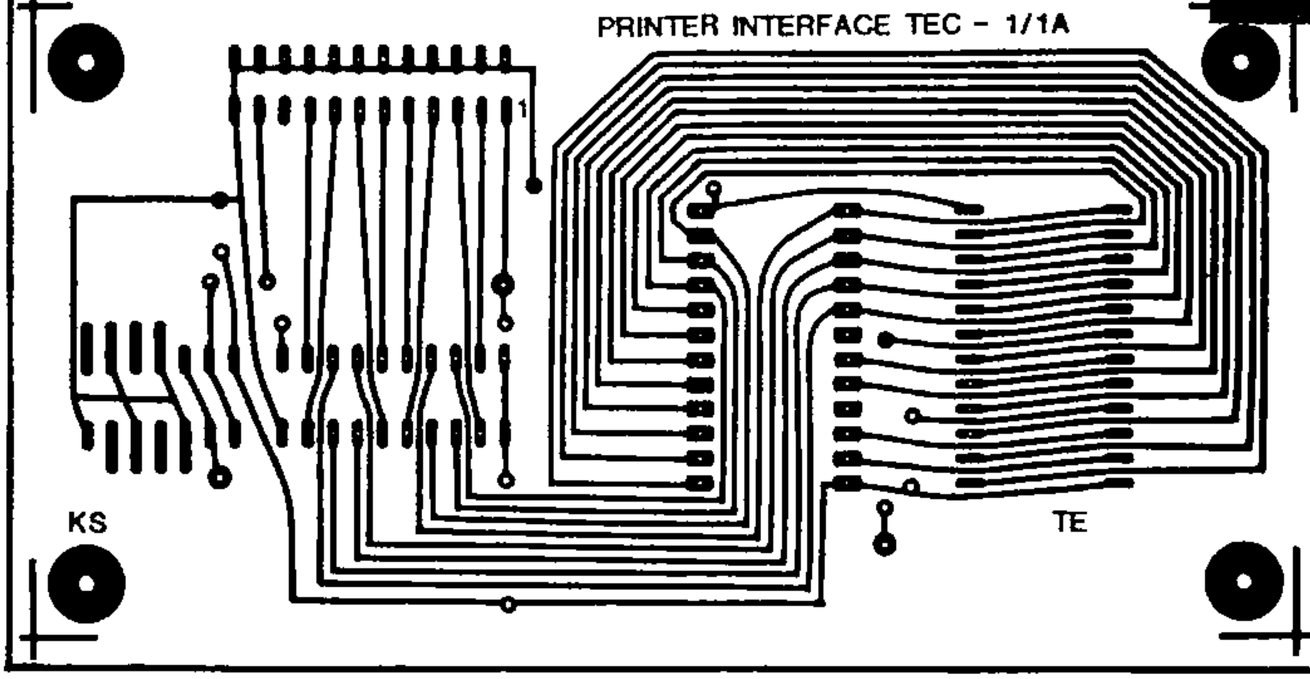
Recall it again by pressing:
ADdress 1880 GO GO.

THE LIST PROGRAM

This program lists any part of the EPROM, RAM or any additional memory you add to your TEC. In fact the first thing you can do is get a print-out of your MONitor ROM. Many readers have written requesting a listing of the MONitor and now they can produce it themselves.

But before you can get a listing, you must make a modification to the operation to the printer. This involves setting the two switches under the printer:





The PC layout for the Printer/Plotter. The overlay and parts positioning can be gained from the photo on P 31.

This is how to do it.

On the bottom of the printer is a small plete. Undo the screws and remove the plate. Inside you will find a bank of 4 switches. Switch 1 should be in the OFF position and switch 2 in the ON position. Don't worry about switch 3 and 4.

Whan the switches are set like this, CR (carriage Return) will set the print-head to tha left of the paper without feeding the paper forward. The paper can then be fad forward by using LF (Line Feed). The switches should be set like this because the program in ROM automatically line feeds after each carriage return. If the switches are not set like this, the typing will be double line spaced.

Enter the following into the TEC:
ADdress 1800 GO GO:

The display will go blank and the printer will CR and LF. Now enter **0000** and the printer will start printing out characters in pairs. This is a listing of the contents of your monitor ROM.

If you want a listing of any of the programs you have typed into memory, start at **0800** or where your program starts, and enter a 4-digit number into the keyboard. It must be 4 digits, so don't forget the leading 0.

The text mode is not very interesting. After all, we have seen electric/electronic typewriters for years, But for a print-head to produce GRAPHICS! That's different!

GRAPHICS MODE

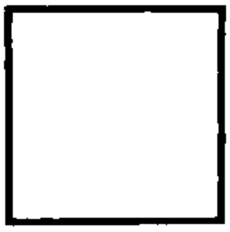
The program at **1880** can also be used to generate graphics on the printer.

Remember, all information must ba programmed into the printer in ASCII.

Type the program below into the TEC's memory at **0800.** An **FF** is placed after the last piece of data to signify the end of a program. Now run the program at **1880** by pressing **ADdress 1880 GO GO.**

at 0800: 0A 0D 12 49 2C 44 38 30 2C 30 2C 38 30 2C 2D 38 30 2C 30 2C 2D 38 30 2C 30 2C 30 0D FF.

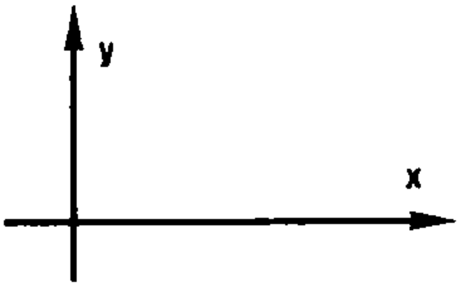
The printer will draw a square.



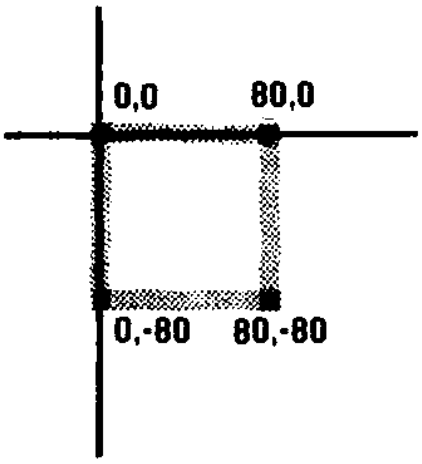
Look at the listing. It may look complex but can be easily decoded using the table. It will decode to this:

OA = LF = Line Feed.
OD = CR = Carriage Return
12 = DC2 = Graphic Mode
49 = I = sets the pen's location as co-ordinates 0,0.
2C = , =Separates I from D
44 = D = draw from present location to the co-ordinate given by the next byte(s) of data.
38 = 8
30 = 0
2C = ,
30 = 0
2C = ,
38 = 8
30 = 0
2C = ,
2D = -
38 = 8
30 = 0
2C = ,
30 = 0
2C = ,
2D = -
38 = 8
30 = 0
2C = ,
30 = 0
2C = ,
30 = 0
OD = CR = carriage return
FF = signifies end of program.

The printer uses a co-ordinate system exactly like the x,y axis used to draw graphs. The origin is 0,0 (or 00,00) and the positive direction of x and y is shown on the diagram.



The co-ordinates of the corners of the box are shown in this diagram. This clearly shows how the values are obtained.



The program can be separated into 4 sections, each drawing one side of the box.This will show how the program goes together.

The following program produces the top of the square:

at 0800 type: **0A 0D 12 49 2C 44 38 30 0D FF.**

ADdress 1880 GO GO.

The result will be:

Let us produce a line the full width of the paper. For this you will need a 3-digit value. The printer is capable of accepting a value as high as 999 (also -999) but this will be too high for our width of paper. Try 300.

The ASCII value is 33 30 30.

at 0800:
0A 0D 12 49 2C 44 33 30 30 2C 30 0D FF.
Press **ADdress 1880 GO GO.**

The final **0D** is important to get the printer to execute the graphics command.

The value **300** will not quite reach the far side of the paper. Try **450.** This will be about the longest line possible and don't forget to use the ASCII values in the program.

Shorten the side of the box to 80 and continue with the experiment.

The second side of the box will be produced at an angle other than 90° by inserting the following co-ordinates: 50, -80

at 0800:
0A 0D 12 49 2C 44 38 30 2C 30 2C 35 30 2C 2D 38 30 0D FF.

Run the program. Does it produce two sides of an irregular figure?

The next side will be produced as follows:

0A 0D 12 49 2C 44 38 30 2C 30 2C 35 30 2C 2D 38 30 2C 31 35 30 2C 2D 38 30 0D FF.

Run the program and see the result.

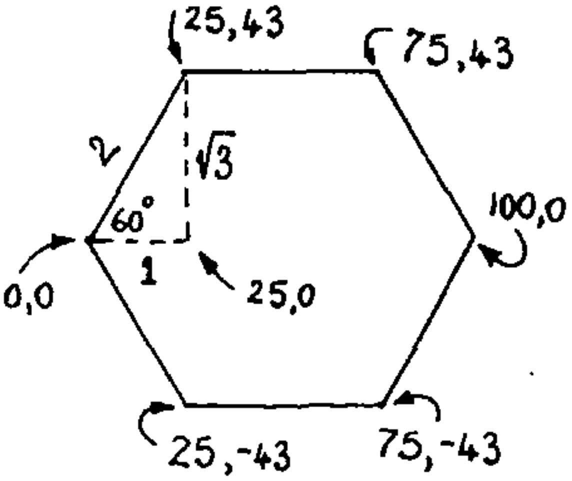
Finally:
0A 0D 12 49 2C 44 38 30 2C 30 2C 35 30 2C 2D 38 30 2C 31 35 30 2C 2D 38 30 2C 0A 0D 11 1D 0D FF.

Produce other shapes and you will understand how to plot co-ordinates.

HEX

The second shape we will investigate is a HEXAGON.

To produce this shape you need to know the value of the internal angle and produce a 30° 60° 90° triangle as shown. This will give you the length of the sides of the triangle and from this the first set of co-ordinates can be obtained (25,43) These values are 1/4 of 100, 173, which are the lengths of the sides of the triangle.

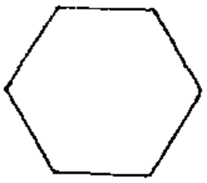


The second co-ordinate, 75,43 is found by adding 50 to the value 25. Continue around the hex shape until the figure is closed.

This is the listing for the printer:

at 0800:

```
12 49 0D 44 32 35 2C 34 33 2C 37
35 2C 34 33 2C 31 30 30 2C 30 2C
37 35 2C 2D 34 33 2C 32 35 2C 2D
34 33 2C 30 2C 30 0D FF.
```



0's and X's

The new instruction with this shape is the MOVE command.—4D

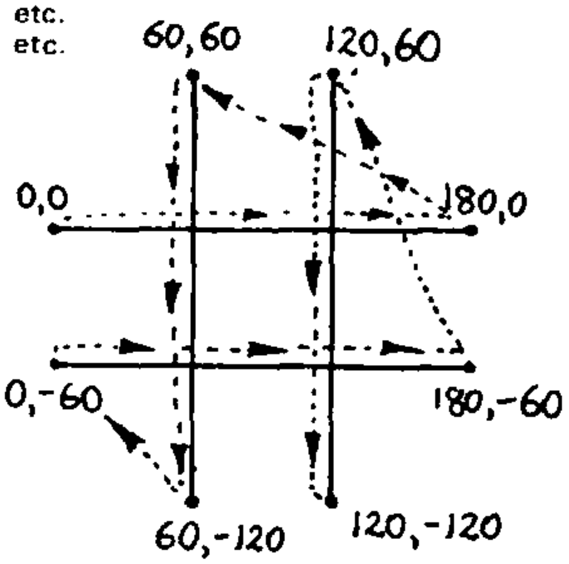
This instructs the pen to lift from the page and move to a specified location without drawing on the paper.

Here is the listing and the shape which will be drawn:

```
0800  0A  0D  12  49  0D  44  31  38
0808  30  2C  30  0D  4D  36  30  2C
0810  36  30  0D  44  36  30  2C  2D
0818  31  32  30  0D  4D  30  2C  2D
0820  36  30  0D  44  31  38  30  2C
0828  2D  36  30  0D  4D  31  32  30
0830  2C  36  30  0D  44  31  32  30
0838  2C  2D  31  32  30  0D  FF
```

This is a decoding of the first part of the listing. This will be sufficient to understand how the program is written.

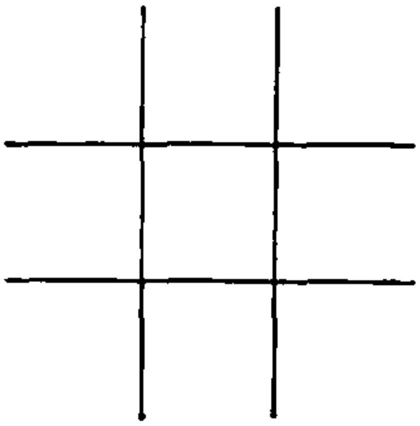
0A = LF =Line Feed
0D = CR =Carriage Return
12 = DC2 = Graphics Mode
49 = I = initialize the co-ords 0,0
0D = CR = signifies the end of the previous command. It does not cause the carriage to return but enables the previous command to be carried out.
44 = D = Draw
31 = 1
38 = 8
30 = 0
2C = ,
30 = 0
0D = CR = end of draw statement.
4D = M = Move. The pen is instructed to move without drawing.
36 = 6
30 = 0
2C = ,
36 = 6
30 = 0
0D = CR End of Move statement.
44 = D = Draw
36 = 6
30 = 0
etc.
etc.



This diagram shows the value of the co-ordinates required to draw the shape.

Copy out the complete listing and decode it to prove that the path taken by the print-head is as shown in the diagram.

ADdress 1880 to use.



WAR GAMER'S DELIGHT

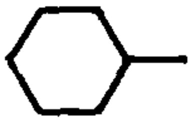
The full impact of this effect is shown on the next page.

The first thing you notice about the program is a set of values at the beginning which the printer does not recognise. This means they must be Machine Code values for an 'operations' program for the Z80. And they are.

The program produces a honey-comb pattern.

Anyone into war games will soon recognise the possibilities of the honey-comb as a playing board. The reason is each block has 6 borders, increasing the possible moves and thus the strategy, over a regular field of squares.

This shape is created using a picture element of a hexagon attached to a straight line thus:



This pattern is repeated 4 times across the paper and then a move to a new starting point -450,-86 down the paper.

The co-ordinates of the new starting point can be explained as follows:

After each picture element is drawn, the printer is initialized. This means that the present co-ordinate of the pen is taken as 00,00.

This gives us a value of 450,00 for the commencement of the 4th picture element with reference to the origin.

The next row of hexagons commence at the left-hand edge, which is -450 with reference to the above X co-ordinate and a y value of -86, with reference to the y value above.

The only way to understand how the honey-comb has been produced is to decode the listing.It contains two loops, one to draw the picture element and the other to count-to-4 across the screen.

Write each of the ASCII codes in a single file and alongside it place the printer value it reprasants.

You can experiment further by making the hexagons smaller. This will use a 2-digit ASCII value for the length of the sides. In the program, the original 3-digit ASCII values have been converted to 2-digit by using 00 for the 3rd value.

WAR GAMER'S DELIGHT

The first 18H bytes (31 bytes is the MAIN program and this contains the instructions to fetch one byte of data from the printer program and send it to the printer.

Data for the printer is stored in the form of a BYTE TABLE and starts at **0820.**

The main program is divided into two separate parts. **0800 - 080F** is a loop which loads the printer program and runs it 4 times.

811 - 81D loads the data for the MOVE commands and each piece of data is sent to the printer until **FF** is detected.

The count-to-4 operation is performed by **DJNZ** (at **80F**) which automatically decrements register B by ONE on each pass of the loop until it becomes zero.

The program then advances to loading HL register-pair with the contents of memory location **0860** and this instruct the print-head to move to the left-hand edge and down the paper to a new starting point. The main program then jumps to the start (**0800**) via instruction **JR Z E7** (at **817**).

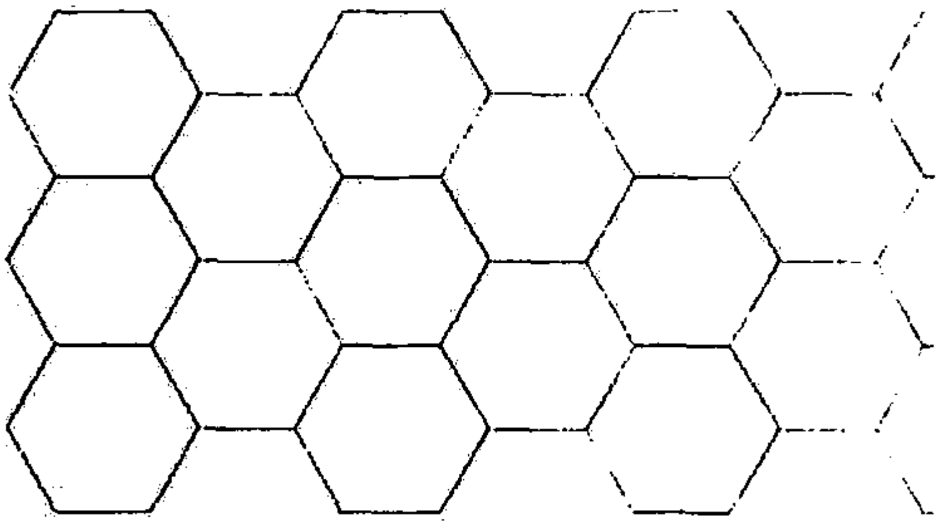
800	06	LD B,04
801	04	
802	21	LD HL,0820
803	20	
804	08	
805	7E	LD A,(HL)
806	FE	CP FF
807	FF	
808	28	JR Z 05
809	05	
80A	D3	OUT (06),A
80B	06	
80C	23	INC HL
80D	18	JR F6 (to 805)
80E	F6	
80F	10	DJNZ F1 (to 802)
810	F1	
811	21	LD HL 0860
812	60	
813	08	
814	7E	LD A,(HL)
815	FE	CP FF
816	FF	
817	28	JR Z E7 (to start)
818	E7	
819	D3	OUT (06),A
81A	06	
81B	23	INC HL
81C	18	JR F6 (to 814)
81D	F6	
81E	FF	NOT USEO
81F	FF	NOT USED

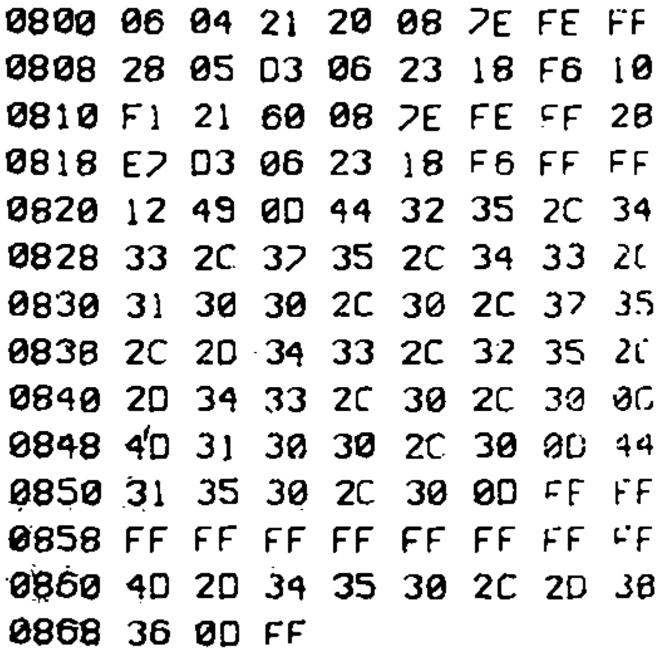
DATA FOR PRINTER:

820	12	Graphics Mode
821	49	Initialize
822	0D	CR
823	44	Draw
824	32	2
825	35	5
826	2C	,
827	34	4
828	33	3
829	2C	,
82A	37	7
82B	35	5
82C	2C	,
82D	34	4
82E	33	3
82F	2C	,
830	31	1
831	30	0
832	30	0
833	2C	,
834	30	0
835	2C	,
836	37	7
837	35	5
838	2C	,
839	2D	-
83A	34	4
83B	33	3
83C	2C	,
83D	32	2
83E	35	5
83F	2C	,
840	2D	-
841	34	4
842	33	3
843	2C	,
844	30	0
845	2C	,
846	30	0
847	0D	CR
848	4D	Move
849	31	1
84A	30	0
84B	30	0
84C	2C	,
84D	30	0
84E	0D	CR
84F	44	Draw
850	31	1
851	35	5
852	30	0
853	2C	,
854	30	0
855	0D	CR
856	FF	End

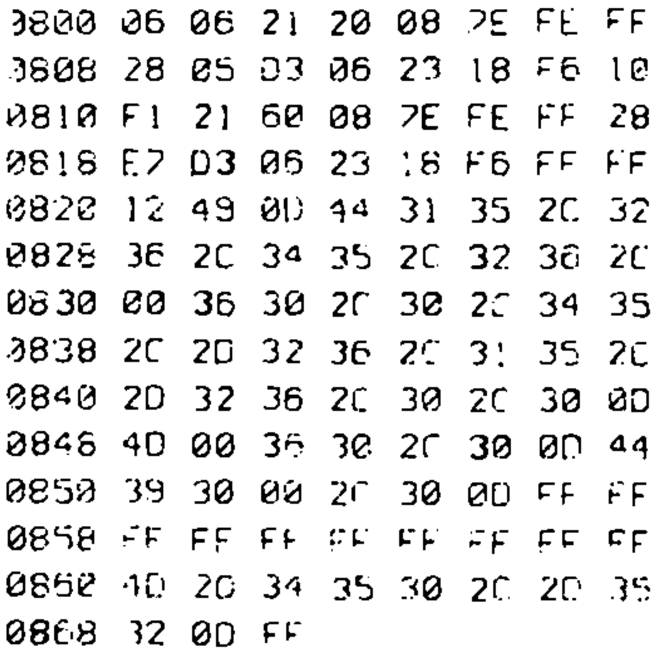
Data for MOVE COMMANDS:

860	4D	Move
861	2D	-
862	34	4
863	35	5
864	30	0
865	2C	,
866	2D	-
867	38	8
868	36	6
869	0D	CR
86A	FF	End

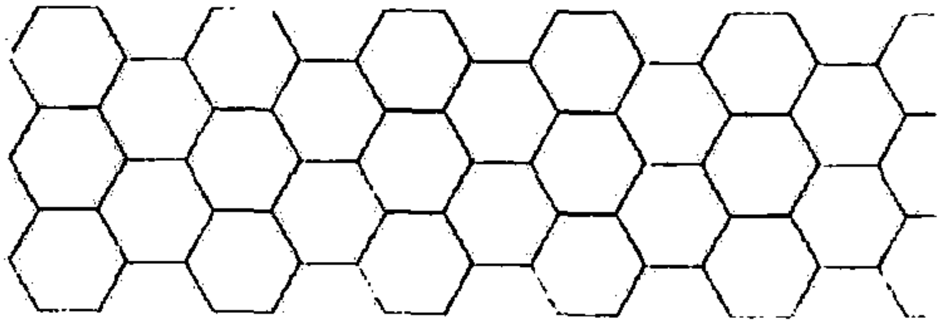




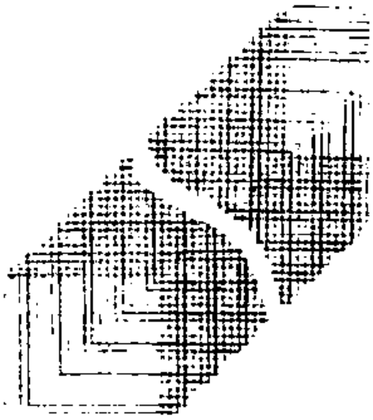
```
0800 06 04 21 20 08 7E FE FF
0808 28 05 D3 06 23 18 F6 10
0810 F1 21 60 08 7E FE FF 28
0818 E7 D3 06 23 18 F6 FF FF
0820 12 49 0D 44 32 35 2C 34
0828 33 2C 37 35 2C 34 33 2C
0830 31 30 30 2C 30 2C 37 35
0838 2C 2D 34 33 2C 32 35 2C
0840 2D 34 33 2C 30 2C 30 0D
0848 4D 31 30 30 2C 30 0D 44
0850 31 35 30 2C 30 0D FF FF
0858 FF FF FF FF FF FF FF FF
0860 4D 2D 34 35 30 2C 2D 38
0868 36 0D FF
```

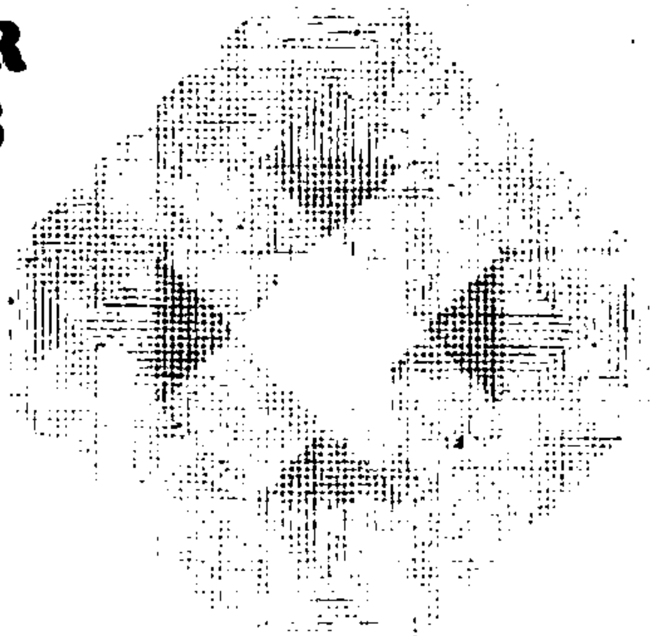


```
0800 06 06 21 20 08 7E FE FF
0808 28 05 D3 06 23 18 F6 10
0810 F1 21 60 08 7E FE FF 28
0818 E7 D3 06 23 18 F6 FF FF
0820 12 49 0D 44 31 35 2C 32
0828 36 2C 34 35 2C 32 36 2C
0830 00 36 30 2C 30 2C 34 35
0838 2C 2D 32 36 2C 31 35 2C
0840 2D 32 36 2C 30 2C 30 0D
0846 4D 00 36 30 2C 30 0D 44
0850 39 30 00 2C 30 0D FF FF
0858 FF FF FF FF FF FF FF FF
0860 4D 2D 34 35 30 2C 2D 35
0868 32 0D FF
```



COMPUTER GRAPHICS





```
800   06  06  3E  0A  D3  06  10  FC
808   3E  12  D3  06  11  60  08  06
810   08  C5  06  04  21  80  08  7E
818   FE  FF  CA  23  08  D3  06  23
820   C3  17  08  10  EF  3E  11  D3
828   06  3E  1D  D3  06  AF  D3  06
830   3E  12  D3  06  C1  10  DA  1A
838   FE  FF  C2  42  08  3E  11  D3
840   06  C7  32  9A  08  13  1A  32
848   9B  08  13  1A  32  9D  08  13
850   1A  32  9E  08  13  C3  0F  08
858   00  FF  00  FF  00  FF  00  FF
860   00  34  2D  34  2D  34  2D  34
868   2D  34  00  34  FF  FF  FF  FF
870   FF  FF  FF  FF  FF  FF  FF  FF
878   FF  FF  FF  FF  FF  FF  FF  FF
880   49  2C  44  38  30  2C  30  2C
888   38  30  2C  2D  38  30  2C  30
890   2C  2D  38  30  2C  30  2C  30
898   0D  4D  00  34  2C  00  34  0D
8A0   FF  FF  FF  FF  FF  FF  FF  FF
```

The Machine Code listing required to produce the DIAMOND.

Being able to draw some of the basic shapes (as we have shown), opens up a whole new world of computer graphics.

If we take the box-shape, we can produce a very effective pattern simply by re-defining the start co-ordinates and repeating the shape many times. The result can be anything from a 'check-tie' to an irregular octogon.

The colourful patterns which can be obtained (of which we can only see the result in black and white) is produced by a combination of drawing, shifting and colour-changing. The first of these to be investigated will be an irregular octogon or DIAMONO.

We have already outlined the structure of the program and briefly it is a set of instructions which are loops. Each sets a particular condition and then decrements on each pass.

For the diamond shape, a square is generated at the origin, 00,00 via the program at **0880. The lengths of the sides of the square are 80.** When the 4 sides have been drawn, the pen lifts off the paper and moves to a new origin with co-ordinates 04,04. The program is now up to location **083C.** It then jumps to **0842.** The contents of the accumulator (which is the value at location **0860** i.e. **00**) is loaded into **089A.** Register pair DE is incremented and now looks at location **0861.** The value **34** is loaded into the accumulator. At **0847** the contents of the accumulator is loaded into location **089B.**

So far, the program at **0880** has not been altered but the next two sets of load instructions will change the data at **89D, 89E** from **00 34** to **2D 34** and this will create the movement in the negative direction.

Complete decoding of the above listing, with explanations.

```
800  06  06     LD B,06          Load B with 6
802  3E  0A     LD A,0A          Load A with the Forward Fead instruction
804  D3  06     OUT (06),A       OUT to the printer pert
806  10  FC     DJNZ 0804        create 6 ieops ef forward feed
808  3E  12     LD A,12          Select the Graphics Mode
80A  D3  06     OUT (06),A       OUT to the printer
80C  11  60 08  LD DE,0860       Load DE with start ef Oiraction Change TABLE
80F  06  08     LD B,08          Sets number ef colour changes befere a diraction change
811  C5         PUSH BC          Save B. B must be paired with C to be saved
812  06  04     LD B,04          Sots number ef squares for each celour
814  21  80 08  LD HL,0880       Load HL with start of DRAWING TABLE
817  7E         LD A,(HL)        Load the data at 880 into A
818  FE  FF     CP FF            detocts ond ef TABLE
81A  CA  23 08  JP Z,0823        At end of tablo, jump to 823
81D  D3  06     OUT (06),A       OUT date valua at 880 to printer
81F  23         INC HL           Increment to 881, 882 etc
820  C3  17 08  JP 0817          Jump to 817 to incremant through Drawing Teble
823  10  EF     DJNZ 0814        Leap ORAWING TABLE 4 times
825  3E  11     LD A,11          Changa to TEXT MDDE
827  D3  06     OUT (06),A       OUT to printer
829  3E  1D     LD A,1D          NEXT COLOUR
82B  D3  06     OUT (06),A       OUT to printer
82D  AF         XOR A            Clear A
82E  D3  06     OUT (06),A       OUT to printer
830  3E  12     LD A,12          Select GRAPHICS MODE
832  D3  06     OUT (06),A       OUT to printer
834  C1         POP BC           Get B from STACK. Actuelly BC.
835  10  DA     DJNZ 0811        Oecrement B and jump to 811 for 6 loops
837  1A         LD A,(DE)        Lead A with data at 860 etc
838  FE  FF     CP FF            Detects end of DIRECTION CHANGE pregram
83A  C2  42 08  JP NZ,0842       If not zero, jumpa te 842
83D  3E  11     LD A,11          If zero, change to TEXT MODE
83F  D3  06     OUT (06),A       DUT to printer
841  C7         RST 0            END OF PROGRAM.★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★
842  32  9A 08  LD (089A),A      Load first byte of Dircation Changa table into locatlen 088A
845  13         INC DE           Increment DIRECTION CHANGE table
846  1A         LD A,(DE)        Load next byte of Oirection Change table into A
847  32  9B     LD (089B),A      Loed this byte into Iscatian 0898
84A  13         INC DE           Increment ths DIRECTION CHANGE table
84B  1A         LD A,(DE)        Load the third byte into the accumulater
84C  32  9D 08  LD (089D),A      Lead this third byte into loaction 089D
84D  13         INC DE           Incremant the OIRECTIDN CHANGE TABLE
84E  1A         LD A,(DE)        Load tha faurth byte of the direction change tablo into A
84F  32  9E 08  LD (089E),A      Load this leurth byte into location 089E
850  13         INC DE           Increment the DIRECTION CHANGE table ready for next
851  C3  0F 08  JP 080F          Jump te 80F to cammence the next diraction
```

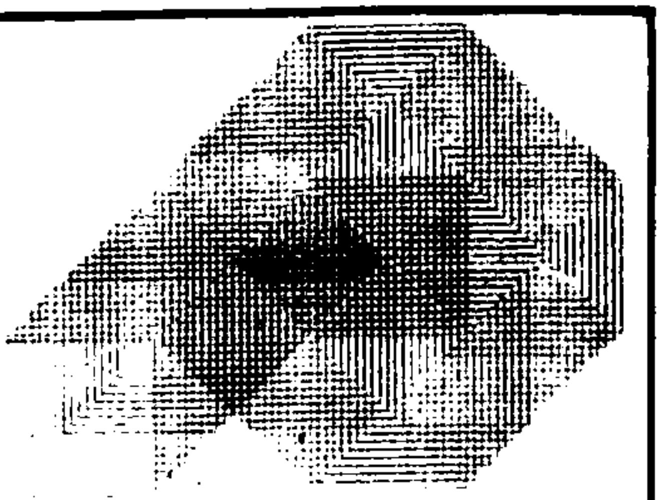
The program will then jump to 80F and draw the second side of the diamond.

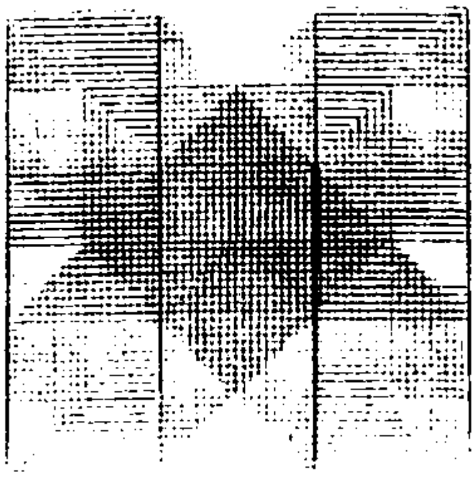
On the next pass, the register pair DE will be looking at locations 0864, 0865, 0866, and 0867. This will change locations 089A, 089B, 089D and 089E to 2D 34 2D 34 and thus the third side of tha diamond will be drawn.

Via the same reasoning, the 4th side of tha diamond will be completed.

Try experimenting and changing this program to produce other patterns. We have included two examples on the right and will be continuing with these and more 'add-ons' in the next issue.

The aim of COMPUTER GRAPHICS is to be able to produce 'forms' and ruled work for invoices etc. and place information in correct locations. It also gives you an understanding of ROBOT movement, an area we all would like to investigate.





860	00	0
861	34	4
862	2D	—
863	34	4
864	2D	—
865	34	4
866	2D	—
867	34	4
868	2D	—
869	34	4
86A	00	0
86B	34	4

880	49	= I
881	2C	= ,
882	44	= Draw
883	38	= 8
884	30	= 0
885	2C	= ,
886	30	= 0
887	2C	= ,
888	38	= 8
889	30	= 0
88A	2C	= ,x
88B	2D	= —

88C	38	= 8
88D	30	= 0
88E	2C	= ,
88F	30	= 0
890	2C	= ,
891	2D	= —
892	38	= 8
893	30	= 0
894	2C	= ,
895	30	= 0
896	2C	= ,
897	30	= 0

898	0D	= CR
899	4D	= Move
89A	00	= 0
89B	34	= 4
89C	2C	= ,
89D	00	= 0
89E	34	= 4
89F	0D	= CR
8A0	FF	= end

The Direction Change table and Drawing table with decoded values.

MON-1B

LOOKING AT THE REGISTERS

The MONITOR ROM for the TEC-1A (it can also be fitted to the TEC-1) is a MON-1B. This ROM has the facility for looking at the registers.

This ROM is the result of a number of requests from reeders who needed to look at the contents of the various registers during the running of a program.

If you would like one of these updated ROMs, send your MON-1 or MON-1A plus $3.00 postage and we will re-burn your EPROM to include the additionel instructions.

There is a limit to when and where you can use the register facility but it can help enormously with dabugging programs.

For instance, it can let you know the progress of a program or delay routine simply by interrupting it part way through.

The way this facility works is as follows:

If you reset the computer while it is executing a program, by pressing the raset button ONCE, the contants of each of the registers is pushed onto a stack.

This steck starts at 0FF0 and increases downwerds to 0FD8.

To look at any of the registers, press reset once and key the address of the register you want to look at.

The following list identifies the location of each register:

Mem ADD:	Reg:	
0FF0		
0FEF	A	
0FEE	F	
0FED	B	
0FEC	C	
0FEB	D	
0FEA	E	
0FE9	H	
0FE8	L	
0FE7	IX	MSB
0FE6	IX	LSB
0FE5	IY	MSB
0FE4	IY	LSB
0FE3	A'	
0FE2	F'	
0FE1	B'	
0FE0	C'	
0FDF	D'	
0FDE	E'	
0FDD	H'	
0FDC	L'	
0FDB	I	
0FDA	—	
0FD9	Stack	MSB
0FD8	Pointer	LSB

Note: Reset clears tha I register and thus it will always equal 00.

Use **JP 0000** if you wish to look at **I.**

An alternate method of saving the registers is to insert a **JP 00 00** instruction in the program at the position you wish to investigate. This will cause a JUMP to the beginning of the MONITOR ROM where it will find a jump to the register-save routine.

This will enable you to exit a program at a pre-determined point and look at the registers. The contents will be shown in the data displays.

Pushing Reset twice will destroy the information.

This is the program at **05F0** which performs the 'REGISTER-SAVE' operation. Don't forget the Monitor ROM has an instruction at 0000 to Jump to **05F0.**

```
05F0   ED   LD (0FD8),SP
05F4   31   LD SP,0FF0
05F7   F5   PUSH AF
05F8   C5   PUSH BC
05F9   D5   PUSH DE
05FA   E5   PUSH HL
05FB   DO   PUSH IX
05FD   FD   PUSH IY
05FF   08   EX AF,AF'
0600   D9   EXX
0601   F5   PUSH AF
0602   C5   PUSH BC
0603   D5   PUSH DE
0604   E5   PUSH HL
0605   ED   LD A,I
0607   F5   PUSH AF
060B   C3   JP 0580
060B   FF   RST 38H
```

TALKING ELECTRONICS®

$2.20
$3.00NZ

Issue No 13

COMPUTER POWER SUPPLY

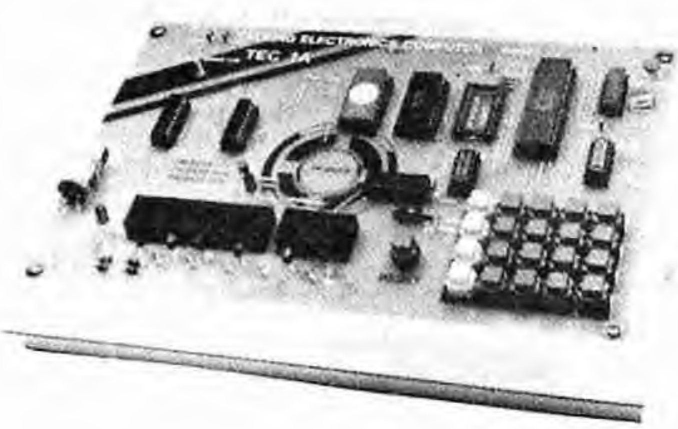
+5v FOR TTL
+12v FOR RELAYS
+30v FOR EPROM PROGRAMMING

2 PROGRAMMING AIDS

NON-VOLATILE RAM
EPROM PROGRAMMER FOR 2716/2732

LOGIC PULSER

KITT SCANNER



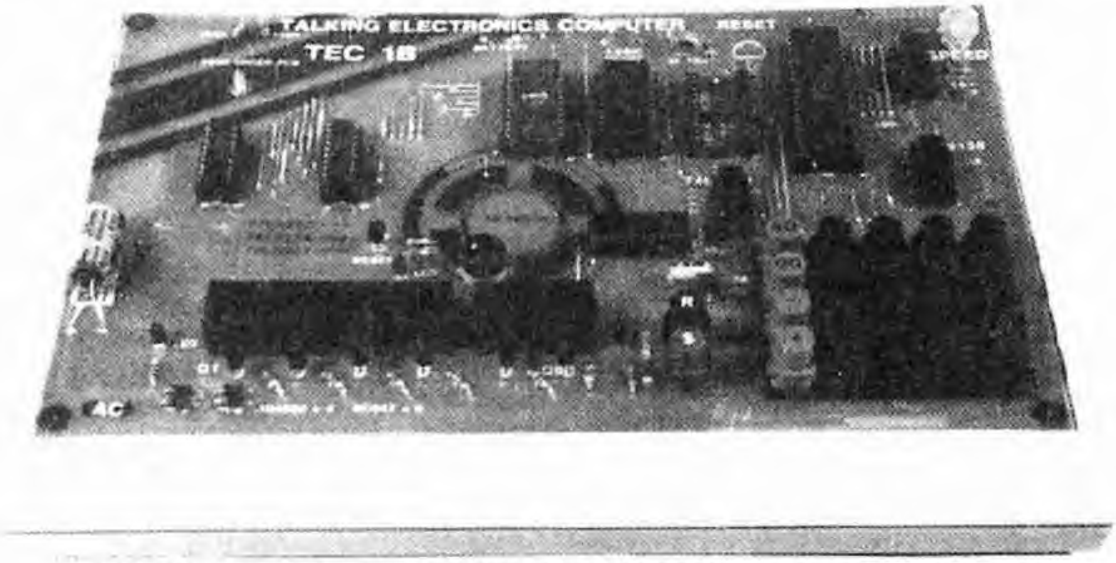
$98 COMPLETE
PLUS $6.00 POST

TALKING ELECTRONICS COMPUTER

PART IV

TEC-1A & TEC-1B

TEC 1A's can be converted to TEC 1B's by adding 1 push button, 1 47k resistor and 1 diode. Update to MON 2 and you have a SHIFT key for functions such as INSERT, DELETE etc.



TEC-1B board with SHIFT and RESET keys in foreground.

PC board: $21.00
Parts for 1B: $77.00
Case: $21.50
Post: $6.00 MAX.

FEATURES IN THIS ISSUE:
- ★ NON-VOLATILE RAM
- ★ EPROM BURNER

SEE ALSO:
TEC POWER SUPPLY on P. 23.

This is the fourth article on the TEC and introduces you to more Machine Code programming as well as two valuable add-ons.

The NON-VOLATILE RAM has been a real boon for assisting in program preparation for the MICROCOMP-1 project described in this issue.

Program can be written directly into RAM and by changing the switch, the contents will be retained for up to a year via the batteries mounted on the board.

This is the answer to .ll those requests from constructors wanting a battery backed-up system or tape-save facility. When the TEC is turned off, the contents of memeory will be saved and thus allow you to move the TEC from one location to another.

The RAM can also be used in place of an EPROM for the purpose of getting a system up and running. When you are satisfied with the design, the program can be transferred to EPROM.

This is where our second 'add-on' comes in. We have designed an EPROM BURNER to fit on the EXPANSION PORT socket.

With all the add-ons connected to the TEC, it was soon realized that the power required was more than could be supplied from a plug pack or 2155 transformer.

This led us to design a power supply exclusively for the TEC and at the same time include all the voltage values needed for the various projects.

So far we need 5v for the electronics, 12v for the relays and 26v for the EPROM BURNER.

The TEC POWER SUPPLY is capable of delivering these and can be expanded to about 1.4 amps at 5v by paralleling two 2155's.

Don't forget, the DC current capability of a 2155 is .7amps and NOT 1 amp and this has been covered in a previous article starting on page 5 of issue 11.

As you can see, one thing leads to another and we have sufficient add-ons to turn the TEC into a powerful programming tool.

The TEC itself has changed too. From the original TEC model, we improved the layout and upgraded the output latches to modern 20 pin types and mounted the regulator under the board so that it would not be broken off.

We have now upgraded the TEC to model 1B and this has seen the inclusion of a shift key.

This shift feature allows the keyboard to have a second command for each key and opens up a world of possibilities.

Two functions which have been lacking on the TEC are INSERT and DELETE. With the addition of the shift key, you will be able to make corrections to your programs and close up gaps as well as create locations for new instructions.

Those who have already built the TEC can add a shift key in one of two ways. The lowar RESET key can be converted into a SHIFT function by wiring a resistor and diode into circuit and connecting to the computer. The only problem with this is the upper RESET button. It will be difficult to access when the Video Display unit is mounted over the Z-80/EPROM area.

A better solution is to drill 4 holes near the lower RESET button and add the necessary components under the board.

The shift function is software controlled and you will need the updated MON 2 to get the shift key to work.

The MON 2 also includes a few othar improvements. The most noticeable of these is the location of the STACK. You will remember the original position of the stack is very close to the top of the 6116.

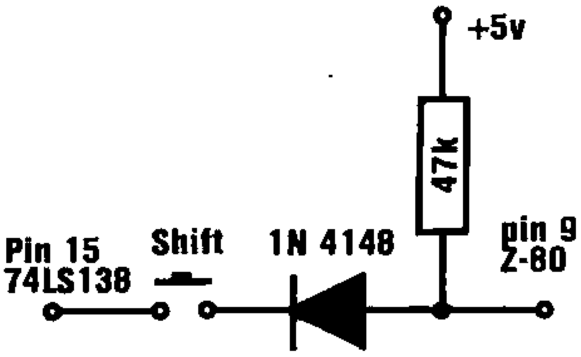
The main problem with this location is not knowing how far you can program before running the risk of hitting the stack.

The MON 2 places the stack at **08C0** and allows up to **C0** bytes to bé stored. There is still a risk of crashing the computer if a stack error occurs as the stack grows down to **0000** and restarts at **FFFF** and will eventually hit the top of your program. Between C0 and **FF** is a storage area for pointers, restarts, display buffer, keyboard buffer, register save area and for interrupts.

This means programming starts at **0900** up to **0FFF** with the on-board 6116 and you don't have the problem of landing in the stack area.

You can upgrade to MON 2 by sending in your ROM and it will be re-burnt to MON 2. The cost is $3.50 plus $1.50 post.

A shift button, 47k resistor and signal diode is available in a separate kit for 80c and this will double the capability of your computer.



Adding Shift to TEC-1 , TEC-1A

MON 2 has 6 other shift functions and we are in the process of writing more software for further functions.

By the time this issue is released, we will have completed this writing and will include documentation with the chip.

The cost of the TEC has risen to $98.00 end it looks like going even higher as the exchange rate for the Aust. dollar drops. But we want to keep the computar below the magical $100 mark for as long as possible.

We have now supplied over 1,000 computers, in 3 different models. Only the earliest model has been fully documented. The upgraded versions very only slightly and you should have no difficulty constructing them.

The reason for this is the simplicity of the board. Everything is fully identified on the overlay and requires only simple assembly.

Chances are it will operate first go but there is always a small possibility that something will be overlooked and it will not come on.

If you are caught in this situation, here is a run down on how to go about fixing it:

You will need a LOGIC PROBE and a MULTIMETER. A CONTINUITY TESTER (to be presented in next issue) will also be handy.

Firstly the visual checks:

If the displays fail to light-up and no sound is heard from the speaker, the most likely fault will be a broken track or poor solder connection. Turn the computer off and check each track with a multimeter switched to LOW-OHMs.

The regulator should get quite hot and should have 5v on the output lead. It must have at least 8v on the input lead to prevent voltage 'drop-out'.

The Z-80 will get quite warm, as will the output latch near the edge of the board.

The jumper near pin 1 of each latch should be checked. Only one must be inserted for each latch. This means you have two unused holes for each latch.

Check each of the keys for correct positioning. All flats must be DOWN.

The notch on each chip must also be DOWN.

Make sure all the pins of the IC sockets go through the holes in the PC board and are properly soldered. We have seen some pins doubled-up under the socket and not making contact with the tracks.

Check the capacitor near the speed control. It must be 100pf - not 100n. 100pf is indicatad by '100' or '101' on a ceramic capacitor whereas 100n is shown as '104' on a mono block or 100nS on a blue body.

Check for non-soldered lands, missing links and incorrectly soldered links. We inspected one project in which the builder had cut the links to the exact length BEFORE soldering and consequently one link did not go through the board completely. It was too short to be soldered but the builder didn't notice. He soldered the land with the result that the link looked as though it was soldered!

Finally check for solder-bridges between adjacent lands with a multimeter set to LOW ohms. Remove the chips to get an accurate reading.

Now for the 'in-depth' diagnosis:

1. Turn the TEC on and check for 5v out of the regulator. Check POWER-ON LED. Check for 5v on each of the chips: 74LS 273 - pin 20. 2716 - Pin 24. 6116 - pin 24. Z-80 - pin 11. 4049 - pin 1 74LS138 - pin 16. 74LS923 - pin 20.

2. Check clock frequency by putting logic probe onto pin 6 of Z-80.

3. Check RESET pin of Z-80 is HIGH.

4. Check NMI line, (pin 17 of the Z-80). It will go LOW when a key is pressed. If not, a switch may be faulty or the keyboard scan oscillator may not be working. Keyboard oscillator is part of the 74C923 and the frequency-setting capacitor and debounce cap are the 100n and 1uf electrolytic.

5. Check pin 19 of the Z-80 with a logic probe. If it is not pulsing, program is not getting through.

6. Logic probe pin 18 of the 2716. Pulses on this pin show the ROM is being accessed.

7. Pulses on pin 18 of the 6116 show RAM is being accessed.

8. No pulses via checks 5, 6 or 7 indicate the full byte in an instruction is not getting through. This may be due to a faulty address or data line.

9. Check Do (pin 9 on the 2716, for continuity to pin 9 of the 6116 and also pin 14 of the Z-80.) Check the other 7 data lines for continuity and also the 11 address lines.

10. With all chips still in circuit, check each pin with the one adjacent to it, for the 2716, 6116 and Z-80. Our continuity checker in issue 14 will be ideal but if you can't wait, a multimeter can be used. Remember protection diodes are contained in most chips and low value resistors may be present on some lines. Low values of resistance may be perfectly acceptable - you are looking for zero ohms or short-circuits between tracks.

11. Check pin 20 of the Z-80 - the IN/OUT REQUEST line. If it is not pulsing, the output of the computer may be putting a load on the data bus.

12. Remove the two output latches and place the negative lead of a continuity testar on one of the pins. Touch every other pin of the output latch with the other lead. Move the first lead and repeat until all pins have been tested. Do the same with the other latch.

This will check for shorts on the data bus as well as between pins of the display.

13 If these fail to locate the fault, ring us at TE. We may be able to help you over the phone. If not, send the TEC in a jiffy padded bag and we will see what the trouble is.

So far we have had about 20 TECs sent in for checking and repair. About 8 of them suffered from voltage surges. This occured when the constructor shorted leads together and/or dropped a screwdriver on the back of the board when the TEC was operating. This cen damage the EPROM, RAM and even the Z-80.

Don't let leads from the 'add-ons' dangle over the rest of the computer or let tha SELECT leads touch each other when fitting them over the pins on the PC board.

The TEC is really very robust and we haven't damaged a unit yet, even though we have three in constant use and they are lat running both day and night.

If you are careful with construction the TEC will work. But as with all pieces of electronic equipment, excess voltage will sound e death knoll.

While on this subject, we repaired two more unusual faults this month.

Both problems wera the same and occured like this:

When the constructor was building the TEC, one or more of the components were soldered without being fully pushed onto the board.

Some time later the constructor discovered the fault and proceeded to push the component into place while trying to resolder the joint.

The result was the land broke away form the copper track and created a hairline fracture which was not spotted.

If this occurs on either the address or data bus, the TEC will fail to come on.

If this happens, the first pin to check is each of the Chip Enable pins on the two output latches.

If a probe on these pins show they remain HIGH, they are not being accessed.

Next check the IN/OUT select chip (below the expansion port) and see if it is being activated by the Z-80. No information on pin 4 could indicate that the program is not getting to the Z-80.

This leads you to suspect either one of the data lines or one or more of the address lines. They may be broken, with the result that the Z-80 is not receiving a full byte of program.

Before you jump to this conclusion, check the Chip Enable pin of the EPROM (pin 18) and see that it is LOW. This will mean the 2716 is being accessed and it should be talking to the Z-80.

If the Chip Enable pin is HIGH, go to the ROM/RAM decoder (below the clock chip) and check pin 4 to see that the pin is being accessed.

If one bus line is missing, the Z-80 will get the wrong op-codes and the program will not flow correctly.

Before we continue with programming, here are a few notes on assembling the TEC-1B as some changes have been mada since the original notes in issue number 10.

The regulator is placed under the PC and bolted to the board via a 6BA nut and bolt. You can add heat fin if a number of add-ons are to be driven, but under normal circumstances, the regulator and board will dissipate the 1½ to 2 watts of heat.

The electrolytic has been changed to 1000mfd 25v and it lays flat on the board to keep a low profile.

The display drivers are slim-line types and 3 alternatives have been allowed for in the PC pattern. The overlay shows which links are to be added for the type chosen. Only ONE link must be used for each chip.

Finally a Z-80 or Z-80A can be used as the CPU chip. We are operating the TEC at 100kHz to 500kHz and this is well below the maximum speed for either type. A Z-80 will operate up to 2.5MHz and Z-80A up to 4MHz.

If any of the keys become worn, their contacts become erratic and sometimes a double-entry occurs. This can be overcome by increasing the value of the 1mfd on the 74c923 keyboard encoder to 4.7mfd or even 10mfd. This will mask out the contact bounce and produce a single pulse.

A 100n up to 10mfd can be used across the reset and it may be necessary to use the higher value if tha Z-80 does not reset properly.

A 10k or 20k cermet can be used as the speed control and it can be either a VTP or HTP type. The advantage of a cermet means you can use your fingers to turn the pot and don't require a small screwdriver.

SHIFT

The latest addition to the TEC softwara is a SHIFT function.

This enablas the number of functions to be increasad from 4 to 24.

It means each of the buttons can be programmed to perform a second function when combined with the SHIFT button.

To access this second function the SHIFT button must be pressed first and kept pressed while the desired key is pressad.

HOW DOES IT WORK?

The keyboard encoder uses 5 lines of the data bus and the remaining 3 lines are not used.

The SHIFT button is connected to one of these lines and the monitor program re-writtan to detect its status when the keyboard is read.

Five functions are currently available. More are in the pipeline and their details will be explained in future articles.

The 5 functions are:

SHIFT +

This is the **INSERT** function. It moves every byte in the program up to the next higher location and inserts 00 into the present address. This operation can be repeated any number of times to produce empty locations.

We have mentioned MON 2 allows programming to start at **0900** and the shift function operates in the area **0900** to **4000.** Addresses above **4000** are not catered for by the software but can be included if required.

Addresses below **0900** may cause a systems crash if you try to insert in this area as it is reserved for scratch pad, pointers and stack etc. Data below **0800** cannot be shifted as it is in ROM.

SHIFT – (shift, MINUS)

This is the **DELETE** key. It performs the opposite of INSERT. The data at the address currently being displayed is removed and all data above this address (and below **4000**) will be shifted DOWN one location. **3FFF** is loaded with **00.**

SHIFT ADdress

This function enables you to jump quickly to a particular location. Suppose you require to address **0A00** on a number of occassions. By pressing **SHIFT ADdress** the micro will jump to **0A00.** For this to happen, you must load a pointer location with the value **0A00,** then every time the SHIFT ADdress buttons are pressed, the display will show **0A00.** The pointer area is two bytes of memory located at **08D2 and 08D3.** By placing the **JUMP ADDRESS** at this location, the operation will be carried out.

We are loading these two locations directly into BC register pair via a 4-byte instruction **ED 4B D2 08** and for the register pair to be correctly loaded, we must place the lower byte first in memory and then the high byte. This means we must load location **08D2** with **00** and **08D3** with **0A.**

SHIFT 3

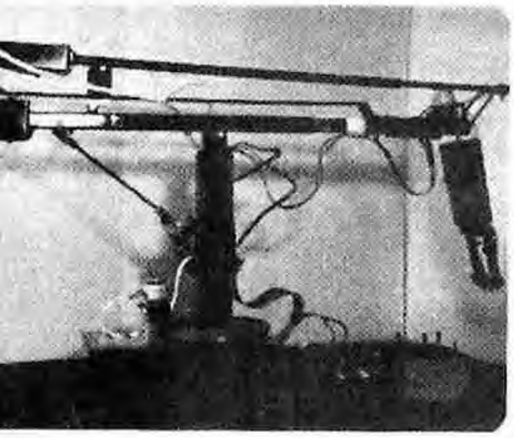
This function works exactly like SHIFT ADdress and enables you to have a second address to jump to. This time the pointer area is at **08D4** and **08D5.**

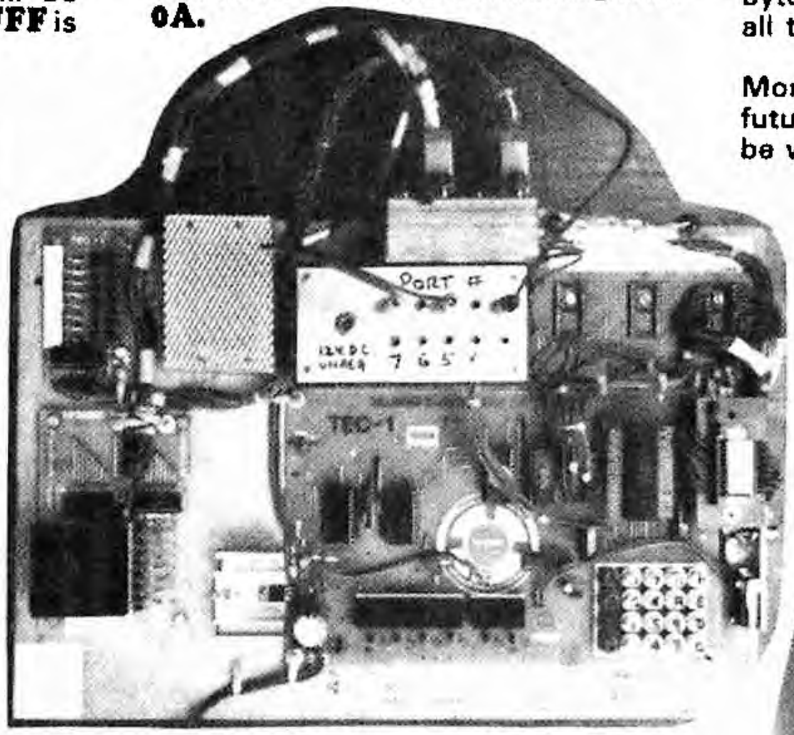
SHIFT 0

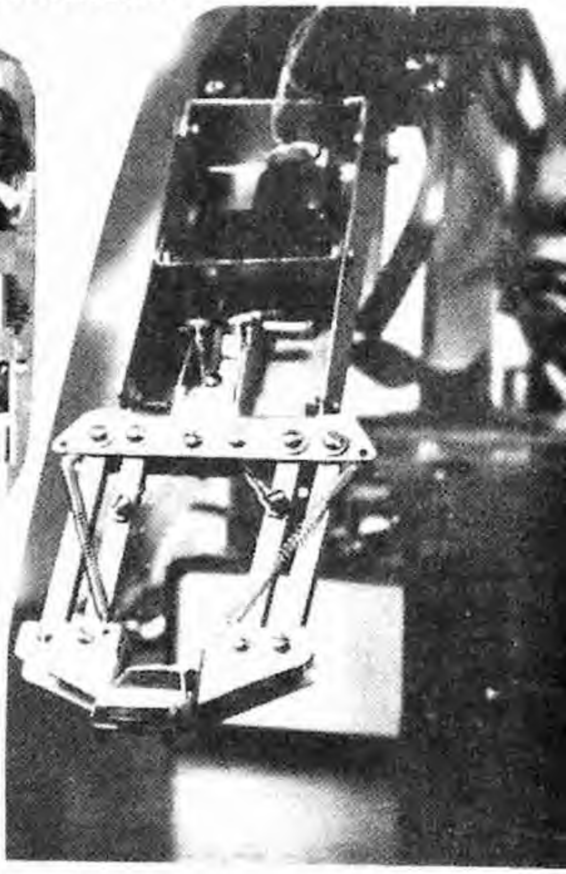
This is a search function. If you want to locate a value in a program or table, you could step through until it is located. This could take a long time. But with this function the value can be found very quickly. You can also locate the address of every other time it appears in a program.

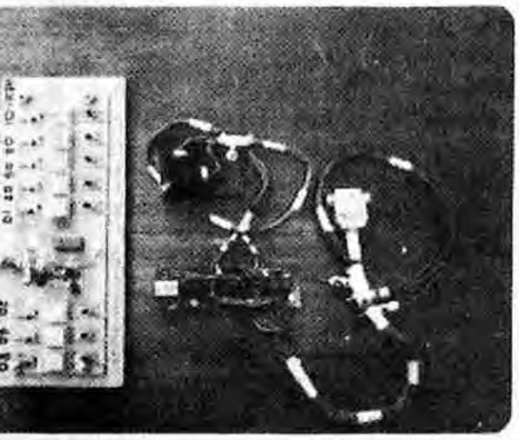
The value of the byte you are looking for is placed at **08E1.** Address the program you are testing and push **SHIFT 0.** The display will illuminate with the address of the byte you are looking for. Pushing SHIFT 0 again will display the second address of the byte. This can be continued to locate all the addresses.

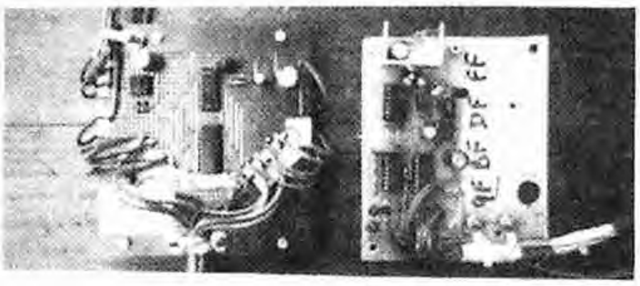
More function will be inclused in future monitors. Any suggestions will be welcome.

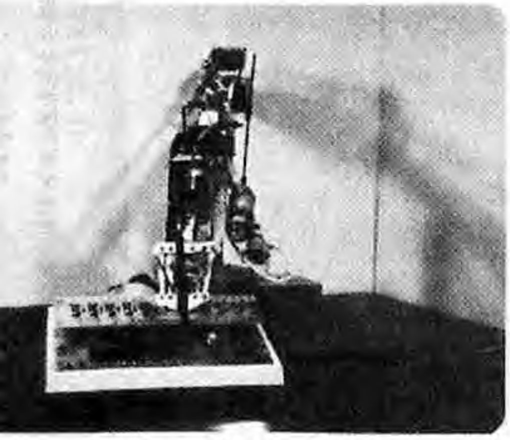


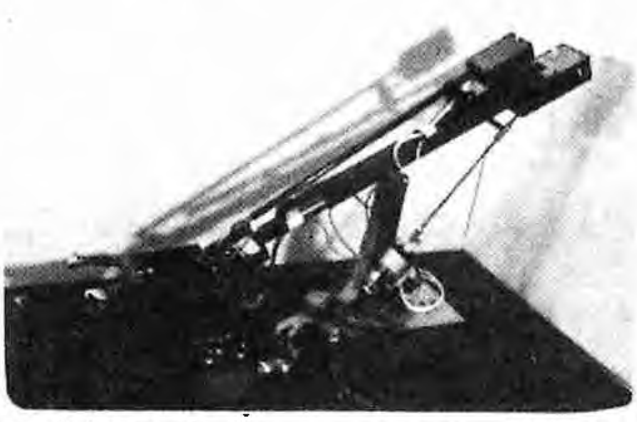


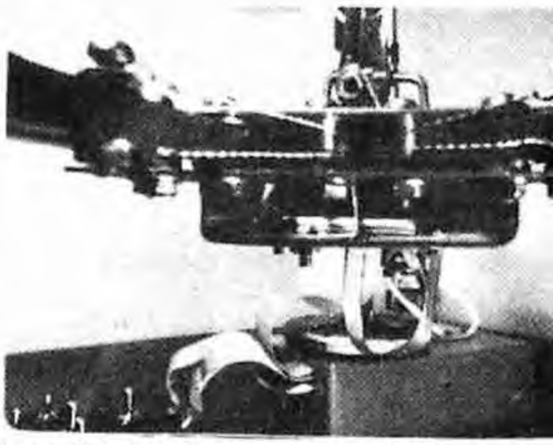












These photos show our science/electronics/computer teacher's add-ons to the TEC and Glen Robinson's Robot Arm. It is made entirely from easy-to-obtain hardware parts, gears, motors and sturdy pieces of steel. A larger photo will do it more justice and this we will show in the next issue.

SIMPLIFYING PROGRAMS

One of the most important features of machine code is the fact that it occupies the least amount of memory.

The skill is to make use of this fact.

If we take the simple program from issue 12 page 21 (1st column). RUNNING SEGMENT A ACROSS THE SCREEN", we can shorten the program by using the following set of instructions:

```
LD A,01         800     3E 01
OUT (02),A      802     D3 02
OUT (01),A      804     D3 02
PUSH AF         806     F5
LD DE 20FF      807     11 FF 20
DEC DE          80A     1B
LD A,E          80B     7B
OR D            80C     B2
JR NZ 080A      80D     20 FB
POP AF          80F     F1
RLCA            810     07
JR 0804         811     18 F1
```

This program saves 8 bytes but has the disadvantage that the delay routine cannot be used by any other programs as it is hidden in the listing.

The delay could be placed apart if desired.

Eight bytes may not seem many to save but is a start to efficient programming.

This is where the byte-saving occured:

The instruction RLCA is a one-byte instruction to shift the contents of the accumulator left. (It does not shift through the carry bit but sets it, as explained in data sheet 13.)

The listing contains a number of **JR** instructions (and a displacement byte). These are 2 byte instructions whereas a **CALL** instruction requires 3 bytes.

THE DISPLACEMENT BYTE.

As listings get longer and more complex, the value of the displacement byte requires a method for determining its value.

When the jump is 5, 10 or 15 bytes forward or backward, the displacement value can be obtained by counting the locations: such as 00, 01, 02, 03 or FE, FD, FC, FB, FA etc. But when the jump is 20, 30 or more locations, the value can be obtained via a simple mathematical procedure.

Determining the value of the displacement byte requires 6 steps. By following these you cannot make a mistake.

Step 1. Count, via normal counting, the number of bytes between the displacement byte and the location being jumped to. Include the location you wish to land on. e.g: Take the follnwing example:

```
11 FF 20
1B
7B
B2
20 dis
```

The number of bytes between **dis** and **1B** are: **20, B2, 7B, 1B.** These are counted as 1, 2, 3, 4. Thus the answer is 4.

We will select a higher value for our problem to emphasise the need for the procedure.

Suppose the number of locations we wish to jump back is 49.

Step 2: Convert 49 to a HEX value by dividing it by 16:

The answer is 31H

Step 3: Convert 31H to binary:

```
     3       1
  0011      0001
```

Step 4: Change each 1 to 0 and each 0 to 1:

Ans: 1100 1110

Step 5: Add 1 to the answer:

Ans: 1100 1111

Step 6: Convert to a HEX value:

C F

This is the value of the displacement byte required to achieve a backward jump of 49 bytes.

The machine code instruction will depend on the **JR** condition and will be one of the following:

28 CF, **20 CF,** or **18 CF**

The steps we have performed are called **TWO s COMPLEMENT.**

Using the knowledge we have gained, we will improve the **BACK and FORTH** program from P 15 of issue 12.

Mainly aiming at byte reduction, we will include a **BIT TESTING** instruction to prevent overshoot of the displays. Bit 0 in the accumulator is tested and if it is a '1', the program will cause a change in direction by rotating the accumulator in the opposite direction.

With these alterations in the program we will save about 12 bytes. Try the program:

```
LD A,01         800    3E 01
OUT (02),A      802    D3 02
OUT (01),A      804    D3 01
CALL DELAY      806    CD 00 0A
RLCA            808    07
BIT 6,A         809    CB 77
JR Z 0804       80B    28 F6
RRCA            80D    0F
OUT (01),A      80E    D3 01
CALL DELAY      810    CD 00 0A
BIT 0,A         813    CB 47
JR Z 080D       815    28 F6
JR 0809         817    18 F0
```

at 0A00:

```
F5
11 FF 20
1B
7B
B2
20 FB
F1
```

The program is required to test bit 6 in the accumulator. If it is found to be a '1', the contents of the accumulator is shifted in the opposite direction. Bit 0 is then tested and when found to be '1', the program jumps back and shifts the accumulator in the original direction.

BYTE TABLE. To use this table, the byte following the JR instruction is counted as BYTE ZERO. From this byte you count in either the positive or negative direction using decimal counting.

0	00	48	30	96	60	1	FF	-49	CF	-97	9F
1	01	49	31	97	61	-2	FE	-50	CE	-98	9E
2	02	50	32	98	62	3	FD	51	CB	-99	9D
3	03	51	33	99	63	4	FC	-52	CC	100	9C
4	04	52	34	100	64	-5	FB	-53	CB	-101	9B
5	05	53	36	101	66	-6	FA	-54	CA	102	9A
6	06	54	36	102	66	7	F9	55	C9	-103	99
7	07	55	37	103	67	-8	F8	56	C8	104	98
8	08	56	38	104	68	-9	F7	-57	C7	105	97
9	09	57	39	105	69	-10	F6	58	C6	-106	96
10	0A	58	3A	106	6A	-11	F5	59	C5	107	96
11	0B	59	3B	107	6B	-12	F4	60	C4	108	94
12	0C	60	3C	108	6C	-13	F3	61	C3	-109	93
13	0D	61	3D	109	6D	-14	F2	-62	C2	-110	92
14	0E	62	3E	110	6E	15	F1	-63	C1	-111	91
15	0F	63	3F	111	6F	-16	F0	-64	C0	-112	90
16	10	64	40	112	70	17	EF	65	BF	113	8F
17	11	65	41	113	71	18	EE	66	BE	114	8E
18	12	66	42	114	72	19	ED	-67	BD	-115	8D
19	13	67	43	116	73	-20	EC	-68	BC	-116	8C
20	14	68	44	116	74	21	EB	-69	BB	-117	8B
21	15	69	45	117	75	22	EA	70	BA	-118	8A
22	16	70	46	118	76	-23	E9	-71	B9	119	89
23	17	71	47	119	77	-24	E8	-72	B8	120	88
24	18	72	48	120	78	-25	E7	73	B7	-121	87
25	19	73	49	121	79	-26	E6	74	B6	-122	86
26	1A	74	4A	122	7A	-27	E5	75	B5	-123	85
27	1B	75	4B	123	7B	28	E4	76	B4	124	84
28	1C	76	4C	124	7C	-29	E3	77	B3	-125	83
29	1D	77	4D	125	7D	30	E2	-78	B2	126	82
30	1E	78	4E	126	6E	31	E1	-79	B1	127	81
31	1F	79	4F	127	7F	32	E0	-80	B0	-128	80
32	20	80	50			-33	DF	81	AF		
33	21	81	51			-34	DE	-82	AE		
34	22	82	52			-35	DD	-83	AD		
35	23	83	53			-36	DC	84	AC		
36	24	84	54			-37	DB	85	AB		
37	25	85	55			-38	DA	86	AA		
38	26	86	56			39	D9	-87	A9		
39	27	87	57			40	D8	-88	A8		
40	28	88	58			-41	D7	-89	A7		
41	29	89	59			-42	D6	-90	A6		
42	2A	90	5A			-43	D5	91	A5		
43	2B	91	5B			44	D4	-92	A4		
44	2C	92	5C			-45	D3	-93	A3		
45	2D	93	5D			-46	D2	-94	A2		
46	2E	94	5E			-47	D1	95	A1		
47	2F	95	5F			-48	D0	-96	A0		

INTRODUCTION TO COUNTING

A microprocessor system is ideally suited to counting situations. It can be programmed to count to any particular number then sound an alarm or operate a relay or even notify the near-completion of a run.

It can count UP or DOWN as well as count in sub-multiples.

Take the case of packing a box of TE magazines.

Firstly the operator requires a count of 10. Each 10 issues must be placed in opposite directions in a box to produce a level stack. The operator then needs to know when a count of 140 is reached, which represents a full box.

Finally the packers need to know how many boxes of magazines have been packed so that the delivery docket can be filled out.

This is effectively 3 counters which must ba interconnected to achived the raquired result. Ideally an audible signal should be produced at the end of each count of 140 so that the packer(s) can concentrate (day dream) on the job.

The chance of finding such a design is almost nil, except via individual modules which will have to be connected together to create the system. The cost of doing this would be about $300!!

But with a microprocessor system such as the TEC, all these up-down requirements are possible in the one unit, by simply providing a program!

The art of producing a suitable program is the content of this section.

We will start from the beginning and explain how counting is achieved, how to interface a 'count-button' and progress to producing a 3-digit up-down counter.

A count-down system is often used as it can be pre-programmed with a START VALUE and the counter decrements to zero. It then sounds a bell, activates a relay and resets to the pre-determined start-value.

After studying the 3-digit counter you will be able to create a 4, 5 or 6 digit counter and even incorporate sub-values to facilitate packing etc.

The counter can also be designed to have 2 concurrent tallies, one being permanently displayed while the other is available on call-up via the press of a button.

They would be displayed for a few seconds and fall back into memory.

Absolutely any combination, application or requirement can be catered for, it only requires programming.

To make it easy to understand, we have started with a simple program. But, as explained, this type of program soon runs out of capability. Thus a more complex system of time-sharing of the displays must be used.

But this too has limitations and finally an even more complex (as far as understanding is concerned) use of registars, must be employed.

With this high-level system, the scope is enormous. The system can be increased to 8 digits, two or more separate readouts, and have tally values available on call-up.

This is where we start . . .

Creating your own COUNTING MACHINE is one of the capabilities of our micro. You can produce a display which increments or decrements by a count of one or more on each press of a button. And the button doesn't have to be the '1' button. In our case we have used the '4' button to show that any button can be used.

By changing tha values in the 'look-up' table, you can create the up or down condition - something which is virtually impossible with discrete counting-chip construction.

You can even produce letters of the alphabet and increment each time 'Z' or 'F' or 'X' appears. You can do anything from counting by 2's to dividing by 'Z'.

For our first exercise we will produce a counter which counts to 9. This is a very simple program. Only one display will be accessed and thus we can output to it so that it turns on HARD, while the computer is in the HALT mode, waiting for an interrupt from the keyboard.

It is important to note the computer does not produce the numbers 0-9, the program creates them. The table at **0900** contains values which turn on various segments of the display to create tha numbers.

0-9 COUNTER

```
LD A,01         800   3E 01      The accumulator is loaded with 01 and outputted to port 01. This connects the
OUT (1),A       802   D3 01      cathode of the first display to earth.
LD HL,0900      804   21 00 09   Load HL pair with the address of the number table.
LD A,(HL)       807   7E         Load the first byte of the number table into the accumulator.
OUT (2),A       808   D3 02      Connect segments of the display to the positive rail to get first number.
LD B,0A         80A   06 0A      Register B is our 'counting register'. It counts 10 bytes from 0900 to 0909.
HALT            80C   76         HALT the program so that first number (0) will appear on the display.
CP 04           80D   FE 04      The program recognises only button '4'.
JR NZ Halt      80F   20 FB      If not button '4', go to HALT. If button '4' pressed, increment HL to look at 0901.
INC HL          811   23         The byte at 0901 is loaded into the accumulator.
LD A,(HL)       812   7E         The value at 0901 (28) creates the figure '2' on the display.
OUT (2),A       813   D3 02      Output 28 to port 02.
DJNZ Halt       815   10 F5      Register B is decremented and if it is not zero, the program goes to HALT.
JP Z 0800       817   CA 00 08   When register B is zero, the program jumps to START (0800).
```

at 0900

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

Type tha program into the TEC and press RESET, GO. The number '0' will appear on the display.

Press various buttons on the keyboard and notice that only button '4' advances the count.

Step through the table by pressing button 4.

1. Experiment with the program by creating the numbers on another display.
2. Create a down-count by inserting the table at **0900** in the opposite direction. i.e: **AF, EF, 29, E7, A7, 2E, AD, CD, 28, EB.**
3. Create a count-to-six by changing the value of **B** (**080A**) to **06.**
4. Create the letters A-F by adding their appropriate hex values to the table, select the correct value for **B**, change tha compare value to anable button 'C' to operate and step through the table you have producad.

TWO DIGITS

When two or more digits are to be displayed, the program must contain a multiplexing or time-sharing arrangement so that each display can show a number from 0 to 9 without interfering with the other. This means a HALT instruction cannot be used as only one display will remain alight!

The program must be constantly looping or 'running' so that both displays are kept on. Each tima the program cycles, it is looking for an interrupt from the keyboard and if one comes along, the program operates on the data it receives and compares it with the value 04. Depending on the result, the program will branch to one of two places.

The program below produces a count-to-99 using the '4' button as the input.

The basic structure of the program is quite simple and uses register pair HL to point to the address (at 0900) for the hex value needed to produce the numbers 0 to 9.

Register pair DE points to the hex value (again at 0900) needed to produce the 10's value.

Each of these register pairs are incremented and compared with FF to see if the end of the table has been reached. The increment of the DE register takes place when FF is detected on the 1's count. When the 10's count reaches the end of the table, the whole program is reset.

The computer does not know it is counting to 10. It merely knows it is incrementing through a table. You could put Chinese values on the display and count to 11, simply by changing the value of a few locations.

Here is the 0-99 program and an explanation of each step:

0-99 COUNTER

```
XOR A          800  AF        Set the accumulator to ZERO.
LD I,A         801  ED 47     Load the interrupt register with ZERO.
LD DE,0900     803  11 00 09  Load DE pair with address 0900.
LD HL,0900     806  21 00 09  Load HL pair with address 0900.   END OF START-UP.
XOR A          809  AF        Beginning of MAIN PROGRAM.   Clear Accumulator.
OUT (1),A      80A  D3 01     Turn OFF 1's display.
LD A,(HL)      80C  7E        Load accumulator with byte pointed to by HL pair.
OUT (2),A      80D  D3 02     Output to port 2.
LD A,01        80F  3E 01     Load accumulator with 1.
OUT (01),A     811  D3 01     Output accumulator to port 1.  Display is illuminated.
LD B,10        813  06 10     Register B is a COUNT REGISTER. Load it with 10 to create 16
DJNZ FE        815  10 FE     loops to turn on 1's display.
XOR A          817  AF        Clear Accumulator.
OUT (1),A      818  D3 01     Output 0 to port 1 to turn OFF display.
LD A,(DE)      81A  1A        Load accumualtor with byte at 0900 etc as pointed to by DE pair.
OUT (02),A     81B  D3 02     Output the value thus obtained to port 2.
LD A,02        81D  3E 02     Load the accumulator with 2
OUT (01),A     81F  D3 01     Output to port 1 to turn on 10's display.
LD B,10        821  06 10     Loed count register with 10 (decimal 16) and create 16 loops to
DJNZ FE        823  10 FE     turn on 10's display.
LD A,I         825  ED 57     Load the interrupt register into the accumulator.
CP 04          827  FE 04     Compare with 4  i.e. subtract 4 from I.  If the result is ZERO,
JP NZ 0809     829  C2 09 08  advance to 082C If the answer is NOT ZERO, go to 0809.
XOR A          82C  AF        Clear Accumulator.
LD I,A         82D  ED 47     Load tha Interrupt register with ZERO.
INC HL         82F  23        Increment register HL to point to address 0901 etc
LD A,(HL)      830  7E        Load the value at 0901 into the accumulator.
CP FF          831  FE FF     Compare the value obtained (eg 28) with FF. If equal, advance to
JP NZ 0809     833  c2 09 08  0836, if NOT equal, go to 0809.
INC DE         836  13        Increment DE.
LD A,(DE)      837  1A        Load the value pointed to by register DE into the accumulator.
CP FF          838  FE FF     Compare with FF to see if end of table has been reached
JP NZ 0806     83A  C2 06 08  If FF is reached, result will be zero. Advance to 083D. If not, go
JP 0800        83D  C3 00 08  JUMP TO START                                          to 0806.
```

at 0900:

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

The **CONDITIONAL JUMP** instruction requires explanation.

In the 00-99 counter program above, there are three places where the Z-80 will jump to another part of the program when a certain condition is met. The condition is **NZ** (NON ZERO). Let us explain how to interpret this:

From the program above:

```
LD A,I
CP 04
JP NZ 0809
```

These 3 lines state: The I register is loaded into the accumulator. The accumulator is compared with 04. Jump to 0809 is the result is NON ZERO.

How does the COMPARE statement work?

The **CP** operation is carried out like a subtract operation and the zero flag (Z flag) will be SET if the result is ZERO end RESET if the result is NON ZERO. This means it will be '1' if the answer is zero and '0' if the answer is not zero.

This is quite confusing because you have to deal with the negative of a negative. To simplify things we can use the word **MET** for ZERO. Thus we get:

NOT 04

JP NZ 0809

I = 04

Jump to **0809** if I is not 04 or go to the next line of the program if I = 04.

Now we come to the THREE DIGIT COUNTER. It has an UP/DOWN facility as well as CLEAR. Push +for increment, − for decrement and push ADdress to zero the display. The counter can also be preset by loading 0B03 and 0B04 with values as shown in the listing on the right:

```
PUSH AF        A00  F5
CALL 0A0D      A01  CD 0D 0A
POP AF         A04  F1
RRA            A05  1F
RRA            A06  1F
RRA            A07  1F
RRA            A08  1F
CALL 0A0D      A09  CD 0D 0A
RET            A0C  C9
```

```
AND 0F         A0D  E6 0F
LD HL          A0F  21 00 0C
ADD A,C        A12  85
LD L,A         A13  6F
LD A(HL)       A14  7E
LD (BC)A       A15  02
INC BC         A16  03
RET            A17  C9
```

THREE DIGIT COUNTER

```
START  LD BC 0B00      800   01 00 0B
       LD DE 0B03      803   11 03 0B
       LD A(DE)        806   1A
       CALL 0A00       807   CD 00 0A
       INC DE          80A   13
       LD A(DE)        80B   1A
       CALL 0A0D       80C   CD 0D 0A
       LD HL 0B02      80F   21 02 0B
       CALL SCAN       812   CD 00 09
       LD A,I          815   ED 57
       LD HL 0B03      817   21 03 0B
INC    CP 10           81A   FE 10
       JRNZ DEC        81C   20 0D
       LD A(HL)        81E   7E
       INC A           81F   3C
       DAA             820   27
       LD (HL)A        821   77
       JRNC START      822   30 20
       INC HL          824   23
       LD A(HL)        825   7E
       INC A           826   3C
       DAA             827   27
       LD (HL)A        828   77
       JR CLEAR        829   18 19
DEC    CP 11           82B   FE 11
       JRNZ RESET      82D   20 0D
       LD A(HL)        82F   7E
       DEC A           830   3D
       DAA             831   27
       LD (HL)A        832   77
       JRNC CLEAR      833   30 0F
       INC HL          835   23
       LD A(HL)        836   7E
       DEC A           837   3D
       DAA             838   27
       LD (HL)A        839   77
       JR CLEAR        83A   18 08
RESET  CP 13           83C   FE 13
       JRNZ CLEAR      83E   20 04
       XOR A           840   AF
       LD (HL)A        841   77
       INC HL          842   23
       LD (HL)A        843   77
CLEAR  LD A,FF         844   3E FF
       LD I,A          846   ED 47
       JR START        848   18 B6
```

SCAN

```
LD B,04        900  06 04
LD A(HL)       902  7E
OUT (02)A      903  D3 02
LD A,B         905  78
OUT (01)A      906  D3 01
LD B 50        908  06 50
DJNZ           90A  10 FE
DEC HL         90C  2B
LD B,A         90D  47
XOR A          90E  AF
OUT (01)A      90F  D3 01
RRC B          911  CB 08
JRNC           913  30 ED
RET            915  C9
```

at 0C00:

```
EB
28
CD
AD
2E
A7
E7
29
EF
AF
```

To make this program easy to understand, we have listed ONE COMPLETE CYCLE. Exactly as it is run by the computer. CALL ROUTINES have been included each time they are called and this makes the listing fairly long.

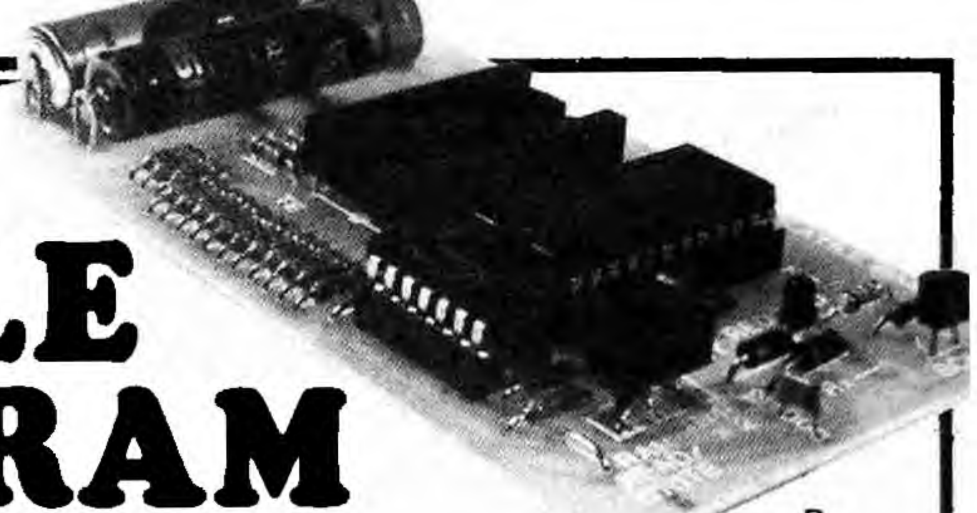
When the program is run for the first time, the display will show the values contained at 0B03 and 0B04. For the purpose of shewing how the program works, we will place 21 at 0B03 and 43 at 0B04. This will cause the display to shew 123 (the value 4 will not appear in this 3 digit counter).

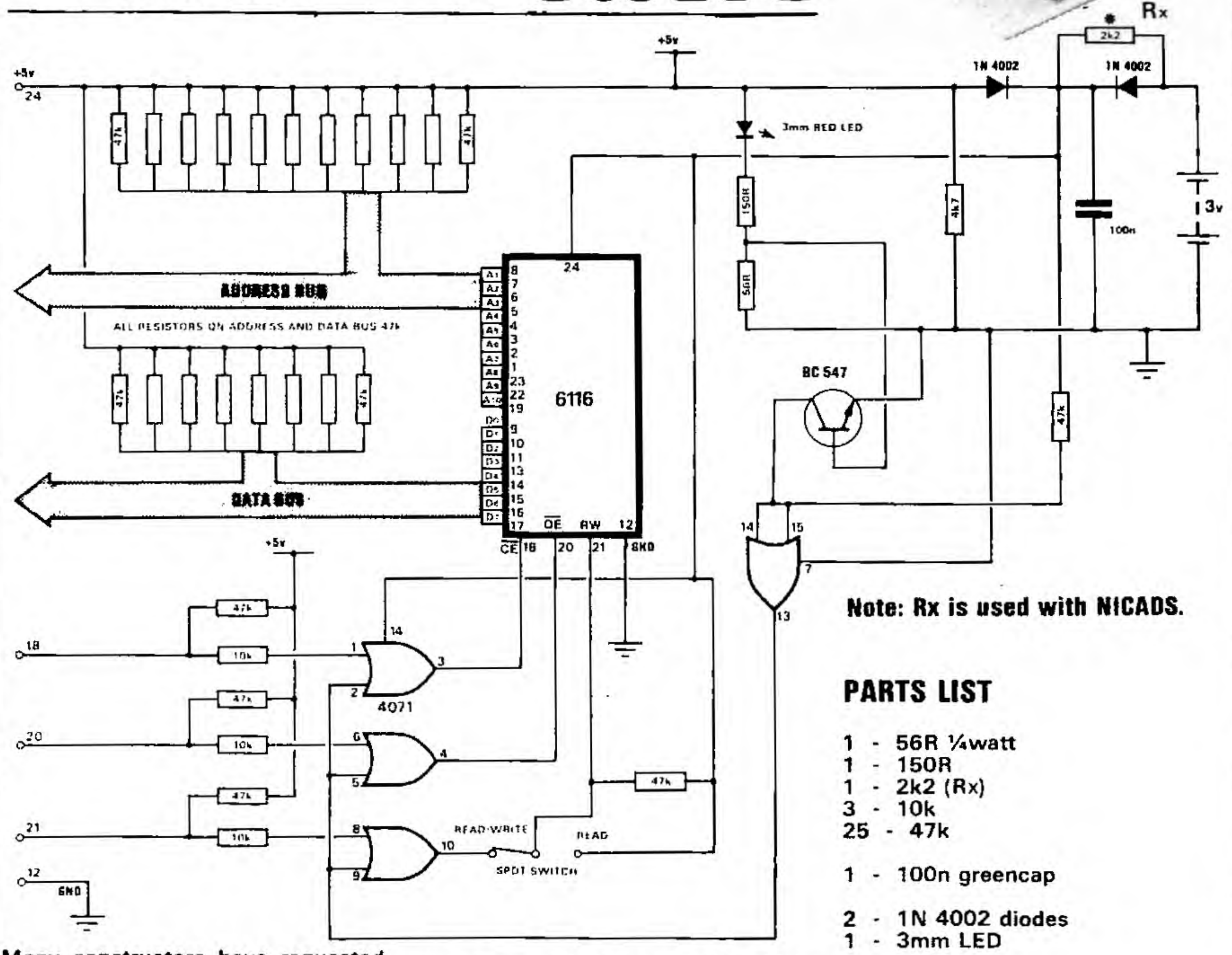
Follow through each of the steps and you will see how the program picks up data from the 'BUFFER ZONE' and converts it values which can be identified as numbers on the display. This program is being executed at more then 100 times per second!

```
START  LD BC 0B00    Location 0B00 stores the value of the units display
       LD DE 0B03    D is loaded with 0B and E is loaded with 03.
       LD A(DE)      Load two nibbles (21 in our example) into the accumulator
       PUSH AF       Save the accumulator
       AND 0F        This instruction zero's the high nibble leaving 01
       LD HL 0C00    H is loaded with 0C and L with 00
       ADD A,L       Add 00 to the accumulator to get 01 (01 is from above)
       LD L,A        Load the accumulator (it has 01 in it) into the L register
       LD A(HL)      Load the value at 0C01 (28) into the accum (HL is now 0C01)
       LD (BC)A      Load the value from the accumulator (28) into the BC register pair
       INC BC        Increment the BC register (it will become 0B01).
       POP AF        Fetch the accumulator (value 21) from the stack
       RRA           Shift the value 21 four places to the right
       RRA           so that the high bits will be transposed
       RRA           with the low bits. The result will be 12
       RRA
       AND 0F        Remove the 4 HIGH bits to get 02
       LD HL 0C00    H will be loaded with 0C and L with 00
       ADD A,L       Add 00 to the accumulator to get 02
       LD L,A        Load 02 into the L register
       LD A(HL)      Load the value at 0C02 (CD) into the accumulator
       LD (BC)A      Load the accumulaotr (it has 02 in it) into the address pointed to by 0C.
       INC BC        Increment the BC register (to 0BD2)
       INC DE        DE is incremented to 0B04
       LD A(DE)      The value at 0B04 (43) is loaded into the accumulator
       AND 0F        The HIGH nibble is cleared to get 03
       LD HL 0C00    H is loaded with 0C and L with 00
       ADD A,L       00 is loaded into the accumolator to get 03
       LD L,A        03 is loaded into L
       LD A(HL)      The vlaue at 0C03 (AD) is loaded into the accumulator
       LD (BC)A      Load AD into location 0B02.
       INC BC        The BC register pair is incremented to 0B03
       LD HL 0B02    Load H with 0B and L with 02
       LD B,04       Load B with 04
       LD A(HL)      Load the accumulator with the value at 0B02 (AD)
       OUT (02),A    Output AD to port 02.
       LD A,B        Load the accumulator with 04.
       OUT (01),A    Output 04 to port 01. This will turn on a,b,c,d,g to get '3'
       LD B,50       B is loaded with 50hex (five-oh or 80 in decimal)
       DJNZ          Perform a jump command for 80 loops
       DEC HL        HL now points to 0B01
       LD B,A        The accumulator (it contains 04) is loaded into B
       XOR A         Clear the accumulator
       OUT (01),A    Turn OFF the display
       RRC B         Shift register B right to get 02 (half its previous value)
       LD A(HL)      Load the value at 0B01 (CD) into the accumulator
       OUT (02),A    Output the value CD to port 2
       LD A,B        Load B (02) into the accumulator
       OUT (01),A    Output 02 to port 1. This turns on the second display and a,b,d,e,g. '2
       LD B,50       Load B with 50 (in hex)
       DJNZ          Perform 50 loops. This is 80 loops
       DEC HL        HL now points to 0B00
       LD B,A        Load 02 into B
       XOR A         Zero the accumulator
       OUT (01),A    Turn OFF the display
       RRC B         Rotate register B to the right to get 01
       LD A(HL)      Load the value at 0B00 (28) into the accumulator
       OUT (02),A    Output 28 to port 2
       LD A,B        Load 01 into the accumulator
       OUT (01),A    Output 01 to port 1
       LD B,50       Load B with 50
       DJNZ          This instruction creates 80 loops of delay-time
       DEC HL        HL is decremented but the 4th location is not used as you will see.
       LD B,A        Load 01 into B
       XOR A         Zero the accumulator
       OUT (01),A    Turn off the display
       RRC B         Register B is shifted and the carry bit is SET
       LD A,I        The accumulator is loaded with a value from the keyboard
       LD HL 0B03    H is loaded with BB and L with 03
INC    CP 10         The value 10 is compared with the accumulator
       DJNZ          If the two are the SAME, the program increments. If not it jumps to DEC.
       LD A(HL)      Load A with 21.
       INC A         Increase the value 21 to 22
       DAA           Decimal adjust the accumulator if needed (not in this case)
       LD (HL),A     Load 22 into the location 0BD3.
       JRNC start    Jump to start is no carry from DAA operation. If a carry is produced, i.e.
       INC HL        when 99 advances to 100 increment HL to 0B11
       LD A(HL)      Load the value at 0B11 into the accumulator
       INC A         Increment A
       BAA           Decimal adjust the accumulator if necessary
       LD (HL)A      Load the accumulator into 0B04
       JR CLEAR      Jump to CLEAR
CLEAR  LD A,FF       Load FF into the accumulator
       LD I,A        Load the accumulator into the interrupt vector register
       JR START      Jump to START
```

NON-VOLATILE RAM

PC Board: $3.60
Parts: $19.60





Many constructors have requested some means of saving the programs they produce on their TEC'c.

Most suggested a TAPE SAVE facility whereby they could load their program onto a cassette and hold it until required. This would allow the TEC to be turned off / or used for other tasks.

Tape save is a project which will appear as a future add-on and has certain advantages. Before we present a tape save we have designed a storage project using a 6116 CMOS RAM chip, which will be the next best thing.

It is a battery backed-up RAM which can be written into and then protected via a switch to become a Read Only Memory. It can be left connected to the TEC or removed at any time and the battery automatically takes over, keeping the contents in an unchanged state.

The 6116 RAM takes an amazing 2 microamps in the storage state (power-down state) and this represents little more than the natural deterioration of the cells.

Under these conditions the two back-up cells should last about 1 year.

Note: Rx is used with NICADS.

PARTS LIST

1	-	56R ¼watt
1	-	150R
1	-	2k2 (Rx)
3	-	10k
25	-	47k
1	-	100n greencap
2	-	1N 4002 diodes
1	-	3mm LED
1	-	BC 547 transistor
1	-	CD 4071 IC
1	-	6116 RAM
1	-	14 pin IC socket
1	-	24 IC socket
1	-	24 pin wire-wrap socket
1	-	24 pin DIP header
1	-	SPDT switch
2	-	AAA cells
		20cm tinned copper wire
1	-	**NON-VOLATILE RAM PC BOARD**

The main advantage of this form of memory is information is immediately accessible and does not have any loading delays as experienced with tape.

When producing Machine Code programs, it is not necessary to have a large RAM memory and a single 6116 will be sufficient for even quite a long program.

We have called this project NON-VOLATILE RAM and have already found it to be invaluable when developing programs for other computers and dedicated systems. It is easy to use and can be written into directly or filled from TEC memory.

Once the data has been deposited, the switch is changed to 'ROM' position and the information is protected.

HOW IS THIS DONE?

The 6116 RAM chip is the centre of the design. It is a low-power CMOS device with exceptionally low stand-by current. Many of the TEC owners will already have one of these chips. It is important to note that only the 6116 can be used as the N-MOS version 58725 consumes 4,000 times more current in stand-by mode.

The 6116 draws about 2 micro-amps whereas the 58725 consumes 8 milli-amps. If you have one of each, use the 6116 for the non-volatile project and retain the 58725 as the TEC RAM.

Theoretically the RAM in the TEC can be converted to battery back-up but this poses a problem as the control lines must be taken HIGH or LOW to prevent the RAM being written over during the time when the TEC is powering down.

We had difficulty in achieving a guaranteed result and opted for a separate RAM card. This allows the card to be transferred to other projects, enabling programs to be generated and corrected until they operate perfectly.

When the RAM card is plugged into the TEC it draws power via diode D1 while diode D2 prevents the voltage from charging the batteries. When the TEC is switched off, the batteries supply a potential to the chip via diode D2. Diode D1 prevents the TEC from drawing on the batteries.

The 2.4volts from the batteries (.6v is lost across diode D2) is sufficient to hold the data.

The LED and resistors R1, R2 form a voltage-loss detection circuit to switch the non-volatile RAM into stand-by mode.

They form a voltage-divider circuit for the base of the BC 547 transistor. When the voltage across the LED and resistor R1 is below 4v, the transistor switches OFF, allowing the inputs of the OR gate to go HIGH, via a buffer. This takes the Chip Enable, Output Enable and Read/Write lines HIGH, protecting the RAM contents from erasure.

All address and data lines are taken LOW to prevent them floating and thus wasting power.

USING THE RAM CARD

The 'on-board' cells will provide power to the chip for about 1 year and this makes it an ideal storage medium for saving programs.

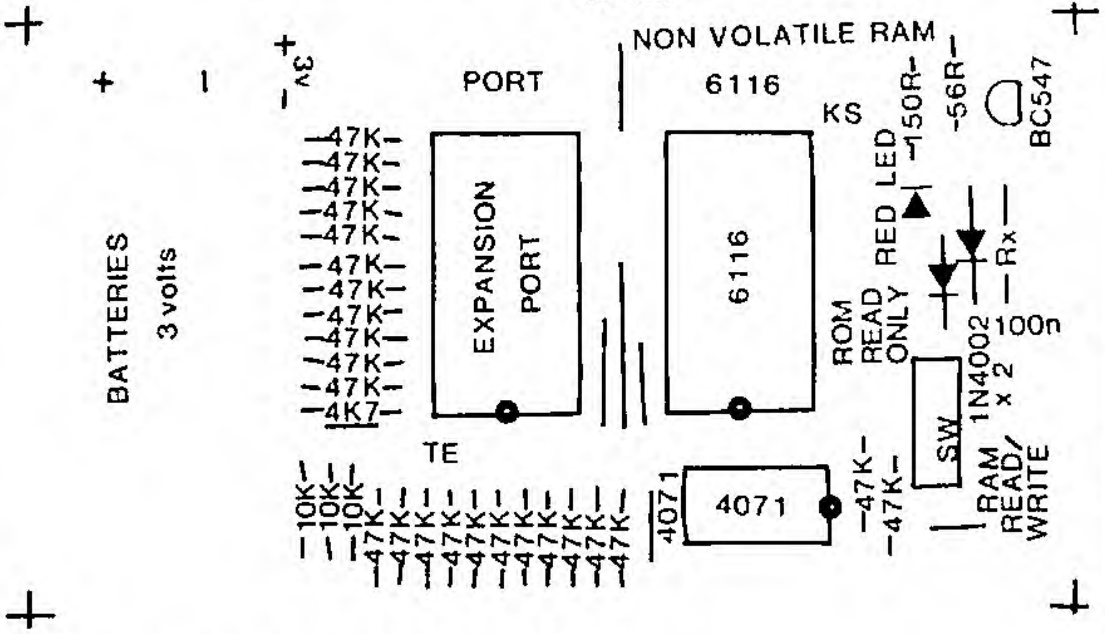
When inserting and removing the RAM from the TEC, the RESET button must be pressed. This will freeze the address and data bus and prevent any glitches from entering either the RAM or TEC. Remove and insert the RAM card quickly to prevent excess voltage appearing on the pins of the 6116.

If this does occur, the circuitry inside the 6116 may heat up excessively and cause the TEC to crash. The RAM will also lose its contents, but may not be permanently damaged.

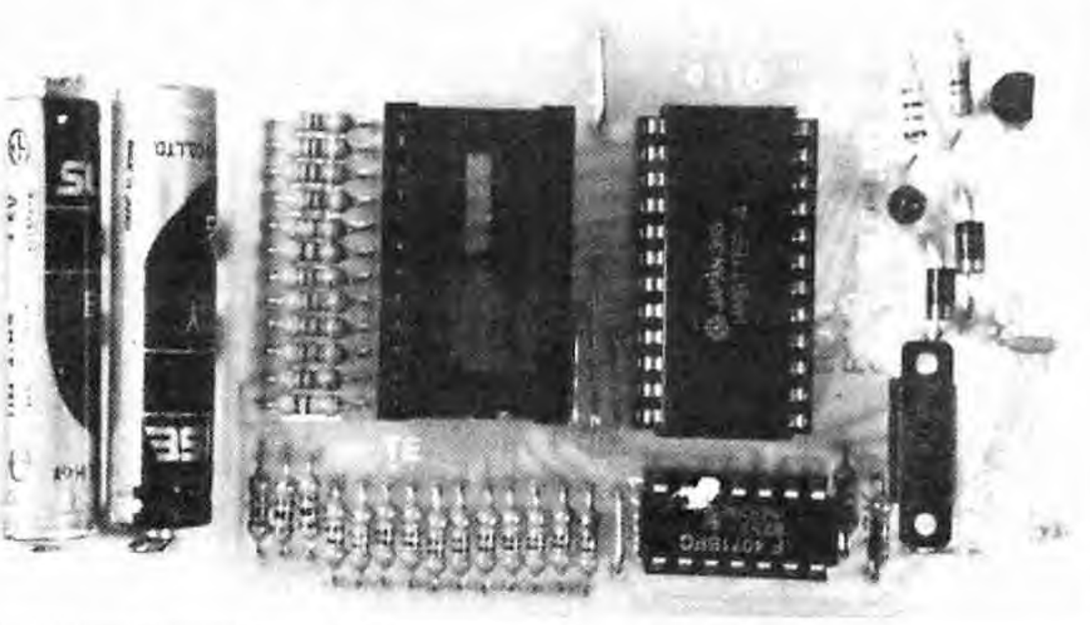
CONSTRUCTION

All the components are mounted on the top of the board in positions as shown by the overlay. The ROM/RAM SELECT is a slide switch and it is best to keep to a slide switch so that the writing on the board reads correctly.

The RAM card is connected to the TEC via a 24 pin wire wrap socket and component header plug soldered together to form a stand-off. The 6116 faces towards the switch as does the 4071 and this may mean the writing on the chip(s) is up-side-down.



See P. 75 for PC artwork.



The RAM card is accessed at the address of the socket in which it is placed. This means it is addressed at **1000** to **17FF** in the expansion port socket. It can also be placed in the RAM socket and is addressed at **0800** to **0FFF**. If placed in the EPROM socket it must already contain a start-up routine for the TEC and is addressed at **0000** to **07FF.**

When placed in the expansion port socket, it can be addressed at higher values by cutting pin 18 of the wire-wrap socket and taking a lead to one of the Chip Select pins near the edge of the board. The lowest of these is addressed as **1800** the next as **2000** and the next as **2800** etc.

If the RAM card is used in the monitor socket it can only be used in the READ MODE as this socket does not have a READ/WRITE line.

This situation also applies when using the RAM card in our dedicated computer project as described in the book: **ELECTRONICS FOR MODEL RAILWAYS.**

The RAM CARD can also be used to create programs for the Microcomp. Any of the programs described in the Microcomp article can be typed into the RAM and executed on the 'comp.

Remember, the Microcomp programs are designed exclusively for the 'comp as it has only a single output latch - the TEC has two output latches.

DUMP ROUTINE

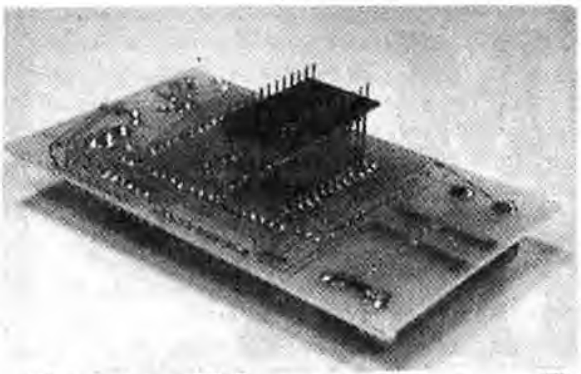
You can use the RAM CARD for many other applications and also transfer up to 2k of program into the card by loading the following into the TEC at **0800:**

```
          TO  11 00 10
        FROM  21 XX XX
 No of BYTES  01 YY YY
              ED B0
              C7
```

Where **XX XX** is the start of the program you wish to copy and **YY YY** is the number of bytes you wish to transfer. e.g: If the program to be copied is at **0900** and 80 bytes long, the program at **0800** is:

```
11 00 10
21 00 09
01 80 00
ED B0
C7
```

Note: If the program starts at **0900** and finishes at**090F**, you must insert **01 10 00** into the program because 0900 to 090F contains 16 bytes of program (10 hex bytes) Not **0F** bytes!



The wire-wrap and DIP header are soldered together to form a stand-off.

If you have written the program at **0800**, you can place the DUMP ROUTINE at **0900**. This is how to do it. Load the program, go back to the start of the program (0900) and push GO. Do not push RESET. The dump program will then be executed. The dump routine can be placed anywhere in RAM by using this method.

Suppose you want to copy the MONITOR ROM. Load the following Dump Routine into the TEC at 0800:

```
11 00 10
21 00 00
01 FF 07
ED B0
C7              Push RESET, GO.
```

Within a fraction of a second the monitor program will be loaded into the CARD. Change the Read/Write switch to READ and the contents will be protected. Remove the Monitor ROM from the TEC. Insert the CARD and turn the TEC on. It will start up as normal.

To remove a program from the CARD, it is best to fill it with FF's. This will be needed in later programming, when you want to transfer a program from the CARD to an EPROM.

The unused locations will contain FF and these can be burnt to any other value. A value such as 00 cannot be 'burnt down'. For more detail on this, see the EPROM PROGRAMMER project.

TO FILL THE CARD WITH FF's:

at 0800:

```
LD BC 07FF        01 FF 07
LD HL 1000        21 00 10
LD A,FF           3E FF
LD (HL)A          77
INC HL            23
DEC BC            0B
LD A,B            78
OR C              B1
JRNZ              20 F7
RESTART 0000      C7
```

TO FILL THE TEC WITH FF's:

The TEC RAM can be filled with **FF's** by loading the following into **0800:**

```
11 FF FF
D5
C3 03 08
Reset, GO.
```

This puts **FF FF** onto the stack and the stack increments downwards to **0800**! - until the computer crashes.

Both TEC and CARD can be filled at the same time by changing the first two lines to:

```
01 00 10
21 10 08
```

Using these programs, any number of locations can be filled, anywhere in RAM. They can be filled with any value such as **BB, CC, or 33** etc.

Programs can be transferred from the TEC to CARD and from CARD to TEC via the DUMP ROUTINE. Some examples are given in the EPROM BURNER article.

IF THE RAM CARD FAILS TO WORK:

There are three major faults which may occur with this project:

1. **You cannot write into memory.**
2. **Information in the RAM card is lost.**
3. **The TEC starts to play up.**

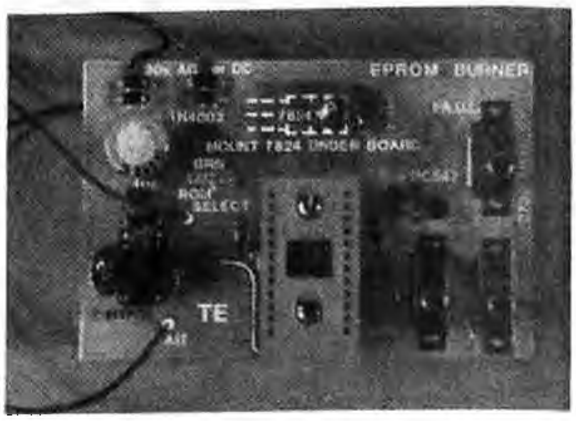
If it is not possible to write directly into the RAM card or dump into it via a DUMP ROUTINE, the fault will lie in the READ/WRITE line. This is pin 21 of the 6116. It must be low to be able to change the data. Test it with a logic probe or high impedance multimeter. Don't forget to address the RAM correctly (via the addresses given previously).

If the information in the RAM card gets lost, the fault will lie with pin 21 of the 6116. It may be floating or go LOW for brief periods so that noise and glitches enter the address and data lines to change the stored data. This problem can also be due to weak batteries or poor contact between the cell and the disk on the bottom of the cell. Try a different brand.

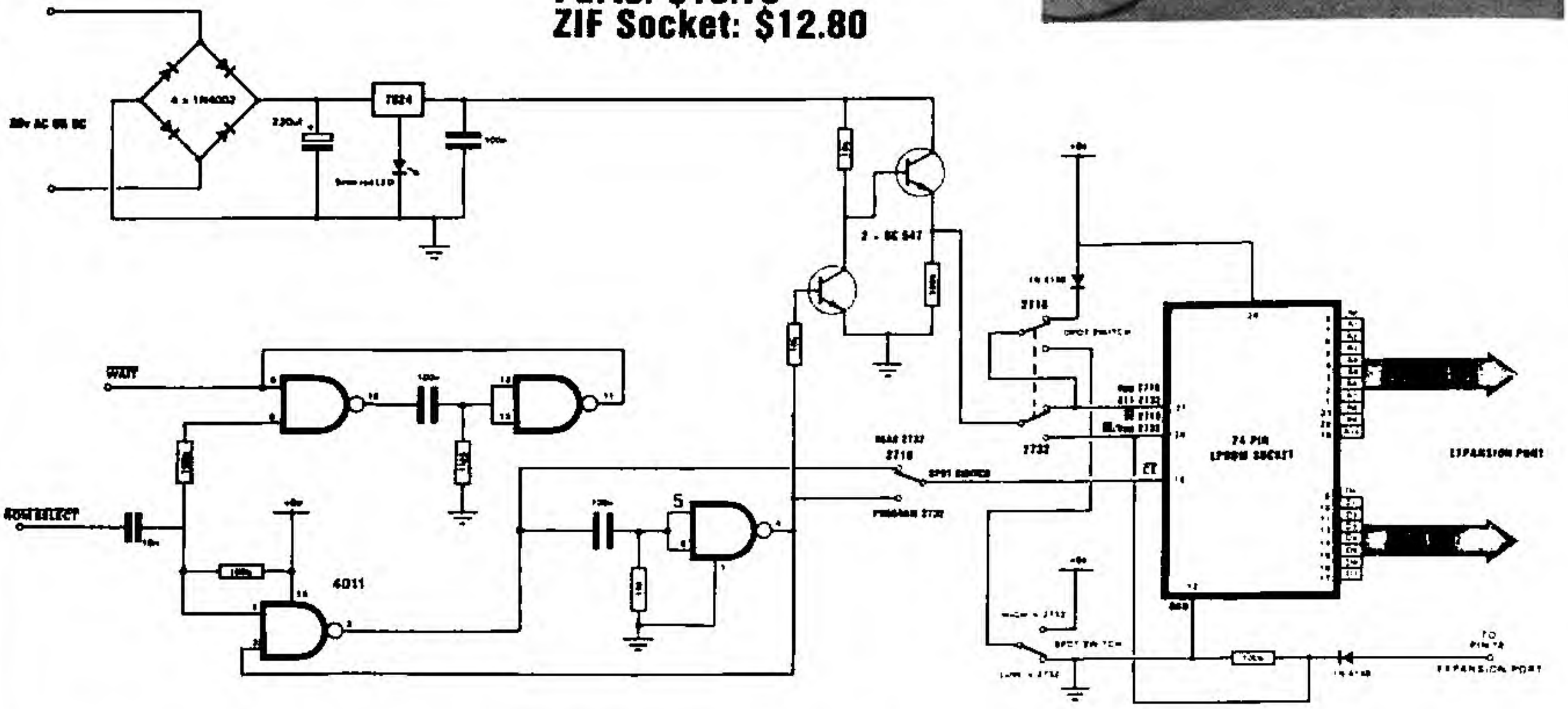
If the TEC starts to play up, it may be due to the RAM card drawing too much current. Feel the RAM chip. If it is getting hot, remove it immediately and let it cool down. The fault may be due to the way you inserted the CARD into the TEC.

Make sure you push the RESET button while inserting the card so that the buses are in a non-active state. If the TEC continues to play up when the card is re-fitted, replace the 4071 and/or the 6116 chip.

EPROM BURNER



PC Board: $3.50
Parts: $13.70
ZIF Socket: $12.80



An EPROM BURNER is the greatest thing to hit the TEC since the regulator was put under the board!

It adds the versatility you have wanted for ages.

To be able to save a program in a permanent form is the final goal of programming.

The TEC RAM CARD and EPROM BURNER combine to make a system capable of generating, testing and producing programs in hard form which can be saved, stored or sold - programs capable of emulating almost any task imaginable.

You can take any project from any magazine or book and convert it to a micro design with a consequent saving in parts, space and cost. Its capability can be increased and its reliability improved by using a tried-and-proven micro design.

By using the NON-VOLATILE RAM as the intermediate stage and the MON 2 monitor (with insert and delete functions) for the production of the program, you can generate, and have running, any machine code program, before burning it permanently into an EPROM.

Burning an EPROM is the final stage and you should be thoroughly satisfied with the performance of a program BEFOREHAND as it cannot be changed once it is burnt.

This is not entirely true as you can change some values and 'burn-down' any value to zero.

This is an important fact to remember when programming and we will explain what we mean:

EPROMs are purchased in blank form and this means the cells (of which there are 16,384 in a 2716 and 32,768 in a 2732) do not hold any charges of electricity.

Due to buffering circuits in the EPROM, the output from a blank device will be a set of HIGHs. Advantage is made of this as you will see. Eight cells are accessed at a time and if the value is read, it will be 1, 1, 1, 1, 1, 1, 1, 1. But we don't want to read a blank ROM - we want to program it with useful commands and data.

In Hexadecimal notation, the blank EPROM produces FF's from each set of 8 locations. This is called a byte and as we burn each byte in the EPROM burner, we convert the FF's into a lower value. If we don't burn a particular location, its value remains FF.

If we burn all 8 cells, the resulting value will be 00 and the designers of microprocessors (such as the Z-80) have given a very clever command to this value. It is "NO-OPERATION" in which the processor glides over the location without affecting any of the remaining program.

PARTS LIST

2	-	10k
3	-	100k
1	-	1M
1	-	1M5
1	-	10n greencap
3	-	100n
1	-	220uf 35v electro
2	-	1N 4148 signal diodes
4	-	1N 4002 power diodes
1	-	red LED
2	-	BC 547 transistors
1	-	4011 IC
1	-	7824 regulator
1	-	14 pin IC socket
1	-	24 pin wire-wrap socket
1	-	24 pin DIP header
1	-	24 pin ZIF socket **12.80 EXTRA**
2	-	10cm hook-up flex
2	-	matrix pins (for TEC)
2	-	matrix pin connectors
4cm		heat-shrink tubing
1	-	6BA nut and bolt
3	-	DPDT slide switches
1	**-**	**EPROM BURNER PC BOARD**

The advantage of having a No-OPeration command as **00** means any location can be 'burnt-down' to 00 if it is raquired to be removed.

This is where the term 'burn-down' comes from. Whenever en EPROM is burnt or programmed, the value produced is less than the starting value for the location.

We said values cannot be changed once burnt, but in some cases you can reduce the value if the following conditions are met;

The main criteria is:**the cells you wish to change must be 1's.**

The table balow shows the values which can be burnt down and those which cannot.

BURN-DOWN TABLE:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ANY VALUE	C A 8 6 4 2 0	C 9 8 5 4 1 0	8 4 0	A 9 8 3 2 1 0	8 2 0	8 1 0	0	6 5 4 3 2 1 0	4 2 0	4 1 0	0	2 1 0	0	0	0

The bold value can be burnt down to the values shown in the column.

Values cannot be 'burnt-up' as wa cannot produce HIGHs in an EPROM BURNER.

The only way we can restore HIGHs or 1's to the cells of an EPROM is to put it under an ultra violet light source, whereby ALL the locations will be arased and converted to 1's.

THE CONCEPT

Locations in an EPROM can be burnt if a voltage of 25v is applied to the Vpp pin and pin CE pulsed for 50 milliseconds.

The cells which will be given a charga of electricity will depend on the address which is being accessed and the value of data present on tha data lines.

These ara the only requirements and programming can be done with a simple set of switchas. Unfartunataly this would take an enormous length of time as 11 switches would be required for the address lines 8 switches for the data lines and each would have to be set for each byte of information.

A 2716 contains 2048 bytes and if a byte is burnt incorrectly, the whole procedure would have to be repeated.

The other inconvenience is all the bytes in the program would have to be converted to binary so that they can be loaded via the switches.

All this would take so long that the operation of burning would become a head-ache.

By using the TEC, the address values increment automatically as the program advances and the values of date are automatically converted to binary when each location is being burnt. This means you can program in Hex.

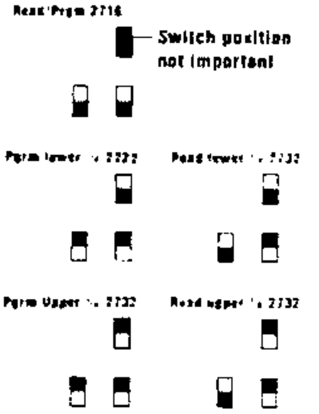
In this way an hours' work is converted to only a few minutes.

This is the function of an EPROM BURNER. It connects an EPROM to the address and data buses, provides the necessary 50 millisecond programming pulse and the 25v supply.

To hold the data steady on the data bus for the duration of the burn, it is necessary to HALT the computer. This is achieved by using another monostabla connected to the WAIT line, with a pulse length which is slightly longer than 50 milliseconds.

When you think of it, 50 milliseconds is 20Hz and when you include a short additional delay for the wait function and a number of machine cycles for the execution of a program to carry out the burn operation, you arriva at a burn rate of about 15 locations per second.

Divide this value into the numbar of locations you wish to burn and you arrive at the length of time for burning an EPROM. That's why it may take a minute or so.



Switch positions for programming and reading 2716's and 2732's

HOW THE CIRCUIT WORKS

The circuit is very simple and consists of a number of building blocks which come together vie the ROM SELECT line from the computer.

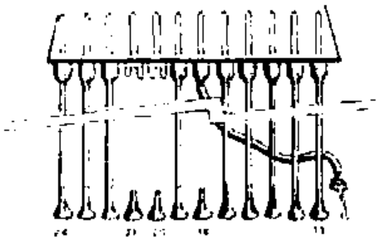
Sterting at the top of the diagram, the 25v is derived from a 7824 voltage regulator which has been 'jecked up' by 1.7v by the inclusion of a red LED in the COMMON line. This gives and output of 25.7v and by the time it reaches the EPROM, a voltage drop of .5v has occured across the switching trensistor. This trensistor is switchad via the output line of the 50 millisacond monostable.

The 25v line need not be switched ON and OFF when programming but must not be present when the EPROM is to be removed from the socket. 8y switching the voltage as we have done, the EPROM can be removed without damage.

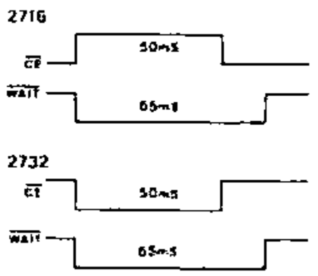
In this article we have included only a very simple burning routine which you can load into **0800** and get the project working.

The final two circuit blocks are monostables or one-shots, craeted from NAND gates. The lower monostable produces a 50 millisecond delay end the upper e 65 millisecond delay.

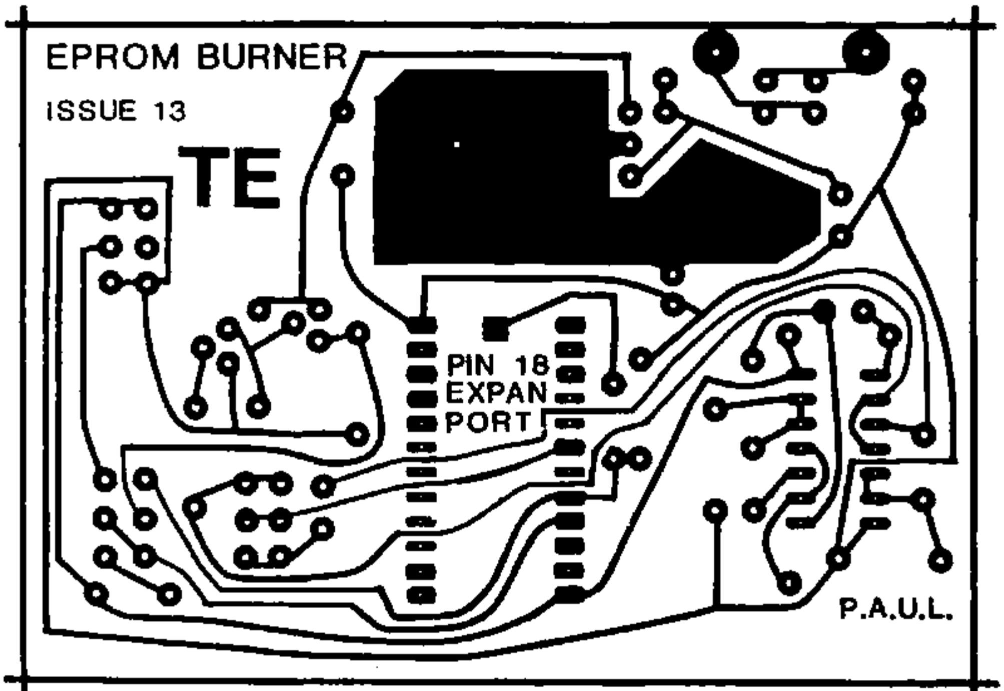
It places deta on the data lines, turns on the required address lines and turns on a Chip Select line (located near the edge of the TEC PC).



Don't forget to add the jumper lead from pin 18 of the wire-wrap to the PC board and cut pins 18, 20 and 21.



The programming pulses differ between the 2716 and 2732.



The timing diagrams on the previous page start at the commencement of the program (see page 48 for the program). The program runs for the first 4 lines and in the fifth line the instruction is to load the contents of the accumulator into the location pointed to by the DE register pair. This instruction does three things: not necessarily in this order:

It places data on the data lines, turns on the required address lines and turns on a Chip Select line (located near the edge of the PC).

This line accesses either a RAM or ROM chip connected to it and the address starts at 1800.

This action triggers both monostables and the WAIT monostable immediately goes LOW to HALT the computer.

The program is stopped and the EPROM BURNER circuit takes advantage of the data appearing on the address and data lines. This address is loaded into the EPROM and the data placed on the cells at this particular location.

The high voltage is turned on for 50 milliseconds and at the same time CE is pulsed. This permanently puts the value of data in the EPROM.

The 50 millisecond monostable ends it timing cycle and 15 milliseconds later the WAIT monostable goes HIGH. This enables the computer to continue through the program and come to a JUMP RELATIVE instruction to bring it back to line 5.

This time DE will be pointing to the next higher location and A will contain a new value of data. The count-register-pair BC will be one less than previously. The program continues to loop until BC pair is ZERO.

At the conclusion of the burn routine the program jumps to address 0000 and the monitor program is executed to bring 0800 on the screen with the reset beep to indicate the end of burn.

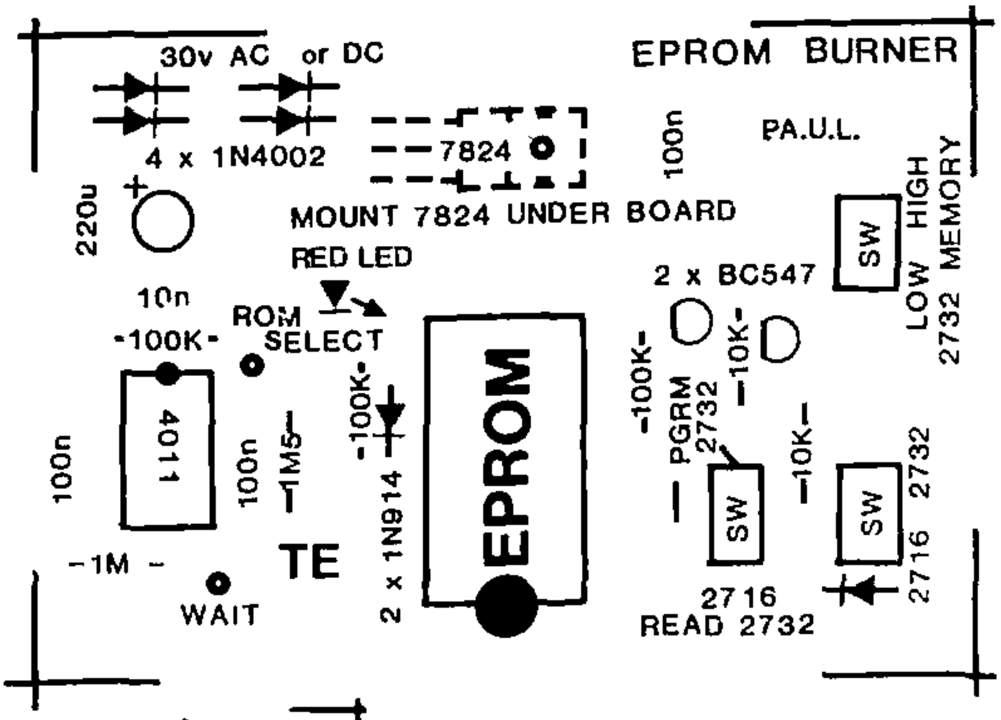
RECAP:

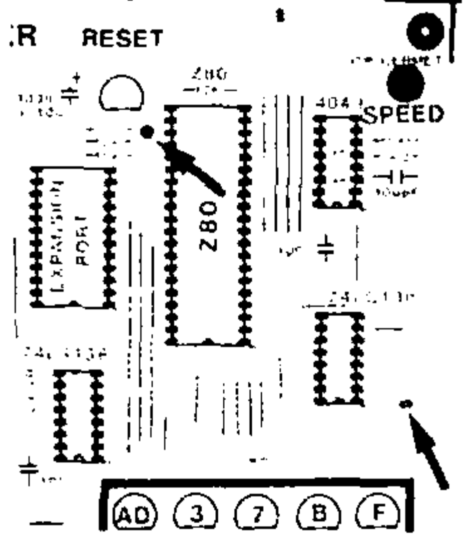
The program burns the EPROM at address **1800 - 1FFF**. To look at the data in EPROM: address **1000 - 17FF**. When the EPROM is removed from the programmer, its address will depend on the project you are using it in, but more than likely it will be the only programmed chip and thus it will be **0000 - 07FF** for a 2716 and **0000 - 0FFF** for a 2732.

CONSTRUCTION

Begin with the resistors and signals diodes. Keep them close to the board before and after soldering but be careful not to damage them with heat. Four power diodes are needed for the 30v bridge. Fit them next, along with the 10n and 100n greencaps.

There is one jumper link on the PC board and this can be made from a lead cut from one of the components.





The Chip Enable pin and Wait pin on the TEC.

The 7824 voltage regulator is mounted under the board and fixed to it with a nut and bolt. The pins fit through the holes provided and are trimmed on the topside of the board.

The two transistors and 14 pin IC socket are the next to the be added and then the red LED.

The 220ul filter electrolytic must be mounted around the correct way and the board is ready for the hardware parts.

Push the 24 pin wire-wrap socket through the holes in the PC and carefully solder it in position. **Cut off pins 18, 20 and 21.**

A short jumper goes from pin 18 of the DIP header to a solder-land on the PC board (located between pins 12 and 13). The easiest way to add this jumper is to solder one end to pin 18 of the DIP plug and the other end to the PC board BEFORE soldering the DIP plug to the wire-wrap socket.

Keep the pins of the wire-wrap full length and solder the 24 pin DIP header into position.

The three slide switches are mounted through the board if the holes are large enough. But if only small holes have been drilled, the legs will need to be extended with short lengths of tinned copper wire and the switches mounted above the board.

Finally connect two jumper leads for the WAIT and ROM SELECT lines and a length of twin flex for the 30v line.

The jumpers require matrix connectors and a short piece of heat-shrink tubing over them to make them sturdy. Heat the tubing with a flame to make it shrink over the connector.

The twin flex requires a 2 pin DIN plug so that it will fit the 30v socket on the TEC POWER SUPPLY.

Fit the 4011 IC and the board is ready.

Two matrix pins will be required on the TEC PC board to take the jumpers. These are soldered as shown in the diagram opposite and are included in the kit.

If you intend to produce a number of EPROMs you will need a ZIF socket. These are expensive (too expensive), but are essential if you want to avoid the damage caused by constantly inserting and removing EPROMs from an IC socket. The wire-wrap socket will accept about 50 insertions and removals before it gets a little weak. If the pins do not make good contact, the wrong values will be burnt.

When fitting a ZIF socket, push it firmly into the wire-wrap by starting at one end and gradually introducing the pins, two at a time. You cannot force all the pins together at once.

Pin 1 of the EPROM is towards you and this means the lever of the ZIF socket is also towards you.

Cont. P. 48. . .

EPROM BURNER

. . .cont. from P. 22.

The lowar switchas must be set for 2716 or 2732.

When buming 2716's the upper switch position does not matter as it is not in circuit.

When burning 2732's the upper slide switch selects tha UPPER 2k or LOWER 2k of tha 2732.

The switch closast to the EPROM is placad in the upper position when programming 2732's and in tha lowar position to read tham.
This switch is placed in the lower position when programming and reading 2716's. Rafer to tha sat of diagrams bafore carrying out any operation.

The high voltaga is derived from a 30v supply. This can be the TEC POWER SUPPLY or from a 30v AC transformer. Very little currant is required, however the voltage must not go below 30v or the regulator will dop out. This is because we are genarating 25.7v and the regulator raquires 3-4v across it for ragulation.

Connection of tha EPROM BURNER board to the TEC is via a 24 pin wire-wrap and DIP header plug. The board fits in the axpansion socket and requiras a WAIT line and ROM SELECT line.

The ROM SELECT line is pin 12 of the 74LS138 and WAIT is pin 24 of the Z-80.

Connect these lines to the TEC and plug the EPROM BURNER board into the axpansion socket. Connact the 30v supply and the red LED will illuminate to indicate all is ready.

You can bum a naw EPROM or blank locations in an old ROM. You can even burn old locations providing they fulfill the requirements mentioned previously.

THE PROGRAM

The BURN PROGRAM can be placed anywhere in the Monitor ROM or typad into tha RAM. If placed in the RAM, it will need to be typed each time an EPROM is to be burnt.

The program is very simple and does not have any checking facility to prevent it burning over previous program.

The absence of this means you can burn or reburn any location(s) anywhare without heving to break a safety lock.

The first threa linas of tha program contain varieblas which have to be set each time you want to burn an EPROM. For this raason, tha three lines must be typed into RAM with a fourth lina to provide a call or jump to the remainder of the program. Tha rest of the program can ba located in ROM (at say 0700).

Hera's how it is done:

You will need a blank 2716.

The first stage is to trensfer the MONitor program into tha new EPROM. Load the following into 0800:

```
LD DE 1800      800   11 00 18
LD HL 0000      803   21 00 00
LD BC 06FF      806   01 FF 06
LD A(HL)        809   7E
LD (DE),A       80A   12
PUSH BC         80B   C5
DJNZ FE         80C   10 FE
DJNZ FE         80E   10 FE
POP BC          810   C1
INC HL          811   23
INC DE          812   13
DEC BC          813   0B
LD A,B          814   78
OR C            815   B1
JR NZ           816   20 F1
Restart 0000    818   C7
```

Make sure the switch selects 2716. Push RESET. GO. The TEC screen will blank for about 2 minutes while the program is burning.

When the screen reappears you can check the operation by addressing 1000 and read tha locations. Compare tham with 0000 and confirm the program has been transferred.

The next stage is to add the burn routine to the MONitor ROM. This is done at 0700. Change the values at 0800 to:

```
11 00 1F
21 09 08
01 10 00
```

Push RESET GO and the program will be transferred in a few seconds.

You have now produced a MONitor ROM with a burn routine at 0700. Place the new ROM into the TEC and it will start up with 0800.

To use the BURN ROUTINE, type the following et 0800:

```
11 ____ ____ TO: ROM address + 1800H
21 ____ ____  FROM: RAM address
01 ____ ____  No of hex bytes
C3 00 07
```

Programs to be burnt into EPROM are placed at **0900** and can extend to **0FF0.** To transfer these programs to EPROM. place the following et **0800:**

```
11 00 18
21 00 09
01 F0 0E
C3 00 07          Push RESET GO.
```

EX: 80 byte program at 0900 to 0000 in EPROM:
at 0800:

```
11 00 18
21 00 09
01 80 00
C3 00 07          Push: RESET, GO.
```

A6 byte program at 0A00 to 0180 in EPROM:
at 0B00:

```
11 80 19
21 00 0A
01 A6 00
C3 00 07          Push: RESET, GO.
```

40 byte program at 0C00 to 02C0 in EPROM:
at 0800:

```
11 C0 0A
21 00 0C
01 40 00
C3 00 07          Push: RESET, GO.
```

If you type a program at **0800**, the BURN ROUTINE can ba located at **0900:**

```
11 xx xx
21 00 08
01 xx xx
C3 00 07
```

decrement to **0900** Push: GO, GO.

Before starting any programming you should fill the TEC RAM with FF's. This will allow any non-program locations to be trensferrad and retain the value FF.

To fill RAM with FF's:

```
11 FF FF
D5
C3 03 08
Reset, GO.
```

Programs can ba transferred from EPROM to the TEC memory via the following routine:
at 0C00:

```
11 00 08
21 00 10
01 ____ ____ (No of bytes)
ED BO
C7
```

decremant to **0C00 Push: GO, GO.**

Program will transfer very quickly · This is not a burn routine but a DUMP ROUTINE which can also be used for the non-volatile RAM project.

Example: B0 bytes in EPROM at 0000 to 0900 in TEC RAM.

at 0800:

```
11 00 09
21 00 10
01 80 00
ED B0
C7          Push Reset, Go.
```

(0000 in EPROM means page-zero in EPROM).

Ex: **BO bytes in EPROM at 0630 to 0900 in TEC RAM.**

at 0B00:

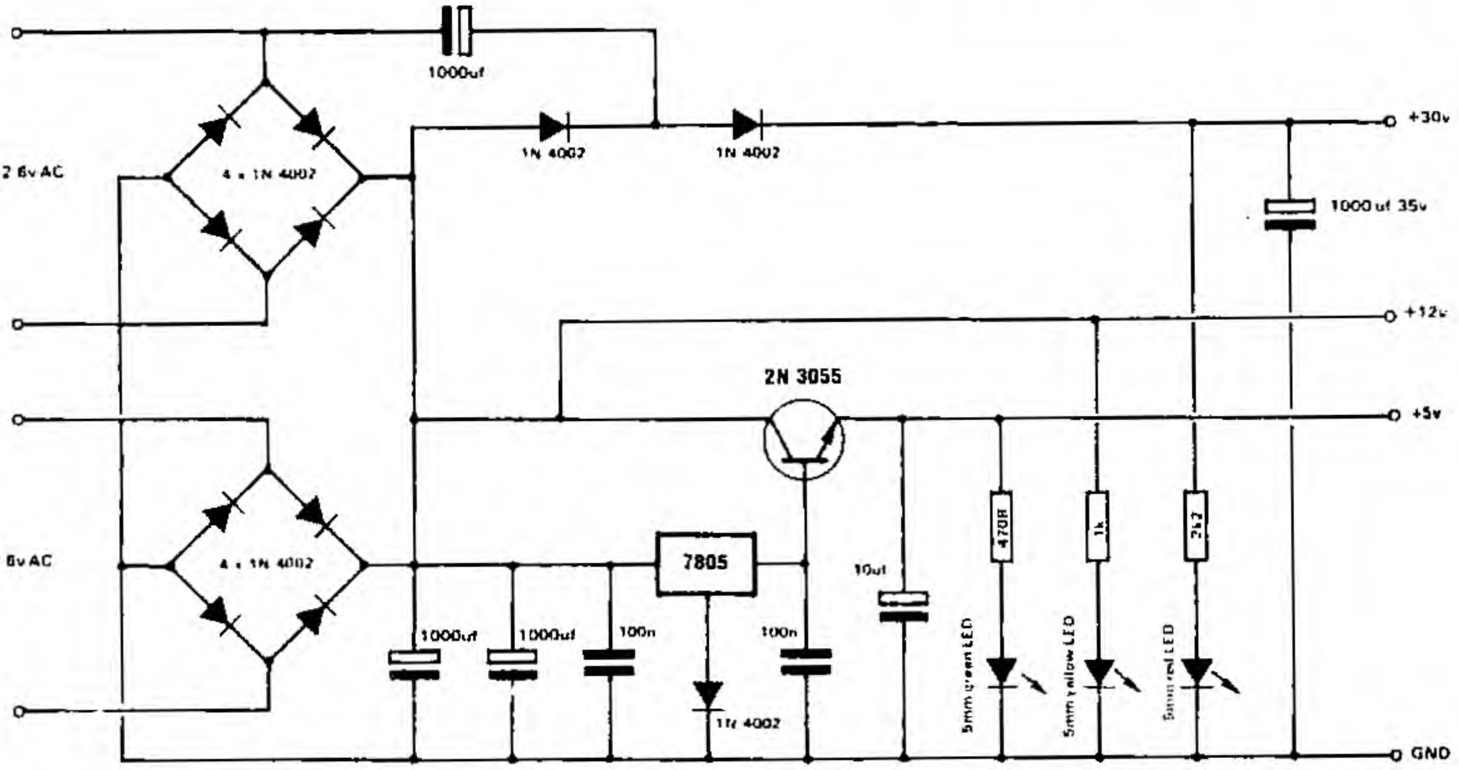
```
11 00 09
21 30 16
01 B0 00
ED BO
C7       Push: Reset, Go.
```

Before attempting any transfer, you must write tha necessary program on e piaca of peper using one of the examples in the text. Check it carefully then typa it into the TEC at 0800 (or othar location as explained).

Using tha programmar and the non-volatile RAM in conjunction with tha TEC will open up lots of possibilitias. Programs can be used on the TEC or MICROCOMP and you will begin to sea how everything is going togather.

TEC POWER SUPPLY

PC: $5.50
Parts: $13.90
Complete Kit incl. Transformer, PC & case: $44.25



With the gradual expansion of the TEC, we have come to the stage where we have run out of voltage and current from a plug pack.

A 12 volt 500mA plug pack may sound ideal in theory but when you connect it to a project requiring about 300mA, a cruel thing happens. The output voltage falls from 12v to 10v!

This may be ok if you are dropping it down to 5v via a regulator, but when you want the full 12v for say a voltage doubling operation, 10 volts is not enough!

For too long we have been gulled into believing the ratings of transformers and plug packs. It's only when we require the full rated output that we realize it will not produce.

We learnt our first lesson with the 2155 in a power supply some years ago. Its stated output is 15v AC at 1 amp and this really means 1 amp AC. It also has an AC rating of 15VA and this is very similar to saying 15 watts.

But as soon as we place a 2155 in a power supply we convert the AC to DC via a bridge and gladly accept the output rise to about 21v DC, which is about 40% higher than the AC voltage.

Since the volt-amp rating of the transformer is a CONSTANT (a **constant** is a value in a formula which does not alter) and is 15VA, we must derate the output current to 700mA to maintain the rating of 15VA (or 15 watts).

Thus we can safely draw only about 700mA from a 2155.

There are further projects being designed for the TEC and they include a VDU, possibly for the next issue. The VDU board takes about 350mA, making a total very near the maximum for a 2155 and above the capability of a 500mA plug pack.

We also have a Relay Driver board requiring 12v-15v for the relays and an EPROM BURNER, in this issue, requiring 30v.

All this has led us to design a power supply capable of delivering these 3 voltages. At a later date it can be expanded to deliver about 1.4 amps to the 5v line, to cater for fully expanded TEC's.

The TEC Power Supply is not only for the TEC, but will also power any other project requiring one or more of these voltage.

The project is mounted in a neat plastic case as supplied by Altronics and Dick Smith and is the LARGER of the two (in the range).

These cases make the project look very professional and can accommodate both 2155's and the PC board. The floor of the case has a number of spigots for mounting the board and transformer(s) so that everything fits firmly and neatly in position.

As with all projects which involve connection to the mains, this project must not be connected until it is checked by someona with experience.

The first quastion you will ask is "Why use two 2155's and two bridges."

The answer is simpla. Working with currents up to 1 amp produca very faw problems. Componants such as diodes, regulators and transformers are designed for a maximum of 1 amp. When you go over 1 amp, tha problams start.

Nothing is dasigned for 1.5 amp or 2 amp and in fact high powar componants start at 3 amp.

This means a 1.5 amp or 2 amp power supply falls into the middle of component availability.

The cheapest and best solution is to produce two 1 amp supplies and parallel them up. This is what we hava done. Two 2155 transformers are taken to two bridges and from there the current gets divided between the thrae outputs. Most of the current will ba raquired by the 5v line while the 15v and 30v lines will not require heavy currents.

In practice, only about 30-50mA will be required on the 30v line for the EPROM BURNER and only about 100mA for the Relay Driver board.

HOW THE CIRCUIT WORKS

The circuit is basically a 7805 ragulatad power supply with a sarias pass 2N 3055 transistor to supply the current for tha 5v rail.

The 15v rail is taken from the input to the regulator and is filtered and smoothad DC, but not regulated.

Refarring to the circuit diagram, the 30v rail is obtained by voltage doubling the 15v line. Tha top bridge is responsible for this and means it is the bridge which operates when only ONE transformer is used. The lower bridge providas the back-up when more than about 700mA total is required.

The 5v line has some interesting points.

The output transistor is an emitter follower in which the base voltage is determined by the output voltage of the 7805 regulator. As we know, the amitter of a transistor in an emitter follower arrangement is .6v lower than the base. Thus, to obtain a 5v output, wa must supply the base with 5.6v. This is achieved by placing a diode in the 'common' line of the 7805 and increasas the output by .6v. Thus it emerges from the 2N 3055 at exactly 5v.

In this arrangement we have lost the shut-down facility of the 7805. But since we have found this to be very unreliable with a 2155 transformer, nothing has really been lost.

PARTS LIST

1	-	470R ¼watt
1	-	1k
1	-	2k2
2	-	100n greencap
1	-	10mfd electro
3	-	1000mfd 16v or 25v
1	-	1000mfd 35v or 63v
11	-	1N 4002 diodes
1	-	5mm red LED
1	-	5mm green LED
1	-	5mm yellow LED
1	-	7805 regulator
1	-	2N 3055 transistor
1	-	T0-3 heatsink
2	-	4BA nuts and bolts
3	-	6BA nuts and bolts
10 x		20cm hook-up flex
1	-	2 pin DIN socket
1	-	3.5mm socket
1	-	RCA socket
1	-	2155 transformer
4cm		heat-shrink tubing
1	-	cord clamp
1	-	power cord and plug-top
1	-	solder tag for earth lead
1	-	H 0482 case
1	-	SPDT switch for mains
		haatsink compound
1	-	**TEC POWER SUPPLY PC**

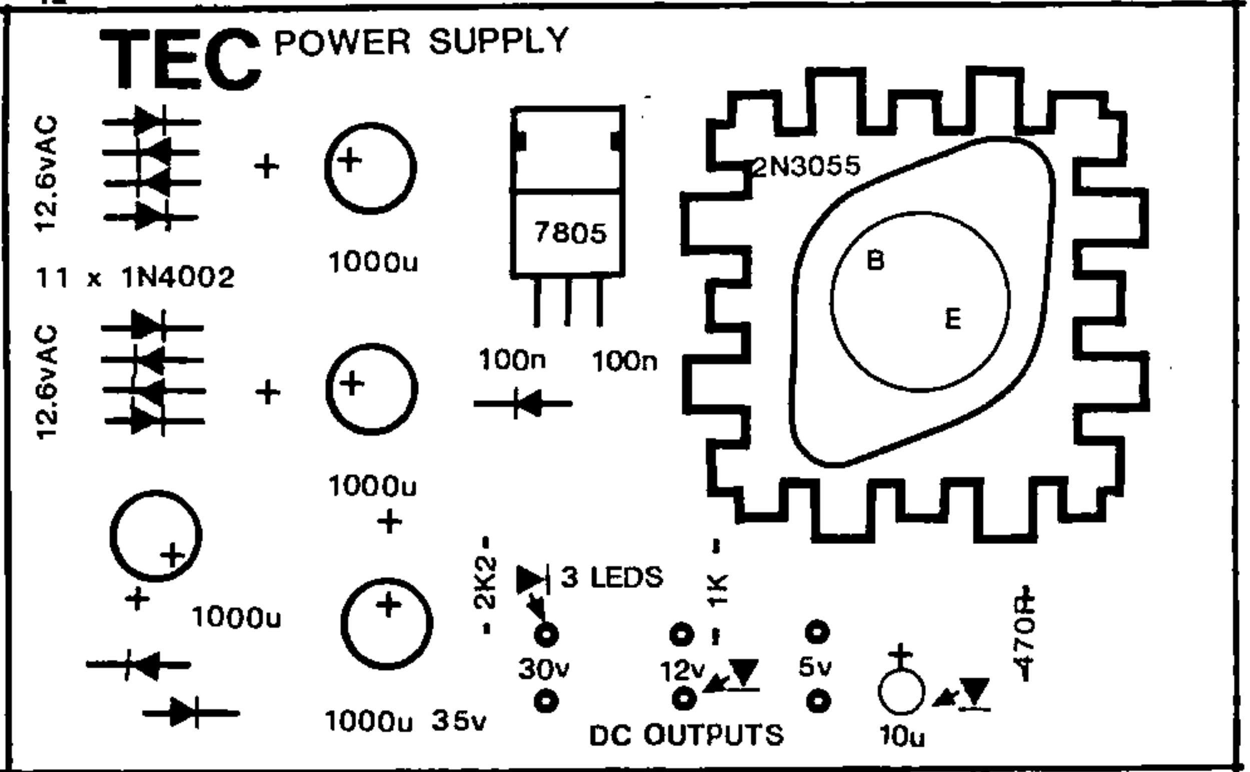
Extra parts: 1 - 2 pin DIN plug, 1 - 3.5mm plug & 1 - RCA plug: $1.30

The 7805 requires about 1.5 amps to be flowing before it will shut down and this current is not available from a 2155.

When looking at the photo of the completed project you will notice the regulator does not have a heat fin. This is because the regulator does not supply the current to the 5v line. The transistor doas all the work. The regulator supplies a voltage and about 50-80mA to the 2N 3055, to drive it. The transistor has a currant gain of about 10 to 20 and thus it is capable of supplying about 700-1400mA. This is why the 2N 3055 must be properly heat sinked.

CONSTRUCTION

All the comonents ara mountad on a single PC board with flying leads to each of the output jacks and the indicator LEOs. If using one transformer, a twisted pair goes from tha 12.6v AC holes on the middle of the edge of the PC to the 0v and 12.6v tapping on the transformer.



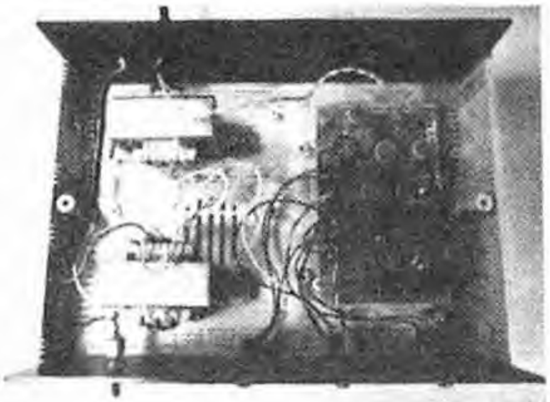
The overlay for the TEC POWER SUPPLY. The 7805 voltage regulator does not require a heatsink but the 2N 3055 must be suitably heatsinked as it is the current regulating component.

Start by fitting the 11 power diodes. The cathode end is identified via a white band around one end and this corresponds to the line on the symbol on the overlay.

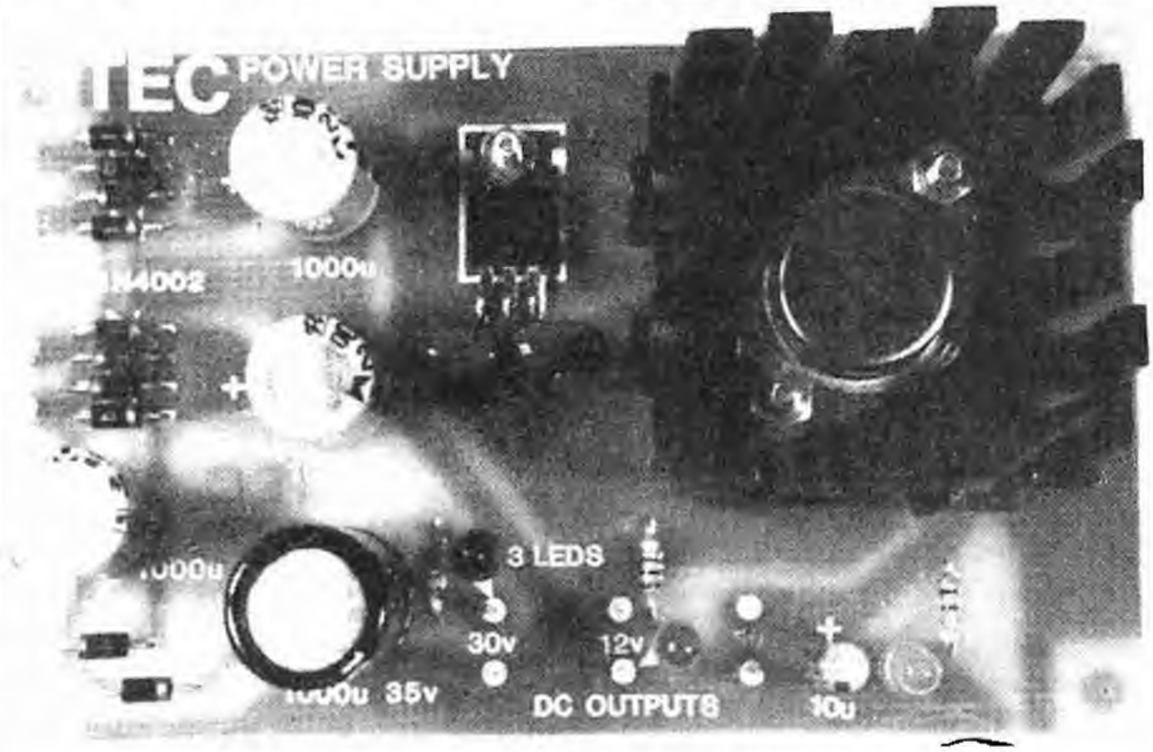
The diodes must be pushed home BEFORE soldering and must touch the board AFTER soldering. This is necessary as the copper tracks are designed to act as a heatsink to prevent the diodes getting too hot.

Next fit the three resistors. These are current-limiting resistors for the indicator LEDs.

Fit the two 100n greencaps and the 4 electrolytics. Note the marking on the electro's indicates the negative lead whereas the board identifies the positive lead. Don't get mixed up.



Layout of 2- 2155 transformers and PC board inside case.



Next push the leads of the 7805 through the hoes in the board and bend the regulator over. Attach it to the board with a nut and bolt and then solder the leads.

The final component to mount is the power transistor.

Place the heat-fin on the board and before the transistor is fitted into place, smear a little thermal compound on the underside of the transistor.

Take care not to short the base or emitter leads against the heatsink.

Place the transistor onto the heatsink and attach to the board with two nuts and bolts. As you tighten the bolts, you will notice the compound squeezes from around the edges of the transistor. If you have done this correctly, it will be even all round. Solder the base and emitter leads. The collector is the case of the transistor and gets its voltage via the bolts. That's why they must be screwed up tightly and don't get the thermal compound on the bolts.

2 pin DIN for 30v

3.5mm for 15v

RCA for 5v.

The next stage is to attach the flying leads to the board for the input and output.

It is suggested that a colour code is used so that each output can be recognised by a colour. This will prevent a major mistake.

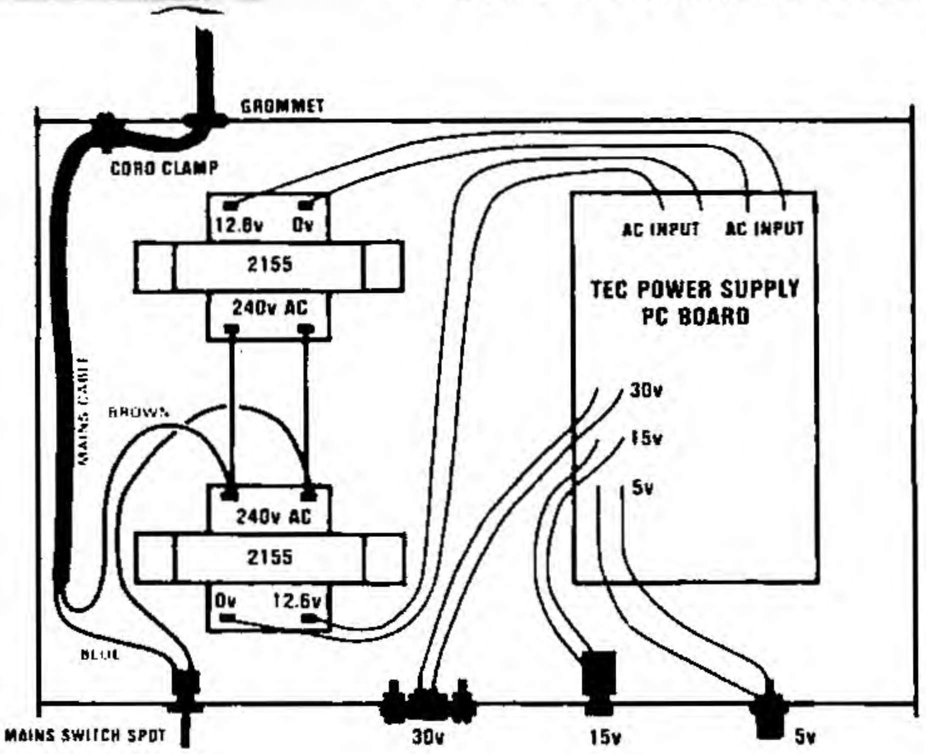
The 3 output sockets are different to each other so that the plugs must also be different. Each project you connect to the supply must be fitted with the correct type of plug and this will prevent the wrong voltage being selected.

The sockets we have chosen are RCA for the 5v line, 3.5mm for the 15v and 2 pin DIN for the 30v. The RCA socket is used for the 5v because it provides the greatest amount of contact between plug and socket for the higher current flow.

These sockets are mounted on the front panel along with the indicator LEDs.

The exact position for these sockets is not critical except you have to make allowance for the heat fin and transformer. This restricts the layout somewhat and the photo shows a suitable positioning.

The only other component on the front panel is a 240v power switch.



Layout for 2 transformers. If only one transformer is required, the smaller case can be used.

WIRING THE MAINS SECTION

Wiring the mains lead to the power supply is simple enough except that it must be done according to RULE. This states that the lead must be attached or 'gripped' to the cabinet so that it will not pull out or pull on the wires inside the box.

Make a hole in the rear panel which is just large enough and push 20cm of the lead through. Place a cord anchor about 2cm from this hole and fasten the lead to the cord grip with a nut and bolt.

Remove 9cm of the outside sheath and separate the three inner wires. The ACTIVE lead is RED or BROWN and must be taken to the switch on the front panel. Bare ½cm of the lead and slip a 2cm length of heat-shrink tubing over this lead. Solder to either terminal of the switch.

Cut the **BLUE** or **BLACK** neutral wire slightly shorter and repeat the same procedure, this time connecting to either of the primary terminals on the transformer. Don't forget the short length of heat-shrink.

Finally the earth lead must be connected. Surprisingly, this lead must be the longest so that it is the last lead to be broken, should the cord be pulled. Solder this to a solder tag and screw it onto one leg of the transformer.

A short jumper is required between the switch and the other input of the transformer. Add this and include the heat-shrink.

The heat-shrink must now be pushed over the exposed parts of each of the connections and shrunk into place by heating with a match or candle. The object of the exercise is to cover all the mains wiring so that nothing can touch a live wire.

If you have bought a MOULDED PLUG, the power cord will now be complete. If the plug-top requires attaching, use the colour code given above for the correct connection of the wires. You may think the choice of colours is rather poor. Don't blame me. It's an international code, to assist colour-blind people.

TESTING THE POWER SUPPLY

Testing the power supply is done in two stages.

Connect the plug to the mains and turn the switch on the case ON and OFF very quickly. The three indicator LEDs should come on. If they do, you can be sure the three voltages are present.

Next you must test the supply to see if any of the components are going to overheat. This is always an important stage in testing a project which has the potential for supplying a lot of power.

Turn the supply ON for 10 seconds, then pull the plug from the wall. Feel the temperature of each component. If everything is cool, repeat for 30 seconds. Again remove the plug and feel each item.

If everything is ok, switch on for 5 minutes and try again.

Parts can be damaged very quickly when in a high current situation like this and if an electrolytic is around the wrong way, it will very quickly heat up and may even explode.

The next stage is to measure the output voltages with a multimeter. The most important voltage is the 5v rail. It must be exactly 5v. TTL projects such as the TEC computer will be connected to this rail and it must be spot on 5v.

The other rails are not quite so critical as they are used to operate relays; and the 30v rail will be taken to a 24v regulator for feeding into a 25v line as shown in the EPROM BURNER project.

Even so, you should make sure they are delivering the required voltage. If the 30v rail, for instance, is only 15v or 20v, something is wrong. And if it is 25v, it will not be sufficient for the EPROM project.

A low voltage, like this, may be due to low mains or a diode missing (not working) in the bridge.

One way to test the diodes is to fully load the supply, via say the 15v rail, and feel the temperature rise of each diode. They should all get equally warm.

If your area suffers from low mains voltage, you can increase the voltage of the supply by connecting between the 0v and 15v tapping on the transformer.

FINAL ASSEMBLY

After testing the power supply for temperature rise, fit the front and back panels and close the case using the two long screws provided. Use white lettra-set or other form of identification to show the three output voltages and the ON position for the mains switch.

POWERING THE TEC

The TEC can be powered from either the 15v line or the 5v line.

If powering from the 15v line, a lead can be taken from the AC terminals on the TEC to the 15v output. The 7805 regulator on the TEC will provide the regulation. It is interesting to note the input lines can be either way around as the TEC has a full wave rectifier and this means the input voltage can be of any polarity.

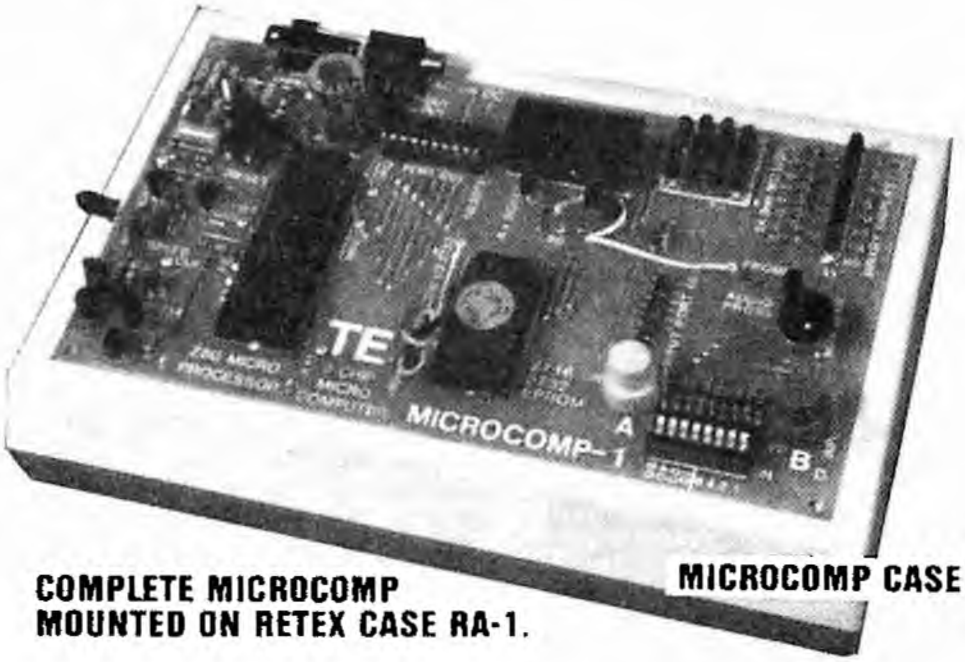
If supplying from the 5v output, things are different. You must connect to the 5v rail of the TEC. This is done by connecting one lead to the earth line and the other to the output pin of the 7805.

MICRO COMP

A 3-CHIP Z-80 COMPUTER

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

Parts: $47.25
Board: $8.50



MICROCOMP CASE $12.50

COMPLETE MICROCOMP
MOUNTED ON RETEX CASE RA-1.

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

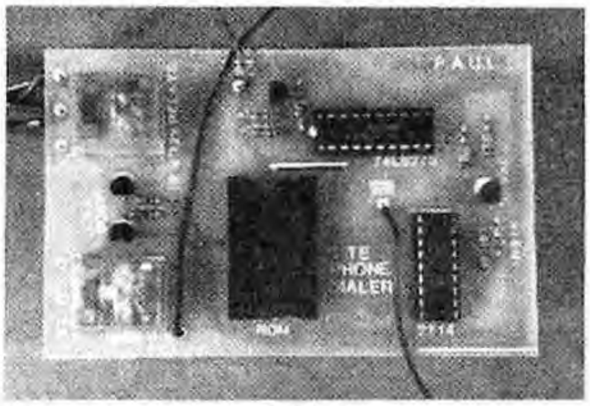
But if you are only starting in this field and want a low-cost introduction into micro-computer programming - THIS IS IT!

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

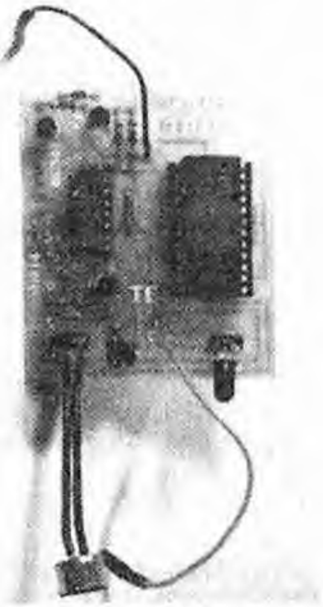
If you are not up to it, don't do it. But if you have made a lot of projects and want to graduate to the next level - this is how to go. You will be amazed with the capabilities of the unit and it will involve you in hundreds of hours of programming.

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

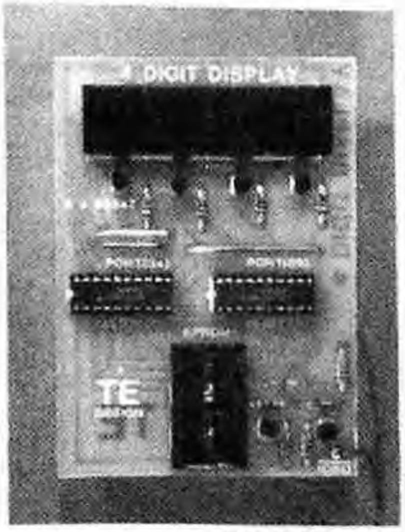
SOME OF OUR 'ADD-ONS':



PHONE DIALLER



MORSE TRAINER



4 DIGIT DISPLAY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Now this has all changed.

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

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

This means the project will be well within your budget.

Apart from the programs already mentioned, we are in the process of producing programs and modules for an Alcohol Breath Tester, a Digital Resistance Meter, Digital Capacitance Meter, a Bio-feedback Unit, a Mini Frequency Countor 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, ma ıy 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 leam about programming without experimenting with the real thing? So building the Microcomp is really an essential part of learning to program.

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

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

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

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

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

But most important Machine Coda 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 e 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 keaps 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 tuming 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 tha 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 tha 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 projact from working.

Digital electronics is extremaly 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 tha 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 transfarred from bulk tubes without being handlad. 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 releasad and at least three prototypes have been made before the project goes to print.

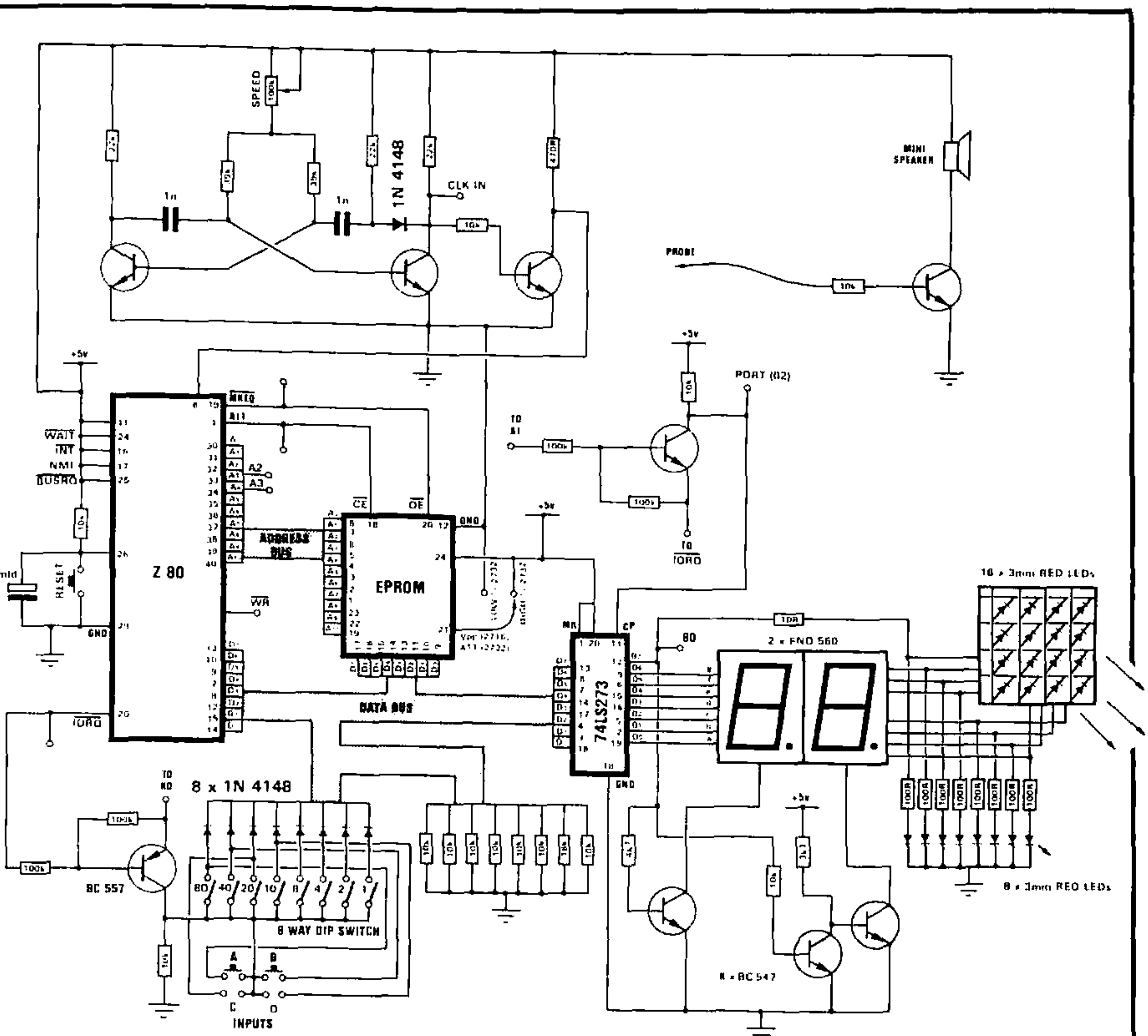
This contributes to the success of our kits and the neatness of the finished project is enhanced by the solder mask on the underside of the board. This prevents solder sticking to unwanted areas and shorting between tracks.

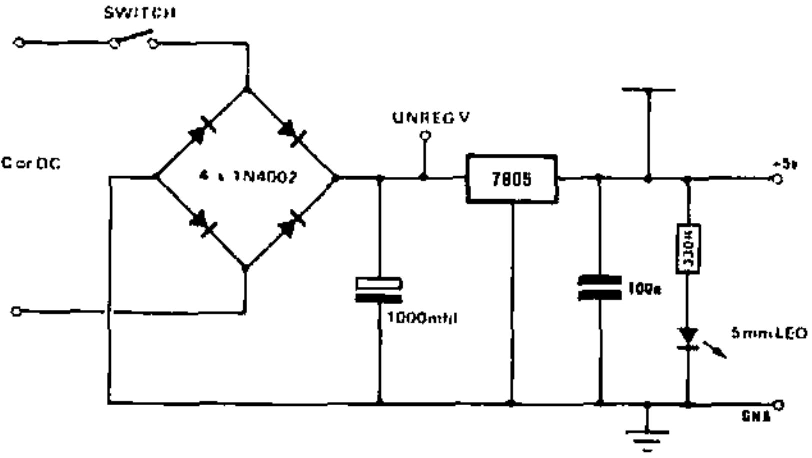
To be sure of success, buy a kit. A number of shops are selling these kits and you will find the cost is less than hunting for the individual bits yourself.

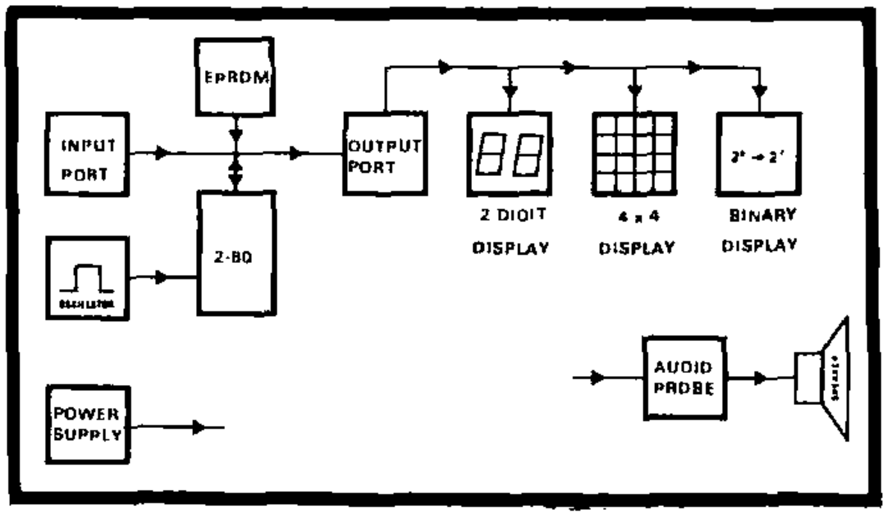
PARTS LIST

1 -	10R
8 -	100R
1 -	330R
1 -	470R
1 -	3k3
1 -	4k7
14 -	10k
3 -	22k
2 -	39k
4 -	100k
1 -	100k mini trim pot
2 -	1n green cap
1 -	100n
1 -	1mfd 63v electro
1 -	1,000mfd 25v electro
9 -	1N 4148
4 -	1N 4002
1 -	5mm red LED (SPEED)
1 -	5mm green LED
24 -	3mm red LEDs
8 -	BC 547 transistors
1 -	BC 557 transistor
2 -	FND 560
1 -	74LS273 IC
1 -	Z-80 CPU
1 -	2732 EPROM (PROGRAMMED)
1 -	7805 regulator
1 -	20 pin IC socket
1 -	24 pin IC socket
1 -	40 pin IC socket
1 -	8 way DIP switch
1 -	DPDT slide switch
3 -	PC mount push switches
1 -	3.5mm mono socket
1 -	mini speaker
1 -	6BA nut and bolt
4 -	rubber feet
13 -	matrix pins
1 -	hollow pin
20cm	hook-up flex
1	metre tinned copper wire
1 -	female matrix pin connector
2cm	heatshrink tubing.

MICROCOMP PC BOARD







BLOCK DIAGRAM OF MICROCOMP

CONSTRUCTION

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

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

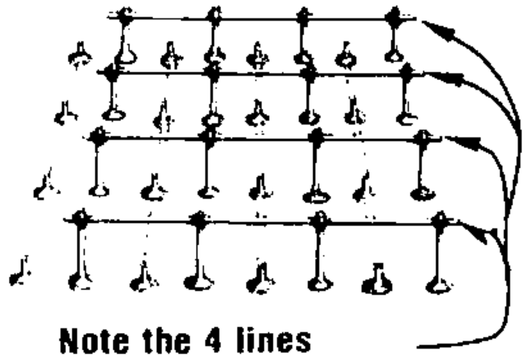
All the components are marked on the PC board and it is possible to build the project without any other help. But as a guide we will go through the assembly and explain everything as we go.

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 DN-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 'PDWER-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.

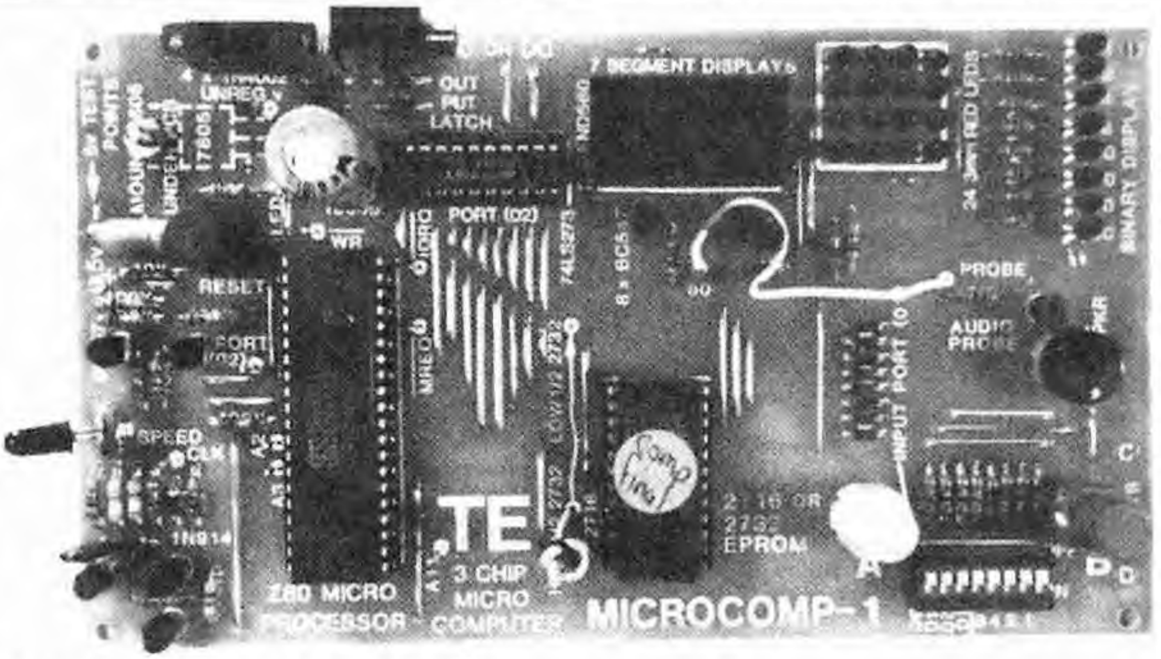
Fit 4 rubber feet to the underside of the board, insert the chips and you are ready for testing.

TESTING

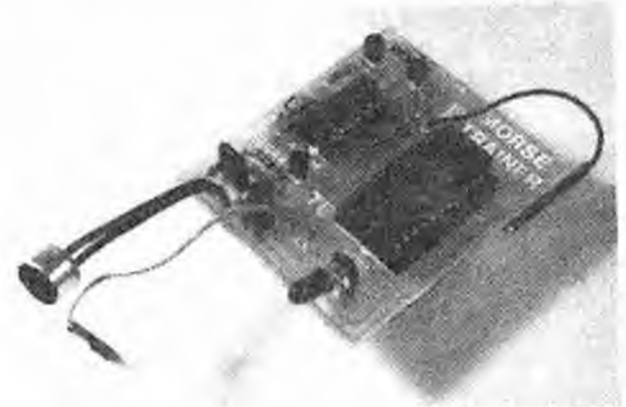
Insert the power plug into the 3.5mm socket and switch the Microcomp ON. The power LED should come on. Make sure all the input switches are OFF. Push button B. The number 99 should appear on the displays. Press button A and the numbers will increment. Push button B and they will decrement. This is a 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.

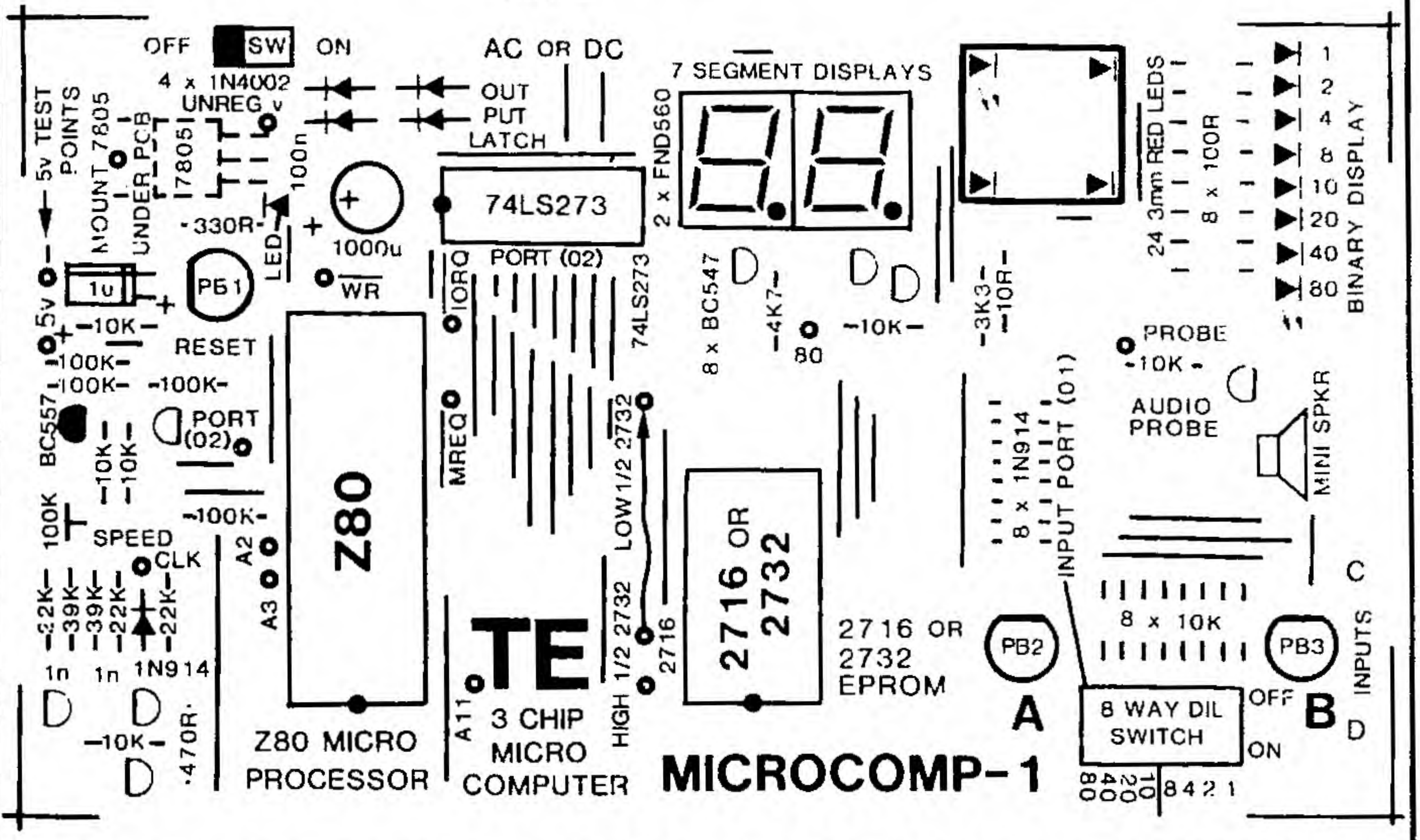
Consider yourself lucky that the computer doesn't work. You will gain a lot by trouble-shooting it yourself and gain experience in finding the fault.



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 hava 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 knifa. 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 spead, 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 multimeter 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 accapt 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 maka 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 - mada 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 informaticn. 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 placa. 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 tha program can be performed.

THE OUTPUT LATCH

The output latch is the third end final chip in the project. This is the chip which drivas 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 programmabla 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 sea 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 tha 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 ovar 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 tha 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 tha 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 tha 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 tha PC board which is bit 7 of the output latch. The Tone program at 0010 outputs a HIGH to this line and than 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 date 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 raad in the same order as it was programmed, the correct data will ba outputted.

The only reason for keeping to an accapted 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 raquired 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. 8y 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 raquires 2 digits. To input a value such as 234 refer to the Hex Conversion table on P 16 of issue 11. It is equivalant 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 mada 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 valua 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 anter 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:

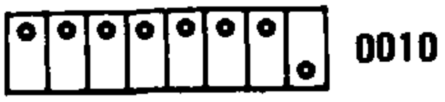
1. Set the switches to address values which are not the start addresses of a program. Why do some of them work?
2. Why does button B address the start of the 00 - 99 counter?
3. Could the DIP switches be replaced with push buttons?
4. Explain what we mean by the input switches are software programmed: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

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
```

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 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.

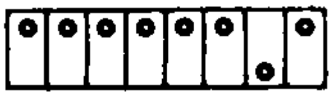
The program will loop continually until the reset button is pressed. The input switch must be OFF to prevent the program being accessed again.

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.

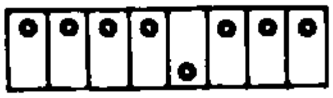
Press the reset button to start a new game.

Data Bytes at 00F5:

```
01
02
04
08
88
90
A0
81
```

RUNNING NAMES

To access this program, switch B 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

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

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

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

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

TABLE OF NAMES:

3	4F
	40
C	39
H	76
I	06
P	73
	00
u	1C
P	73
	00
B	7C
U	3E
I	06
L	38
T	78
	00
B	7C
Y	6E
	00
	FF
A	77
N	37
D	5E
Y	6E
	FF
B	7C
A	77
S	6D
I	06
L	38
	FF
B	&c
E	79
R	33
T	78
	FF
B	7C
I	06
L	38
L	38
	FF
B	7C
O	3F
B	7C
	FF
B	7C
R	33
U	3E
C	39
E	79
	FF
C	39
A	77
R	33
L	38
	FF

C	39
H	76
A	77
R	33
L	38
E	79
S	6D
	FF
	00
	00
E	79
N	37
T	78
E	79
R	33
	00
I	06
N	37
P	73
U	3E
T	78
	00
V	3E
A	77
L	38
U	3E
E	79
	00
	00
	00
	00
	00
	FF
C	39
L	38
I	06
F	71
F	71
	FF
C	39
L	38
I	06
V	3E
E	79
	FF
C	39
R	33
I	06
S	6D
	FF
C	39
O	3F
L	38
I	06
N	37
	FF

C	39
R	33
A	77
I	06
G	3D
	FF
D	5E
A	77
V	3E
I	06
D	5E
	FF
D	5E
O	3F
U	3E
G	3D
	FF
E	79
D	5E
	FF
E	79
V	3E
A	77
N	37
	FF
G	3D
E	79
O	3F
R	33
G	3D
E	79
	FF
G	3D
L	38
E	79
N	37
	FF
G	3D
R	33
E	79
G	3D
	FF
I	06
A	77
N	37
	FF
J	1E
O	3F
H	76
N	37
	FF
P	73
A	77
T	78
	FF

P	73
E	79
T	78
E	79
R	33
	FF
P	73
H	76
I	06
L	73
I	
P	
	FF
R	33
A	77
L	38
P	73
H	76
	FF
R	33
O	3F
Y	6E
	FF
S	6D
C	39
O	3F
T	78
T	78
	FF

S	6D
T	78
A	77
N	37
	FF
T	78
O	3F
N	37
Y	6E
	FF
L	38
I	06
T	78
T	78
L	38
E	79
	00
O	3F
L	38
	00
I	06
	FF
?	53
?	53
?	53
	FF

-	40
-	40
-	40
G	3D
U	3E
E	79
S	6D
S	6D
	40
	40
	40
	FF
A	77
N	37
	00
O	3F
L	38
D	5E
	00
P	73
R	33
O	3F
	FF
1	06
9	6F
8	7F
5	6D
	00
	00
	FF

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

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

NUMBERS AND LETTERS

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

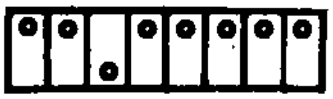
Some of the letters are hard to create on a 7-segment display and the closest possible resemblance has been created.

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 OATA

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



0200

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

For page 1. turn on switch 1. Fer page 2. turn on switch 2. For page 3. turn on switches 1 and 2. etc. Switches 8, 10, 20, 40, and 80 are meskad OFF via the instruction at 206 and thus they de net affect tha page-acceseing.

The progrem is designed to loop sround FF bytes and et page '2' the program is capeble of reading itself!!

At paga zaro (er any other page) the program starts by displaying tho eddresa value. This will be shewn with LOW BRIGHTNESS. Pushing butten A will display the value of data at the eddress. This will be shown with FULL BRIGHTNESS.

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

A fast-forward facility is provided by pushing button B when tho address valus ie baing displayed. This will enabla you to fast-forward around e page to pick up a missed lecation.

You cen selact a differant page numbar at any time and the corract deta will be displeyed.

This pregram is very handy for reeding the contents ef the EPROM and proving tha data to be as steted.

The display values ere generated from a byte table eituated 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 tee soon we have run eut of space. There are lets mere programs in the EPROM end these will be covered in the next issue.

When you buy e kit you will be able te accese these programs and see how they work.

The Microcomp is designed to fit into s cassette cese and be stored like a book. Hopefully you will be using it all the time and it wan't see the book-shelf. I hepe 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 sur 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 an the switches is loaded into the accumulator.
AND 07             206  E6 07     The accumulator is ANDed with 7 - enly 01. 02 & 04 detected.
LD D,A             208  57        The value en the switches (up to 07) is saved in 'D'.
LD A,E             209  7B        The COUNT REGISTER is loaded in the sccumulator.
AND 0F             20A  E6 0F     AND 0F removes the 4 upper bits leaving the 4 lewer bits.
LD HL 0280         20C  21 80 02  Lead HL with the start ef the BYTE TABLE.
ADD A,L            20F  85        ADD 80 te the accumulator.
LD L,A             210  6F        A new volue for L is created (far later use).
LD A,00            211  3E 00     The accumulater is zereed.
OUT (82),A         213  D3 02     The accumulater is outputted to port 2.
LD B,10            215  06 18     9 is leaded with 10 (16 in decimal)
DJNZ 0217          217  10 FE     DJNZ  A dalay ef 16 is craatad.
LD A,(HL)          219  7E        The accumulator is loaded with the valus peinted
                                      te by HL and outputted te port 2.
OUT (02),A         21A  D3 82
LD A,E             21C  7B        The count register is leaded inte the sccumulator.
RRA                21D  1F        The accumulater is retated RIGHT. The 4 high bits
RRA                21E  1F              movs dewn te the 4 lower places and are ANDed
RRA                21F  1F            with 0F .
RRA                220  1F
AND 8F             221  E6 0F      AND 0F remeves the 4 uppar bits
LD HL,0280         223  21 80 82  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 velue for L is creatsd.
LD A,00            228  3E 00     Zere the accumuletor.
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        Loed the accumulator with the velue 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,(81)          235  DB 01     Leok at tha input switchas
BIT 7,A            237  CB 7F     Test bit 7 to soe if switch A is prassed.
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 bin 2 of registar C to sea if it '1' ar '0'.
JR NZ 0204         23F  20 C3     If it is '1', jump to line 3. If it is zaro, jump to next loop.
JR 024E            241  18 0B     Jump to start of laop '2'
RES 2,C            243  CB 91     Resst bit 2 of register C.
BIT 6,A            245  CB 77     Test bit 5 te sse if hutton B is prassed.
JR Z, 0204         247  28 BB     If it is not pressed. jump to line 3.
INC E .NOP. NOP    249  1C 00 00  Incrament register E
JR 0204            24C  18 B6     Jump to line 3.
LD HL,0288         24E  21 80 02  Load HL with start of byte table.
LD A,(DE)          251  1A        Load A with the data pointed te by DE.
AND 8F             252  E6 0F     And the accumulstor with 0F.
ADD A,L            254  85        Add register L to the accumulator.
LD L,A             255  6F        Create e new value far L.
LD A,(HL)          256  7E        Loed A with the data pointed to by HL.
OUT (02),A         257  D3 02     Output this data te port 2.
LD A,(DE)          259  1A        Loed A with the data byte peinted te by DE.
RRA                25A  1F        Rotate the sccumulator RIGHT so that the 4 high order bite
RRA                25B  1F            are shifted te the 4 lower positions.
RRA                25C  1F
RRA                25D  1F
AND 0F             25E  E6 0F     AND the accumulator with 0F to remeve 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 accumulater.
LD L,A             264  6F        Create a new value for L.
LD A,(HL)          265  7E        Load A with the value peinted to by HL.
SET 7,A            266  CB FF     SET bit 7 ef the accumulator te turn on display 1.
OUT (02),A         268  D3 02     Dutput the accumulator te 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 hes not been pressed.
SET 2,C            278  CB d1     BET bit 2 of register C indicating button A pressed.
BIT 1,C            272  CB 49     Test bit 1 ef register C to see if button A hes been released.
JR NZ 024E         274  20 D8     Jump if bit 1 ef register C is '1'.
INC E .NOP. NOP    276  1C 00 08  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 ef loop 2.
```

TALKING ELECTRONICS

$2.20

$3.00NZ

Issue No 14

CONTINUITY TESTER

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

CRYSTAL OSCILLATOR   CO-ORDINATOR   GUITAR PRACTICE AMP

TEC-1A & 1B

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

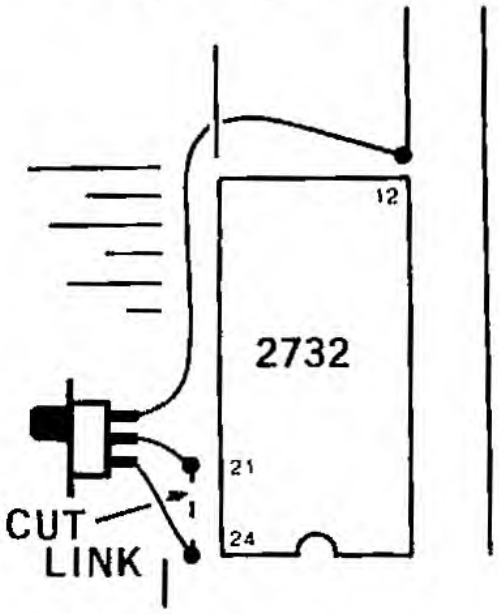
TALKING ELECTRONICS COMPUTER

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

PART V

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

TEC 1B with SHIFT KEY FITTED.



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

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

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

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

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

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

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

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

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

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

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

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

THE 2732 MONITOR

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

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

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

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

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

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

Gradually you will realise it is best to use MON 2 for most of your programs.

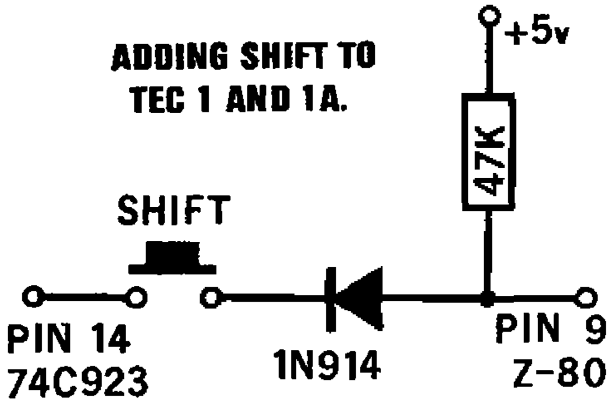
There are two major differences between MON 1B and MON 2. MON 1B uses a simple routine that places the value of a key directly into the accumulator, without firstly saving the value of the accumulator. Thus its original value is destroyed. MON 2 loads the key value into location **08E0** and thus your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

Unless you load directly into the A register.

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

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

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



TEC 1A/1B CONSTRUCTION HINTS:

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

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

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

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

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

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

NOTES ON THE 8x8 DISPLAY

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

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

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

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

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

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

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

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

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

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

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

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

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

DISPLAYING LETTERS AND NUMBERS

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

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

The letter M is displayed as a small 'n', with a bar over the top. This corresponds to a feature in mathematics where a dot is placed over the first and last digits in a number to indicate the number repeats. (This is called a recurring number or recurring fraction).

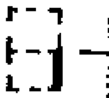
The letter W is displayed as a small 'u' with a bar over the top, for the same reason. The letter 'U' is displayed as a capital letter while V is a small 'u'.

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

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

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

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



TESTING A BLANK 2716 FOR FF's

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

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

- by James Doran. 3218

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

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

Your ideas, combined with others, will help us to present an article.

MON 2 HEX LISTING:

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

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

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

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

MON 2 HEX LISTING FOR TEC 1B:

```
0000  C3 00 02 FF
0004  FF FF FF FF
0008  2A C0 08 E9
000C  FF FF FF FF
0010  2A C2 08 E9
0014  FF FF FF FF
0018  2A C4 08 E9
001C  FF FF FF FF
0020  2A C6 08 E9
0024  FF FF FF FF
0028  2A C8 08 E9
002C  FF FF FF FF
0030  2A CA 08 E9
0034  FF FF FF FF
0038  2A CC 08 E9
003C  FF FF FF FF
0040  FF FF FF FF
0044  FF FF FF FF
0048  FF FF FF FF
004C  FF FF FF FF
0050  FF FF FF FF
0054  FF FF FF FF
0058  FF FF FF FF
005C  FF FF FF FF
0060  FF FF FF FF
0064  FF FF F5 DB
0068  00 32 E0 08
006C  F1 ED 45 FF
0070  FF FF FF FF
0074  FF FF FF FF
0078  FF FF FF FF
007C  FF FF FF FF
0080  EB 28 CD AD
0084  2E A7 E7 29
0088  EF 2F 6F E6
008C  C3 EC C7 47
0090  E3 00 28 E8
0094  4E C2 2D 6B
0098  EB 4F 2F 4B
009C  A7 46 EA E0
00A0  AC A4 AE C9
00A4  10 08 18 04
00A8  2C 00 FF FF
00AC  FF FF FF FF
00B0  00 00 00 00
00B4  FF FF FF FF
00B8  FF FF FF FF
00BC  FF FF FF FF
00C0  1B 18 1E 1D
00C4  12 17 0E 29
00C8  0B 22 29 17
00CC  12 0C 24 29
00D0  29 29 29 29
00D4  FE 1C 1D 16
00D8  17 0E FF FF
00DC  FF FF FF FF
00E0  CD 09 02 03
00E4  16 04 CD 09
00E8  02 0B CD 90
00EC  04 CD 70 02
00F0  21 DF 08 CB
00F4  C6 CB 8E C3
00F8  78 03 FF FF
00FC  FF FF FF FF
0100  FD 10 30 FD
0104  11 EF 12 E1
0108  13 D5 14 C9
010C  15 BE 16 B3
0110  18 A9 19 9F

0114  1A 90 1C 8E
0118  1E 86 20 7F
011C  22 77 24 71
0120  26 6A 28 64
0124  2A 5F 2D 59
0128  2F 54 32 50
012C  35 4B 38 47
0130  3C 43 3F 3F
0134  43 3C 47 38
0138  4B 35 50 32
013C  54 2F 59 2D
0140  5F 2A 64 28
0144  6A 26 71 24
0148  77 22 7F 20
014C  86 1E 8E 1C
0150  96 1A 9F 19
0154  A9 18 B3 16
0158  BE 15 C9 14
015C  D5 13 E1 12
0160  EF 11 FD 10
0164  FF FF FF FF
0168  FF FF FF FE
016C  FF FF FF FF
0170  C5 D5 E5 F5
0174  A7 28 03 5F
0178  10 01 1E 80
017C  21 00 01 07
0180  85 6F 4E 23
0184  46 7B D3 01
0188  10 FE 46 AF
018C  D3 01 10 FE
0190  0D 20 F1 F1
0194  E1 D1 C1 C9
0198  FF FF FF FF
019C  FF FF FF FF
01A0  F5 E5 2A D6
01A4  08 7E FE FF
01A8  20 03 E1 F1
01AC  C9 FE FE 28
01B0  F1 23 CD 78
01B4  01 18 EE FF
01B8  FF FF FF FF
01BC  FF FF FF FF
01C0  21 DF 08 CB
01C4  46 20 07 CB
01C8  C6 CB 8E C3
01CC  78 03 CB 66
01D0  CB CE C3 78
01D4  03 FF FF FF
01D8  C5 06 80 CD
01DC  A0 02 10 FB
01E0  C1 C9 FF FF
01E4  ED 4B D2 08
01E8  CD 90 04 CD
01EC  70 02 C3 78
01F0  03 FF ED 4B
01F4  D4 08 CD 90
01F8  04 CD 70 02
01FC  C3 78 03 FF
0200  ED 73 E0 08
0204  31 00 09 F5
0208  C5 D5 E5 DD
020C  E5 FD E5 08
0210  D9 F5 C5 D5
0214  E5 ED 57 F5
0218  AF 32 CC 08
021C  32 CD 08 3E
0220  FF 32 E0 08
0224  C3 48 02 FF

0228  FF FF FF FF
022C  FF FF FF FF
0230  FF FF FF FF
0234  FF FF FF FF
0238  FF FF FF FF
023C  FF FF FF FF
0240  31 C0 08 AF
0244  D3 02 D3 01
0248  21 B0 00 11
024C  D8 08 01 05
0250  00 ED B0 CD
0254  70 02 3E 08
0258  CD 70 01 3E
025C  0F CD 70 01
0260  3E 01 32 DF
0264  08 CD A0 01
0268  CD 60 03 18
026C  F8 FF FF FF
0270  F5 E5 C5 CD
0274  69 02 E6 F0
0278  0F 0F 0F 0F
027C  32 DC 08 0A
0280  E6 0F 32 DD
0284  08 C1 E1 F1
0288  C9 21 D8 08
028C  7E 07 07 07
0290  07 23 86 47
0294  23 7E 07 07
0298  07 07 23 86
029C  4F 0A C9 FF
02A0  F5 E5 D5 C5
02A4  11 D8 08 AF
02A8  D3 01 CD 50
02AC  03 CB 4E 28
02B0  02 C6 E7 D3
02B4  02 3E 20 D3
02B8  01 06 20 10
02BC  FE AF D3 01
02C0  CD 50 03 CB
02C4  4E 28 01 CB
02C8  E7 D3 02 3E
02CC  10 D3 01 06
02D0  20 10 FE AF
02D4  D3 01 CD 50
02D8  03 CB 4E 28
02DC  02 CB E7 D3
02E0  02 3E 08 D3
02E4  01 06 20 10
02E8  FE AF D3 01
02EC  CD 50 03 CB
02F0  4E 28 01 CB
02F4  E7 D3 02 3E
02F8  04 D3 01 06
02FC  20 10 FE AF
0300  D3 01 00 C3
0304  18 03 FF FF
0308  FF FF FF FF
030C  CD 09 02 C5
0310  E1 31 C0 08
0314  E9 FF FF FF
0318  CD 50 03 CB
031C  46 28 01 CB
0320  E7 D3 02 3E
0324  02 D3 01 06
0328  20 10 FE AF
032C  D3 01 CD 50
0330  03 CB 46 28
0334  02 CB E7 D3
0338  02 3E 01 D3

033C  01 06 20 10
0340  FE AF D3 01
0344  C1 D1 E1 F1
0348  C9 FF FF FF
034C  FF FF FF FF
0450  21 80 00 1A
0354  85 6F 7E 13
0358  21 DF 08 C9
035C  FF FF FF FF
0360  F5 E5 21 E0
0364  08 3E FF BE
0368  28 0E 7E E6
036C  1F CB 6E 20
0370  02 C6 14 C3
0374  A8 03 FF FF
0378  E1 F1 C9 FF
037C  FF E1 F1 C9
0380  FF FF FF FF
0384  CD 09 02 C5
0388  DD E1 DD 23
038C  DD E5 E1 7C
0390  FE 40 28 08
0394  DD 7E 00 DD
0398  77 FF 10 EE
039C  3E 80 32 FF
03A0  3F CD 70 02
03A4  C3 78 03 FF
03A8  C6 01 CD 70
03AC  01 C3 21 04
03B0  CD 09 02 0B
03B4  DD 21 FE 3F
03B8  DD 7E 00 DD
03BC  77 01 DD 2B
03C0  DD E5 E1 79
03C4  BD 20 F2 78
03C8  BC 20 ED DD
03CC  36 01 00 CD
03D0  70 02 C3 78
03D4  03 FF FF FF
03D8  E5 F5 DD E5
03DC  C5 AF 32 DF
03E0  08 06 06 21
03E4  D8 08 3E 29
03E8  77 23 10 FC
03EC  2A D0 08 7E
03F0  FE FF 20 06
03F4  C1 DD E1 F1
03F8  E1 C9 FE FE
03FC  28 EE DD 23
0400  D0 08 06 05
0404  DD 7E 01 DD
0408  77 00 DD 23
040C  10 F6 7E 32
0410  DD 08 23 06
0414  40 CD A0 02
0418  10 FB 18 D3
041C  FF FF FF FF
0420  FF D6 01 36
0424  FF CB 67 C2
0428  C0 04 CB 6F
042C  C2 C0 04 21
0430  DF 08 CB 46
0434  CA 55 04 57
0438  CD 09 02 21
043C  DF 08 CB 5E
0440  20 03 AF CB
0444  DE 07 07 07
0448  07 E6 F0 82
044C  02 CD 70 02

0450  C3 7D 03 FF
0454  FF 57 21 DF
0458  08 CB 9E CB
045C  66 20 08 01
0460  00 00 CD 90
0464  04 CB E6 CD
0468  89 02 78 07
046C  07 07 07 E6
0470  F0 5F 79 07
0474  07 07 07 E6
0478  0F 83 47 79
047C  07 07 07 07
0480  E6 F0 82 4F
0484  CD 90 04 CD
0488  70 02 C3 7D
048C  03 FF FF FF
0490  F5 E5 21 D8
0494  08 78 E6 F0
0498  07 07 07 07
049C  77 23 78 E6
04A0  0F 77 23 79
04A4  E6 F0 07 07
04A8  07 07 77 23
04AC  79 E6 0F 77
04B0  E1 F1 C9 FF
04B4  FF FF FF FF
04B8  FF FF FF FF
04BC  FF FF FF FF
04C0  21 DF 08 CB
04C4  9E CB A6 FE
04C8  10 CA E0 00
04CC  FE 11 CA E6
04D0  00 FE 12 CA
04D4  0C 03 FE 13
04D8  CA C0 01 FE
04DC  14 CA 50 05
04E0  FE 15 CA FF
04E4  FF FE 16 CA
04E8  FF FF FE 17
04EC  CA F2 01 FE
04F0  18 CA 70 05
04F4  FE 19 CA FF
04F8  FF FE 1A CA
04FC  FF FF FE 1B
0500  CA FF FF FE
0504  1C CA 60 06
0508  FE 1D CA FF
050C  FF FE 1E CA
0510  FF FF FE 1F
0514  CA FF FF FE
0518  20 CA FF FF
051C  FE 21 CA FF
0520  FF FE 22 CA
0524  FF FF FE 23
0528  CA FF FF FE
052C  24 CA B0 03
0530  FE 25 CA 84
0534  03 FE 26 CA
0538  FF FF FE 27
053C  CA E4 01 C3
0540  78 03 FF FF
0544  FF FF FF FF
0548  FF FF FF FF
054C  FF FF FF FF
0550  CD 09 02 60
0554  69 3A E1 08
0558  23 BE 20 FC
055C  44 4D CD 90
0560  04 C3 53 02
0564  FF FF FF FF
```

HOW THE CIRCUIT WORKS
(and a general discussion.)

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

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

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

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

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

STARTING UP

When the power is applied to the computer, the reset line on the Z-80 is taken low for an instant via the 100n capacitor and this resets the internal workings of the Z-80.

Its first operation is to look for the first byte of data at address zero, in the monitor. Depending on this being a one-byte, two-byte or three-byte instruction, the Z-80 will execute it or request one or two more bytes.

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

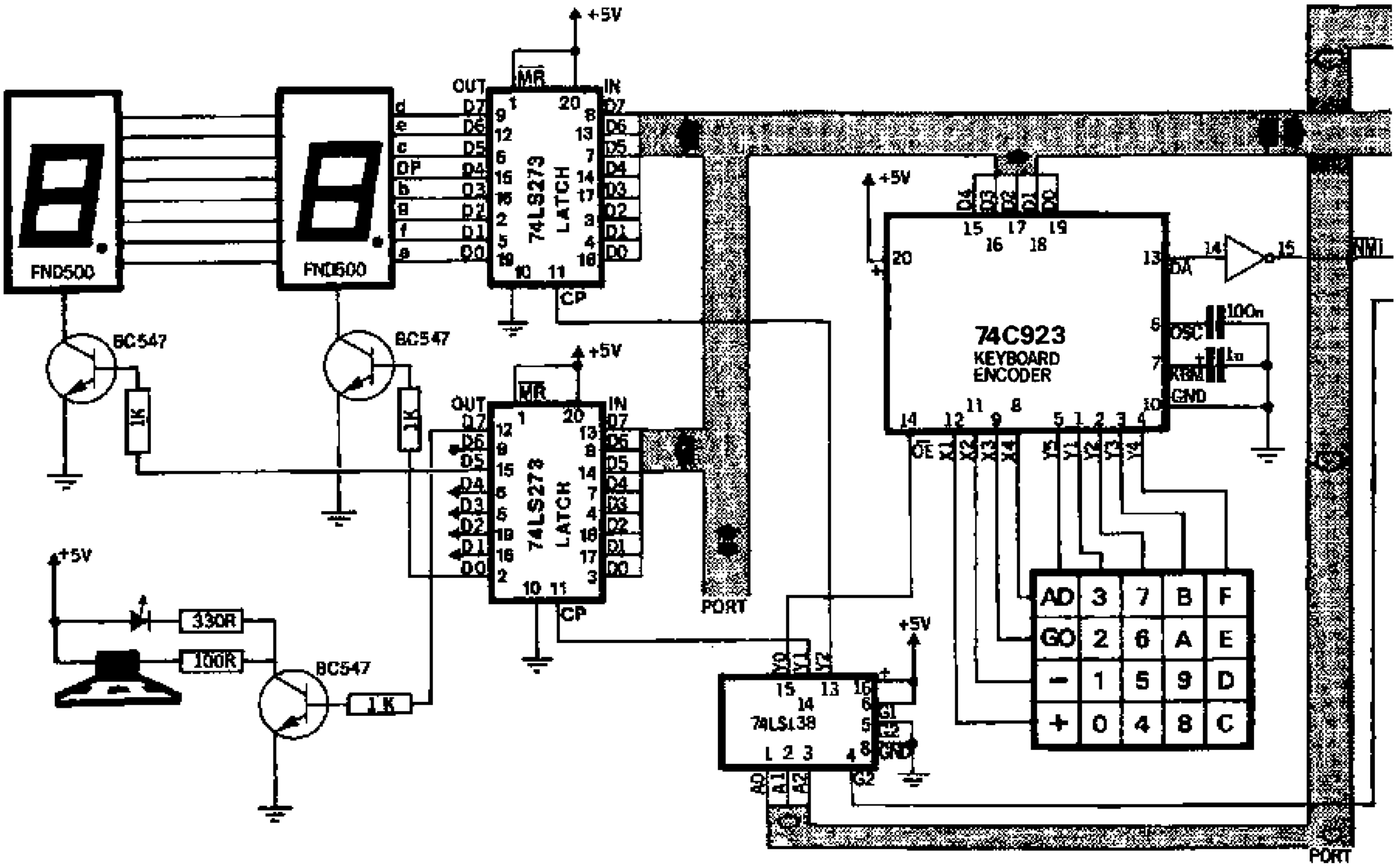
All signals within the computer are at a level equal to rail voltage (called HIGH) or ground (called LOW). For this reason they are celled digital circuits.

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

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

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

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



TEC 1B COMPUTER CIRCUIT

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

This has been done so that the TEC 1B is compatible with the original TEC 1 and it can be upgraded by adding a monitor switch and a programmed 2732 EPROM.

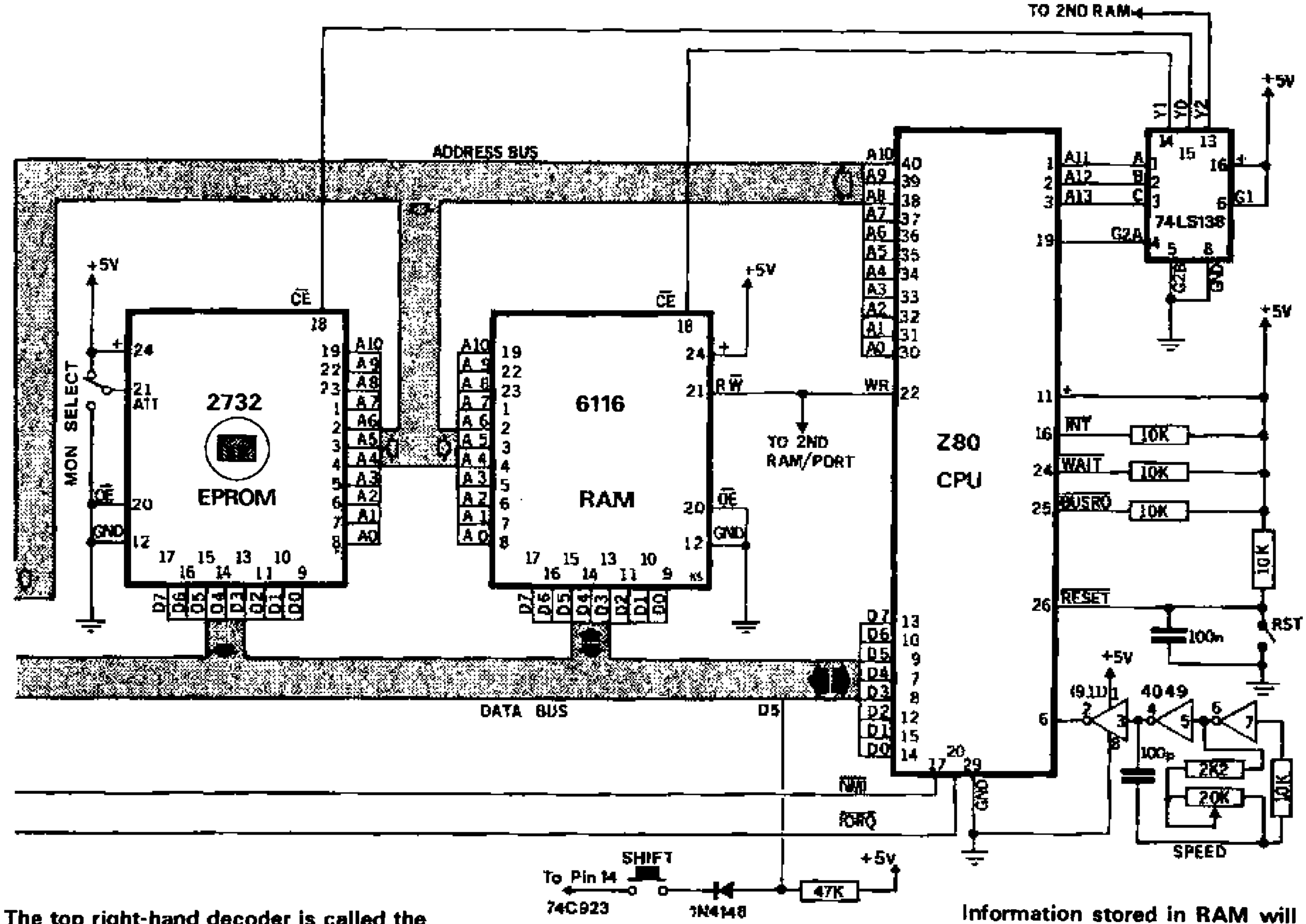
The original TEC 1 had a 2716 EPROM but these chips are no longer manufactured and thus a 2732 is now used. When a 2732 is placed in a 2716 socket the upper half of the chip is accessed and thus MON 1 program has been placed in the upper half.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

'ADD-ONs'

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

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

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

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

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

PROGRAMS FOR THE TEC DISPLAYS and a sound Program:

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

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

SPACE INVADERS 'SHOOTING'

Philip Barns, 2118

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

This program does just that.

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

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

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

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

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

LD C,B          828     48
DEC C           829     0D
JR NZ 0829      82A     20 FD
RETURN          82C     C9
```

THE BOX G.L Dunt 3219.

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

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

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

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

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

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

at 0840:

```
↓  01     ↓  01     ↓  01     ↓  20
   E4        E4        80        E4
   01        01        10        20
   E4        E4        C4        E4

   01        01        01        10
   E8        E1        80        E4
   01        01        20        10
   E8        E1        E0        E4

   01        01        02        08
   E4        01        80        E4
   01        02        20        08
   E4        E0        E0        E4

   01        01        04        04
   E2        04        80        E0
   01        02        20        08
   E2        E0        E0        04

   01        01        08        02
   E4        80        80        E0
   01        02        20        08
   E4        E0        E0        04

   01        01        10        01
   E4        80        04        E0
   01        04        20        04
   E4        A4        E0        04

   01        01        20        01
   E2        80        E1        E0
   01        08        20        02
   E2        64        E1        04
```

Halilovic's Piano:

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

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

BOOMERANG

G. Sheehan & D. Svendsen. 3175

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

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

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

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

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

The delay at 0900 produces the speed of execution.

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

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

at 0820:

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

Delay at 0900:

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

PROGRAMS FOR THE 8x8 DISPLAY:

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

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

FAN OUT Mk III

Dean Svendsen 3175.

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

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

at 0815:

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


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

BOUNCING BALL AND ROLLING BALL.

G.L. Dunt. 3219.

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

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

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

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

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

If we repeat this 4 times, the whole of the board will be covered. This will reproduce the effect as described on P. 26 of issue 12. Using the same technique, we can travel across the display and back again, to produce a weaving effect as the LED advances up the display. To complete the travel we need to move the LED from the top right hand corner to the lower left hand corner, ready for the start of the next sequence.

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

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

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

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

```
LD C,01          0800     0E 01
LD D,01          0802     16 01
LD A,C           0804     79
OUT (03),A       0805     D3 03
LD A,D           0807     7A
OUT (04),A       0808     D3 04
CALL 0900        080A     CD 00 09
RLC D            080D     CB 02
JR NC 0807       080F     30 F6
RR D             0811     CB 1A
RLC C            0813     CB 01
LD A,C           0815     79
OUT (03),A       0816     D3 03
LD A,D           0818     7A
OUT (04),A       0819     D3 04
CALL 0900        081B     CD 00 09
RR D             081E     CB 1A
JR NC,0818       0820     30 F6
RL D             0822     CB 12
RLC C            0824     CB 01
JR NC,0804       0826     30 DC
RRC C            0828     CB 09
LD A,D           082A     7A
OUT (04),A       082B     D3 04
LD A,C           082D     79
OUT (03),A       082E     D3 03
CALL 0900        0830     CD 00 09
RRC C            0833     CB 09
JR NC,082D       0835     30 F6
RL C             0837     CB 11
RLC D            0839     CB 02
LD A,D           083B     7A
OUT (04),A       083C     D3 04
LD A,C           083E     79
OUT (03),A       083F     D3 03
CALL 0900        0841     CD 00 09
RLC C            0844     CB 01
JR NC,083E       0846     30 F6
RRC C            0848     CB 09
RLC D            084A     CB 02
JR NC,082A       084C     30 DC
RRC D            084E     CB 0A
RRC D            0850     CB 0A
LD A,D           0852     7A
OUT (04),A       0853     D3 04
CALL 0900        0855     CD 00 09
RR D             0858     CB 1A
JR NC,0852       085A     30 F6
RRC C            085C     CB 09
LD A,C           085E     79
OUT (03),A       085F     D3 03
CALL 0900        0861     CD 00 09
RRC C            0864     CB 09
JR NC,085E       0866     30 F6
JP 0800          0868     C3 00 08

At 0900:

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

RAIN DROPS:

Jim Robertson,

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

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

  at 0900:

  SCAN

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

  at 0A00:

  RANDOM NUMBERS:

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

PHONE DIALLER

TURNING THE TEC INTO A PHONE DIALLER

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

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

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

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

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

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

PHONE DIALLER PROGRAM 1.

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

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

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

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

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

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

PHONE DIALLER - Part 1

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

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

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

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

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

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

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

```
at 0880:

EB
28
CD
AD
2E
A7
E7
29
EF
AF
```

PHONE DIALLER - Part 2

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

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

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

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

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

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

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

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

HOW THE PROGRAM WORKS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

DIALLER Part 2 listing:
Main Program:

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

Clear:

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

Shift:

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

Scan:

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

```
at 08A5:

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

PHONE DIALLER · Part 3

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

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

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

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

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

USING THE PROGRAM

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

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

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

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

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

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

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

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

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

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

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

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

At the end of dialling, the name and number will scroll and you can confirm it to be correct.

A SUMMARY OF THE PROGRAM

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

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

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

PHONE DIALLER PROGRAM:

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

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

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

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

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

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

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

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

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

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

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

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

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

SUGGESTIONS

The program can be keyed into the TEC and fills about 3 pages, from 0800 to 0AEE.

After this is done, it is wise to save a copy of the program in non-volatile RAM so that it is not lost.

To save the program, type the following dump routine at 0F00:

```
11  00  10
21  00  08
01  00  07
ED  B0
C7
```

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

                                      This is the end of the MAIN PROGRAM. The sub-
BEEP                                  routines below are called by the main program.

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

CLEAR

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

KEY VALUE

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

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

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

Switch to ROM mode and everything will be preserved.

You can now turn the TEC off.

To transfer the program back to 0800, address1780 and change 2 of the bytes to the following:

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

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

Push GO again and the program will run.

All names and numbers will be available.

AUTO REDIAL

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

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

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

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

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


JR
```

MEMORY ADDRESS

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

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

PAUSE

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

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

SCAN 1

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

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

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

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

SCAN 2

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

SCAN 2 is identical to SCAN 1 in the scanning section. The only difference is the 'checking' instructions, to see if a particular key is pressed. SCAN 1 above checks to see if a function key is pressed, whereas SCAN 2 performs the scan without any checks.

By careful programming both routines could be incorporated into one. This would require a 'check bit' and if 'set', the sub-routine would check the function keys.

Cont. P.51:

. . . from P.20.

PHONE DIALLER Part III

EXPERIMENTING FURTHER

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

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

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

The diagram below shows how the DISPLAY BUFFER operates.

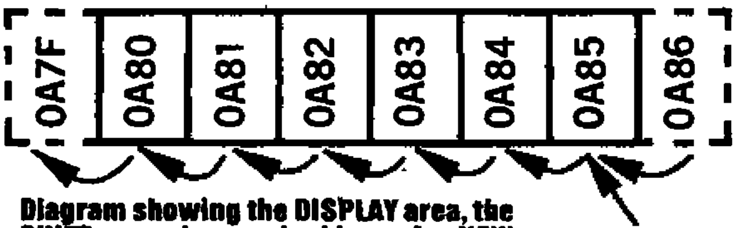


Diagram showing the DISPLAY area, the SHIFT procedure and address for NEW DATA.

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

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

SCROLL

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

SHIFT

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

at 0A00

DISPLAY TABLE:

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

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

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

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

at 0A0C:

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

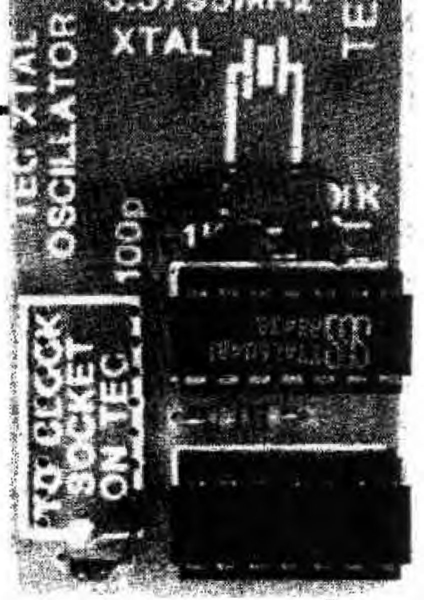
at 0A2C:

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

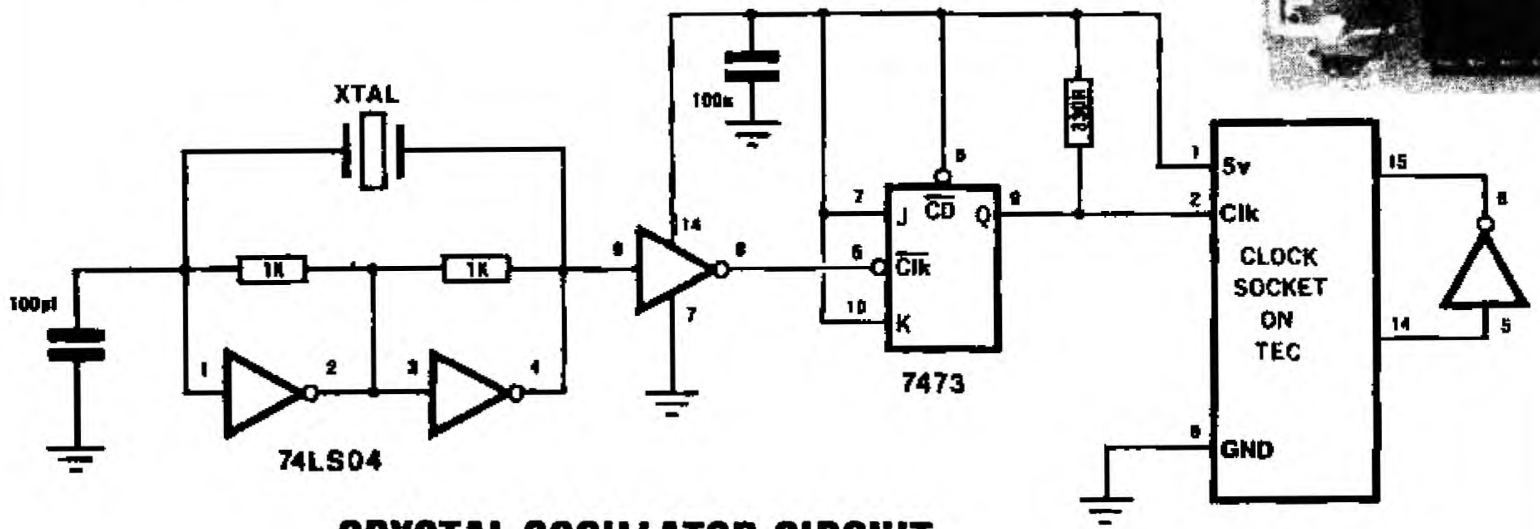
```
S  A7
P  4F
A  6F
C  C3
E  C7
=  84
F  47
   00
C  C3
L  C2
E  C7
A  6F
R  44
=  84
C  C3
   00
R  44
E  C7
T  46
U  EA
R  44
N  6B
=  84
A  6F
   00
D  EC
I  28
A  6F
L  C2
=  84
D  EC
   00
E  C7
N  6B
D  EC
=  84
E  C7
   00
   00
   00
   00
10 10
```

CRYSTAL OSCILLATOR

Kit of parts: $9.85
PC Board: $2.10
Complete: $11.95



CONVERTS THE TEC TO REAL-TIME CAPABILITY



CRYSTAL OSCILLATOR CIRCUIT

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

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

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

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

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

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

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

To remedy this situation Paul has produced a crystal oscillator module that plugs into the 4049 socket. It contains an inverter chip (74LS04) and a divider chip so that a 4MHz crystal or colour-burst (3.5795MHz) can be used (because they are cheap) and a divider chip (7473) to divide the frequency by two so that the TEC will run at about the maximum speed permissible for a Z-80 CPU.

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

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

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

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

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

A 100pf capacitor at the front end provides guaranteed start-up and the crystal provides a capacitive feedback that is a maximum at the fundamental frequency of the crystal. This is why the oscillator circuit operates at the frequency as specified on the crystal.

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

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

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

PARTS

1	-	330R
2	-	1k
1	-	100pf ceramic
1	-	100n monoblock
1	-	3.5795MHz crystal
1	-	74LS04 IC
1	-	7473 IC
2	-	14 pin IC sockets
1	-	16 pin dip header
1		CRYSTAL OSCILLATOR PC BOARD

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

ASSEMBLY

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

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

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

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

When you have the new board in place, the first program you can try is the Clock in issue 12, P.23. The best idea is to type it into the non-volatile RAM at **1000** and down-load it to **0900** via a block-transfer program:

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

To convert the program to operate with 4MHz crystal, two of the inbuilt delay values must be altered and a 'fine tune' delay added to the end of the program. This will create a clock that is accurate to within a second a day.

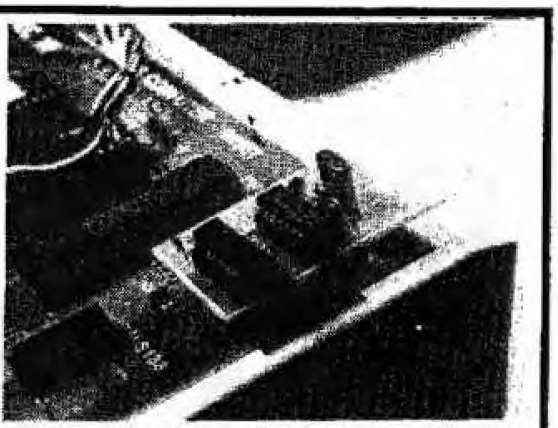
Type the complete program as per issue 12 then change the following locations and also add the extra 7 bytes:

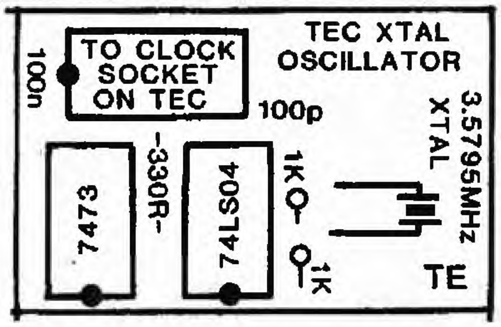
For a 4MHz crystal:

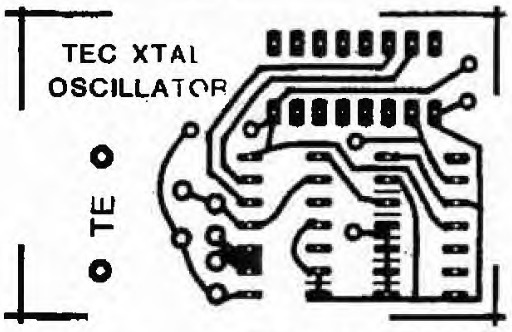
```
94C   06 FA
962   1E 41
970   C3 93 09
993   06 55
995   10 FE
997   C3 00 09
```

For a 3.5795MHz crystal:

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

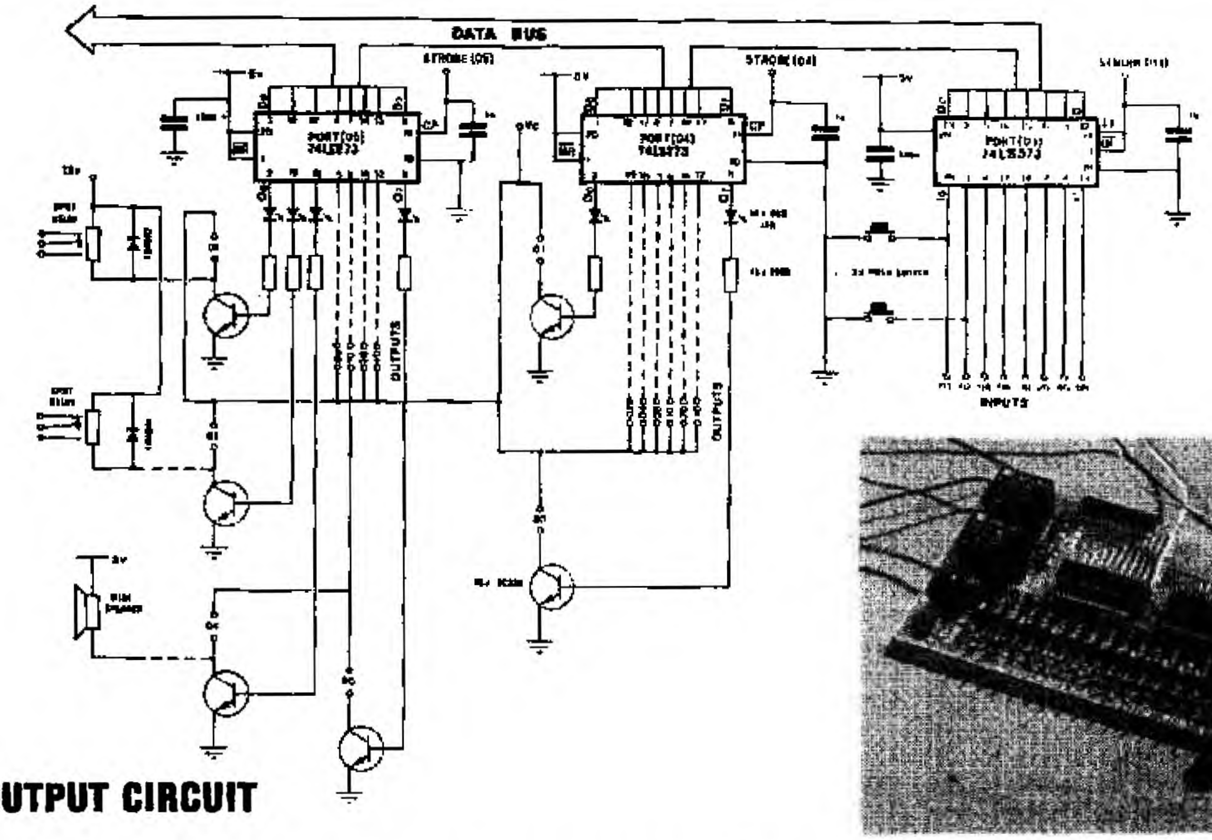




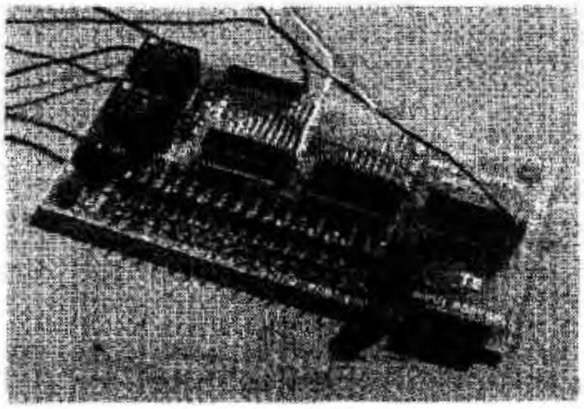


INPUT/OUTPUT MODULE

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



INPUT/OUTPUT CIRCUIT



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

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

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

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

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

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

Each of the 8 input and 16 output lines is further identified by a hex value on the PC overlay and this will assist you when writing a program.

The most interesting use for the board will undoubtedly be for robotics and when designing in this field, a whole new world of mechanical and electromechanical terms will be encountered.

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

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

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

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

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

PARTS

16 - 220R ¼watt

3 - 1n greencap
2 - 100n

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

2 - 74LS273 IC
1 - 74LS373 IC

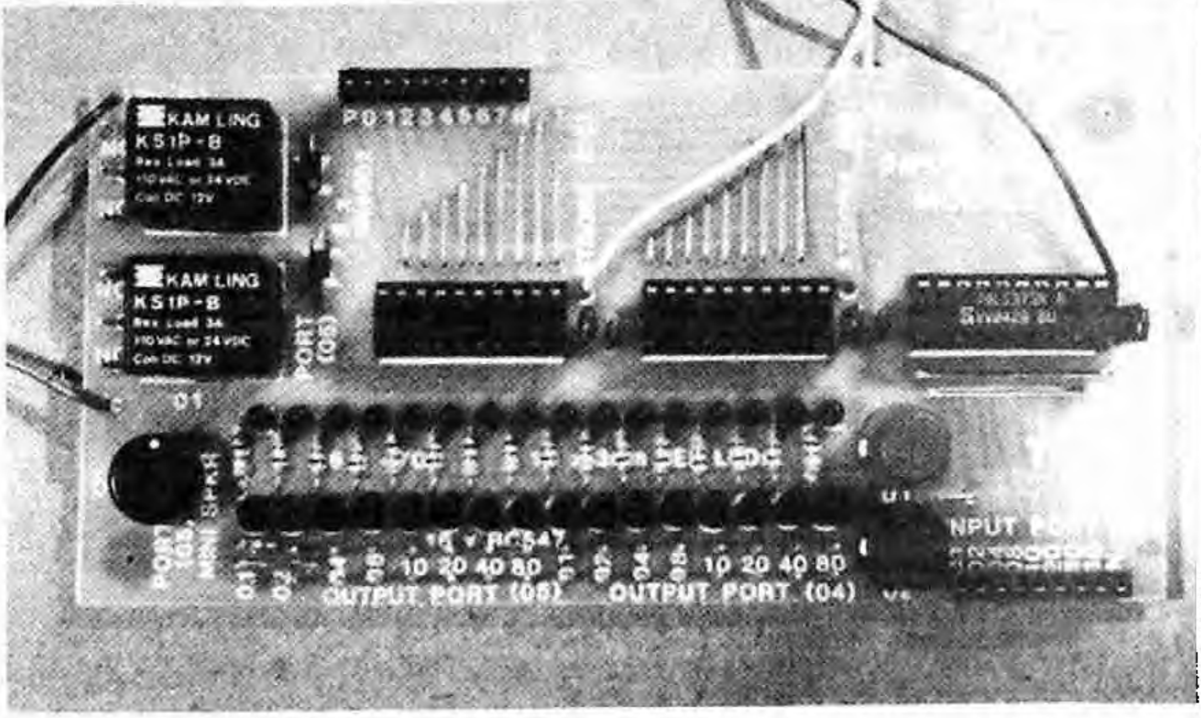
3 - 20 pin IC sockets
2 - PC mount push buttons
1 - Mini Speaker 8OR
2 - SPDT relays

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

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

1 - INPUT/OUTPUT MODULE PC

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



Equally suitable is PC board as it has a copper surface that can be soldered to and thus small brackets can be added for shafts etc.

The only material I would avoid is sheet metal. Even though it has good strength, the same can be provided via plastic with the use of a few strengthening pieces, without the difficulty of cutting folding and drilling. For tinplate to have any strength it must be reasonably thick and you will require heavy duty tools etc to shape it.

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

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

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

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

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

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

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

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

Apart from the problems you will encounter adapting the mechanics into the available space, there will be problems interfacing the motor to the electronics.

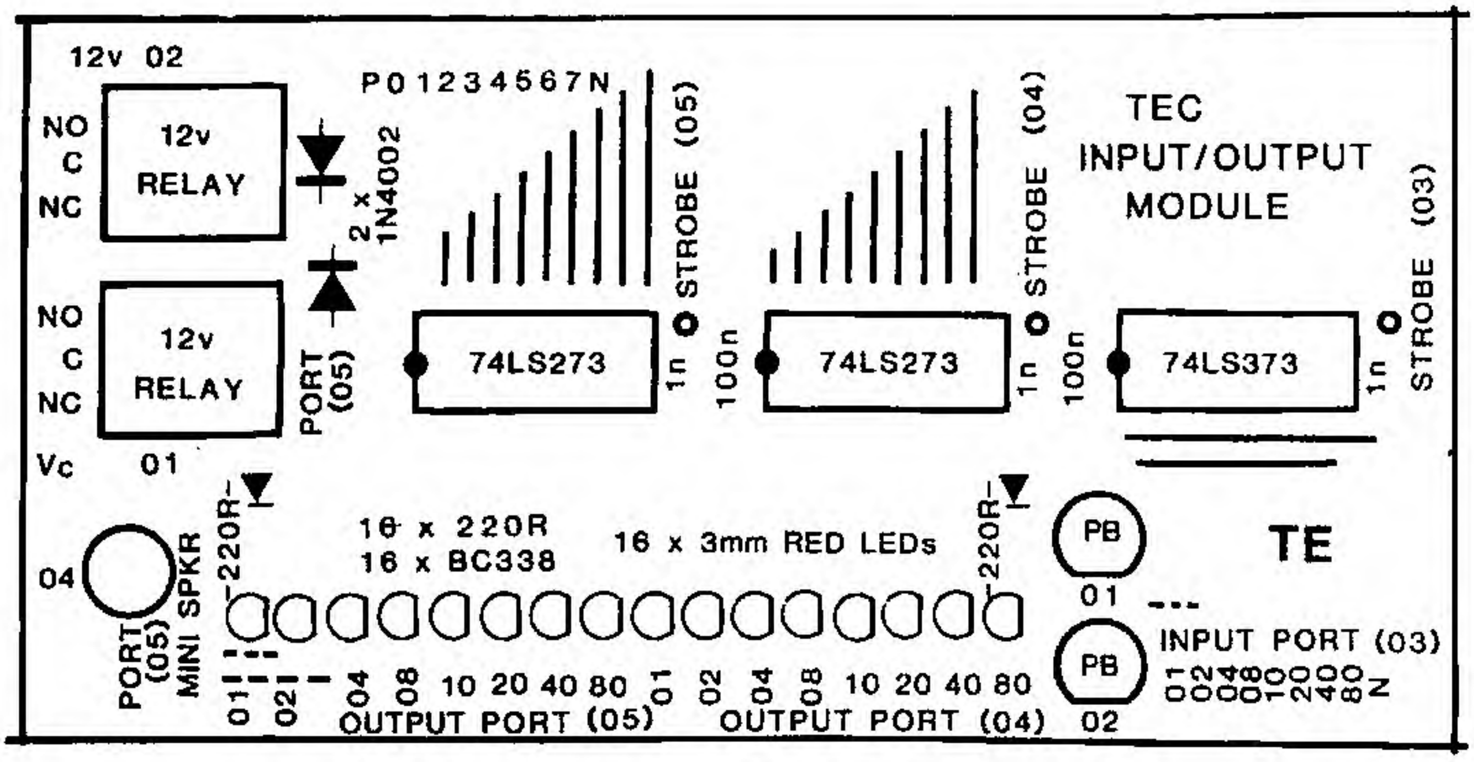
One of the major problems will be noise. Motors are inherently high noise producers and they must be kept far away from the electronics, both physically and electrically.

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

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

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

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



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

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

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

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

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

ASSEMBLY

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

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

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

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

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

Next fit the resistors, followed by the LEDs transistors and two spike-suppressing diodes. The overlay shows how these components are placed.

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

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

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

MATRIX PINS

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

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

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

TESTING

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

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

TEST PROGRAM

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

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

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

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

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

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

Follow through the program and see how this is done:

ORGAN PROGRAM

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

```
at 0A00:
00      64      3C
FA      BD      CF
84      5C      34
DE      F3      AF
7C      54      2C
BE      D7      FF
74      4C
F9      B7
6C      44
DD      EB
```

KEY PAD CONTROLS OUTPUT LINES

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

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

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

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

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

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

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

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

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

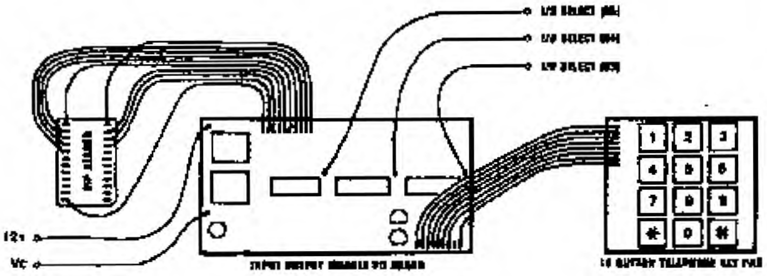
It's up to you. Study the program and try making some modifications.

For a very simple test program, try this:

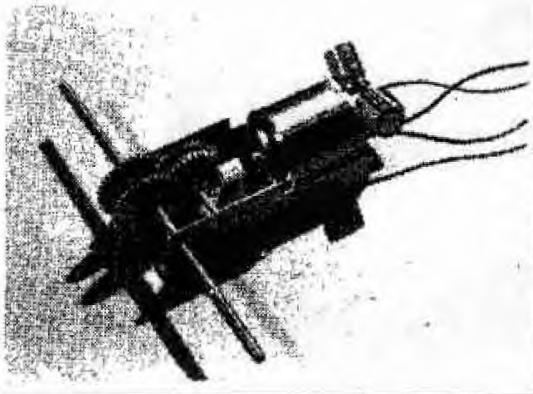
```
3E FF
D3 04
C7
```

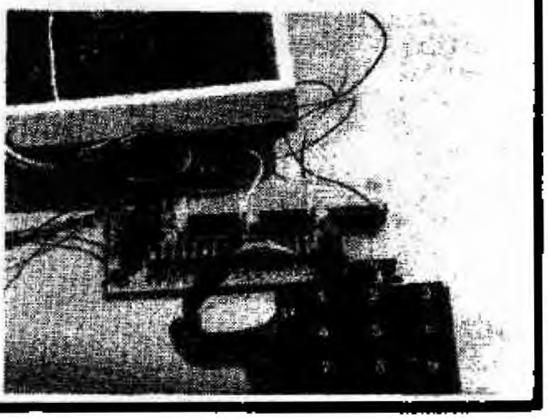
Eight LEDs will illuminate to show the program and board is working.

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

At 0A00:

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



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





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

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

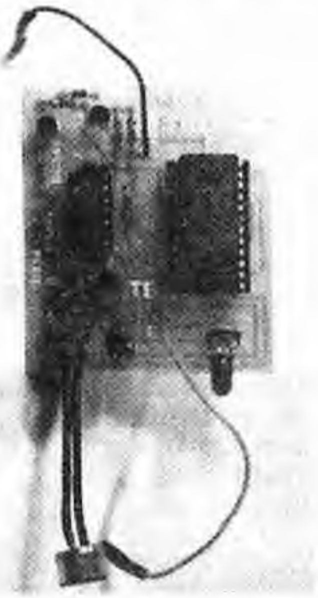
MICRO COMP

A 3-CHIP Z-80 COMPUTER

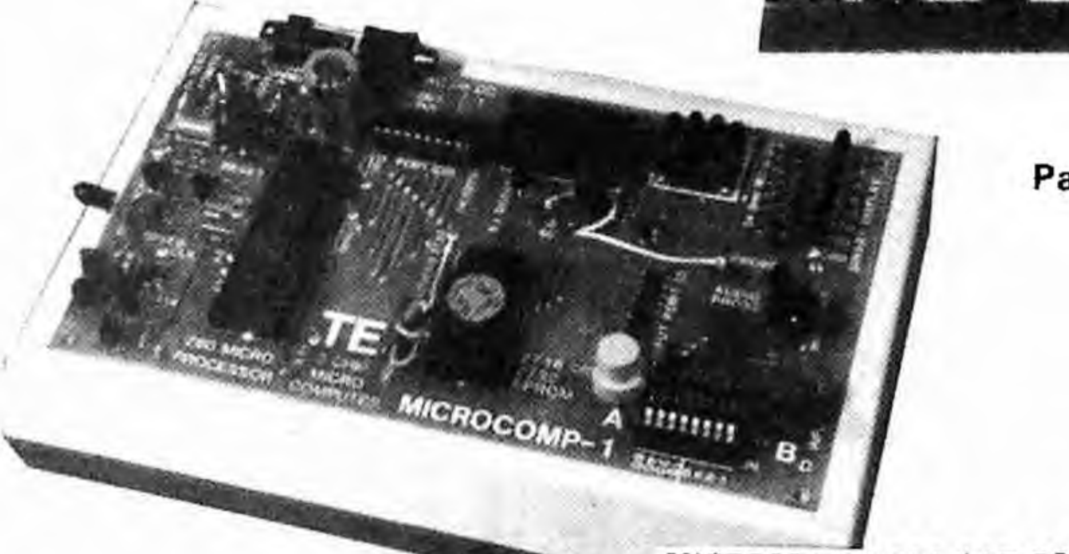
$59.95 COMPLETE
COMES WITH **FREE**
STORAGE BOX!!

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

TO BE RELEASED



MORSE TRAINER
$13.30complete



MICROCOMP CASE $15.00

COMPLETE MICROCOMP
MOUNTED ON RETEX CASE RA-1.

Part II

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

TUG O' WAR & BLACK JACK

TWO programs for the MICROCOMP.

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

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

TUG O' WAR

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

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

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

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

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

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

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

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

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

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

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

DESIGNING THE PROGRAM

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

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

The rest of the effect lies in the program.

This is how we went about designing it:

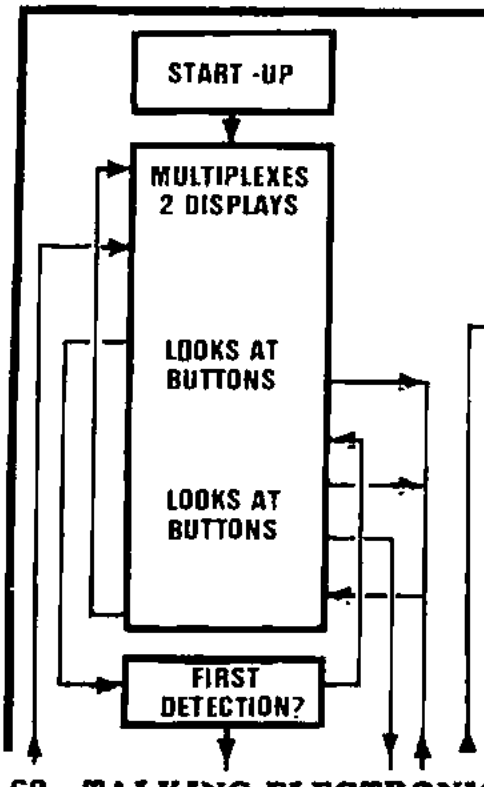
When the game starts, the two displays are illuminated with zeros. This requires a

cont. P. 62 . . .

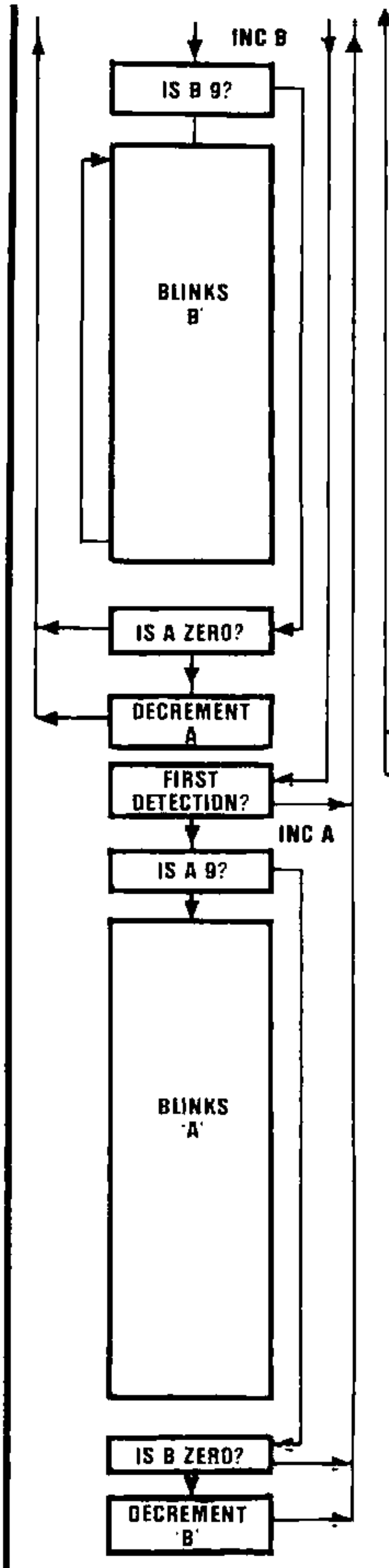
The TUG O WAR program starts below and continues on the next page. It requires a table of 46 bytes for the display and this is placed at **00C0:**

```
AT C0:

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

TUG O' WAR PROGRAM:



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



```
SET 0,C          0037  CB C1     Set bit 0 ef C befere processing butten B.
INC DE           0039  13        Incrament painter far RH display.
LD A,(DE)        003A  1A        Load A with second byte In teble.
CP 67            003D  FE 67     Compare the accum. with 67 te see if end of table hes been reached.
JR NZ,006D       003D  20 2E     Jump if end ef table NOT reached. Incremant if reached.
LD C,10          003F  0E 10     Loxd C with 10 for multiplexing time-length.
LD A,(DE)        0041  1A        Load the accumulator with data poimted to by DE.
OUT (01),A       0042  D3 02     Dutput to the latch.
LD B,10          0044  06 10     Load B with 10 for short deley.
DJNZ 0046        0046  10 FE     Decrement 6 16 timee.
XOR A            0048  AF        Zxro the eccumulator.
SET 7,A          0049  CB FF     Set the highest bit to turn on the LH diaplxy.
ADD A,(HL)       004B  86        ADD the byte pointed to by HL. te the eccumulator.
OUT (02),A       004C  D3 02     Dutput ta the latch.
LD B,10          004E  06 10     Laad B with 10 for short daley.
DJNZ 0050        0050  10 FE     Decremant B 16 timee.
DEC C            0052  0D        Decremant C.
JR NZ,0041       0053  20 EC     Jump if C nat zero.  Incremant if C is zera.
LD C,10          0055  0E 10     Load C with 10.
XOR A            0057  AF        Zera A ta turn off RH diaplay to craata BLINK.
OUT (02),A       0058  D3 02     Dutput ta the letch.
LD B,10          005A  06 10     Laad B with 10 ta craata a short dalay.
DJNZ 005C        005C  10 FE     Oecramunt B 16 times.
XOR A            005E  AF        Zera A.
SET 7,A          005F  CB FF     SET tha highest bit of the eccumulater to tum an the LH display.
ADD A,(HL)       0061  86        ADD tha byte painted te by tha HL pair, to the accumulator.
OUT (02),A       0062  D3 02     Out put ta tha latch.
LD B,10          0064  06 10     Laad B with 10.
DJNZ 0066        0066  10 FE     Oecremant B 16 times.
DEC C            0068  0D        Decrement C.
JR NZ,0057       0069  20 EC     Jump if C nat zera.
JR 003F          006B  18 D2     Jump to start of BLINKING ROUTINE.
LD A,(HL)        006D  7E        Load A with date byta pointad to by HL.
CP 3F            006E  FE 3F     Campara with 3F to see if LH displey is zero.
JR NZ,0074       0070  20 02     Jump if nat zaro.
JR 0015          0072  18 A1     Jump ta start af pragram if zaro.
DEC HL           0074  2B        Oxcrxmxnt playar A pointer.
JR 0015          0075  18 9E     Jump ta start af program.
BIT 1,C          0077  CB 49     TEST bit 1 of tha C register.
JR NZ,002D       0079  20 D2     Jump if bit 1 la SET. Incrément te next instruction if not set.
SET 1,C          007B  CD C9     SET bit 1 of the C rxgister.
INC HL           007D  23        Incremant player A pointer.
LD A,(HL)        007E  7E        Laad the data byta inte tha accumulator.
CP 67            007F  FE 67     Compara the accumulator with 67.
JR NZ,00AE       0081  20 2B     Jump if tha twa are not the sama. Ga ta neat instructien if the seme.
LD C,10          0083  0E 10     The naxt 25 instructions produca a multiplexing affect on the
LD A,(DE)        0085  1A        twa displays se that the LH display tums on and off in a
OUT (02),A       0086  D3 02     9LINKING pattarn.
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            00A1  AF
OUT (02),A       00A3  D3 02
LD B,10          00A5  06 10
DJNZ 00A7        00A7  10 FE
DEC C            00A9  0D
JR NZ,009D       00AA  20 EF
JR 0083          00AC  18 D5
LD A,(DE)        00AE  1A        Laad the accumulater with the value pointed ta by DE.
CP 3F            00AF  FE 3F     Compare the accumulater with 3F to see if the RH displxy is zero.
JR NZ,00B6       00D1  20 03     Jump if player B Is zera, increment te naxt instruction if not zera.
JP 002D          00B3  C3 2D 00  Jump to atert ef progrem.
DEC DE           00B6  1B        Decrement player B pointer.
JP 002D          00B7  C3 2D 00  Jump ta atart af pragram.
```

loop in which the value for each display is looked after by a seperate 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 goveraned 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.

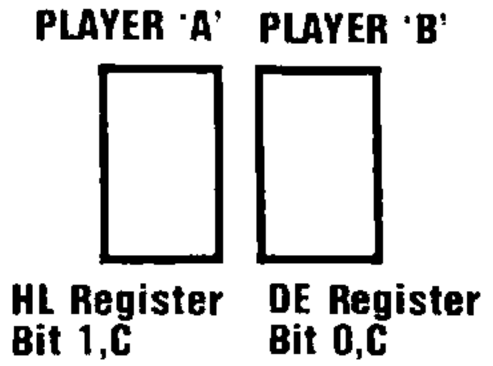
In our program, the register in charge of the table is incremented only after a multiplexing operation (which may involve a number of passes of a loop).

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 end to be run at page zero, it can be shifted to any othar location by simply changing all the absolute address values.

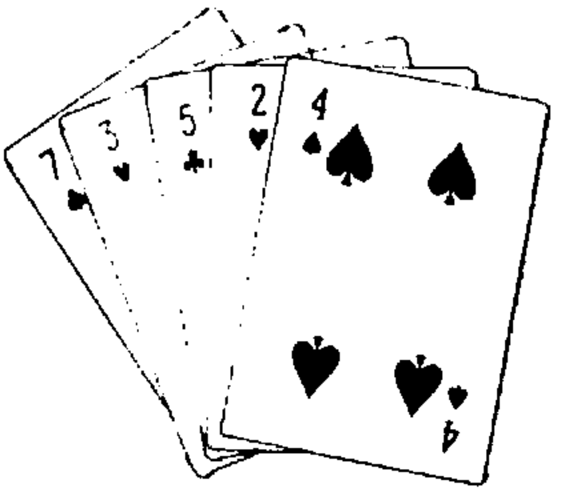


The diagram shows the two displays and their associated register pair. The Debounce is done in register 'C'.

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 inta account but it would be a simple matter to make it do so.

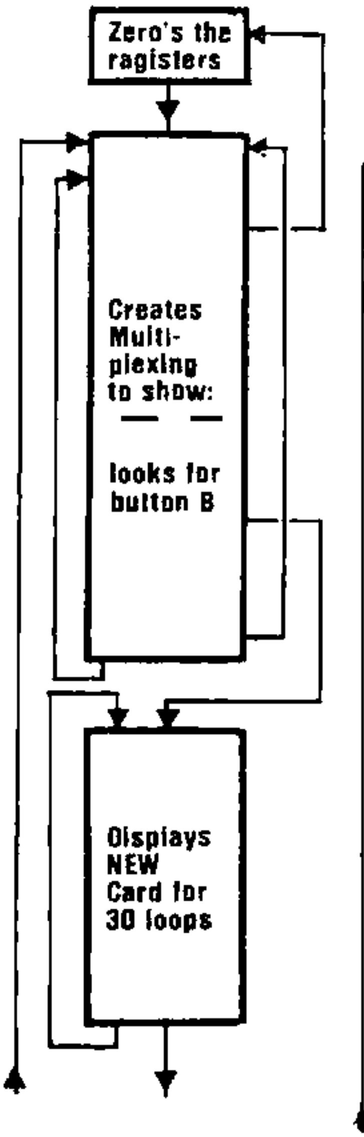
Here's the program: Type it on the TEC, hold it in the non-volatile RAM and play it on the Microcomp.

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

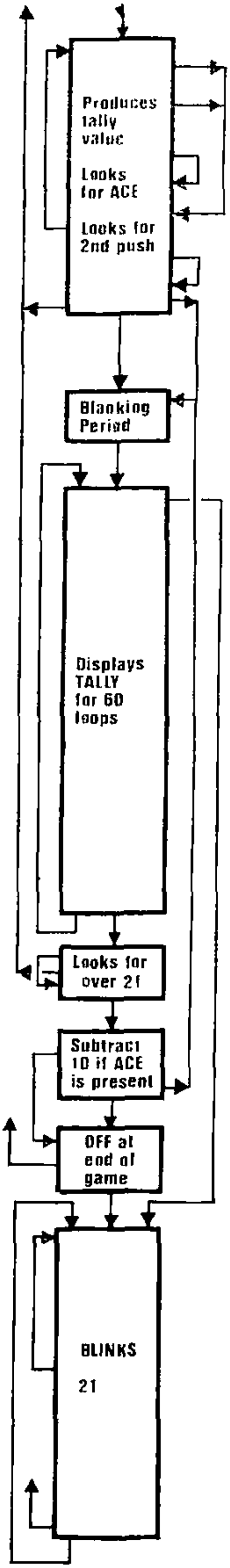
```
3F   0
06   1
5B   2
4F   3
66   4
6D   5
7D   6
07   7
7F   8
67   9
```

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
```


Zero's the ragisters
Creates Multi-plexing to show:
— —
looks for button B
Displays NEW Card for 30 loops

```
XOR A            0000  AF           Zere the Accumulater.
LD I,A           0001  ED 47        The I register must be londed via A. I reg. detacts 2nd push of button.
LD E,A           0003  5F           Zsre E. Rog E is our tally registar to detect '21' etc
LD C,A           0004  4F           Zero C  Reg C is our TEST register fer ACE detectien.
LD IX 0110       0005  DD 21 10 01  Load IX with stert of OISPLAY TABLE.
IN A,(01)        0009  DB 01        Button B must not be pressed when micre passes this point etherwise
CP 40            000B  FE 4a        pregram will jump to start of routine. This preuent cheating if
JR Z,0000        000D  28 F1        the button is kept prsssad
LD IY,0113       000F  FD 21 13 01  Lead IY with stert of table for displeying ualue of card.
LD H,0D          0013  26 0D        H caunts the number of greups of bytes in the table  There are 0D
LD A,(IX + 00)   0015  DD 7E 00     Lnad the accumulator with the first byte in the teble.        groups.
OUT (02),A       0018  D3 02        Output this value to the eutput latch.
LD B,08          001A  06 08        Load B with a value te produce a short delay.
DJNZ 001C        001C  10 FE        Create 8 loops of dacrementing register B.
XOR A            001E  AF           Zero the sccumulator before advancing to the next two operations.
SET 7,A          001F  CB FF        SET the highest BIT in the acc. so thst the LH display will illuminate.
ADD A,(IX + 01)  0021  DD 86 01     ADD the value of the second hyte in the table to the accumulator
OUT (02),A       0024  D3 02        Output the result to the letch. The LH display will illuminete.
IN A,(01)        0026  DB 01        Input tho value on the switchss ta the accumuletar
CP 40            0028  FE 40        Compare the sccumulator with '40'.
JR Z,003B        002A  18 0F        Jump if the sccumulater is squal to 40.
LD B,04          002C  06 04        Load B with 04 ready for a shert delay.
DJNZ 002E        002E  10 FE        Create 4 leeps of dacramenting reg B to displsy ths LH digit.
INC IY           0030  FD 23        Increment the IY registsr 3 times so that it looks at the start of the next
INC IY           0032  FD 23        group  This register is our random number gensrster and increments
INC IY           0034  FD 23        censtantly, while the displeys sre displaying.
DEC H            0036  25           Registsr H will detect the end of the byte table.
JR NZ,0015       0037  20 DC        Jump to displaying RH then LH digit, if H is net zere.
JR 000F          0039  18 D4        When H is zere, IY end IX registar go te start af table.
LD D,30          003B  16 30        D will gousm the length of tims for displaying the random number.
LD A,(IY + 00)   003D  FD 7E 00     Ths accumulater is loaded with the displsy velue for the random No.
OUT (02),A       0040  D3 02        This vslus is outputted to port 02.
LD B,20          0042  06 20        The RH displey will illuminate for a delay dstermined by the value of B.
DJNZ 0044        0044  10 FE
XOR A            0046  AF           The accumulator is zeraed ready for the next tws instructions.
SET 7,A          0047  CB FF        Bit 7 is SET to turn on the LH display.
ADD,(IY + 01)    0049  FD 86 01     The vslue of the secend byte in the group is added to the accumulator
OUT (02),A       004C  D3 02        snd eutputted to pert 02.
LD B,20          004E  06 20        The LH display is illuminated for a period of time as determined by the
DJNZ 0050        0050  10 FE        value of B.
DEC D            0052  15           D is decremented by ene snd the program loops agsin.
JR NZ,003D       0053  20 E8        When D is zero, tha micro aduances to the next instruction.
```



```
LD A,(IY + 02)  0055  FD 7E 02  Load A with the 3rd byte in the group. We know the byte must have a
INC E           0058  1C        value of one or greater and so we can safely INCrement E
BIT 1,E         0059  CB 4B     Register E is our TALLY register. We require it to add the values of the
JR Z,0066       005B  28 09     cards and hold the result in decimal form. The problem comes when
BIT 3,E         005D  CB 5B     you add one to 9. The register will show 0A. We must convert 0A to ten.
JR Z,0066       005F  28 05     This can be done by a DAA instruction or by software. We have opted
LD B,06         0061  06 06     for software. We detect 0A via bit 1 and 3 being HIGH and then
INC E           0063  1C        increment the E register 6 times. Each time the tally register is
DJNZ 0063       0064  10 FD     incremented (apart from the decimal adjusting loop), the accumulator
DEC A           0066  3D        is decremented and when the accumulator is zero, the program
JR NZ,0058      0067  20 EF     advances
LD A,(IY + 00)  0069  FD 7E 00  Load the accumulator with the first byte of the group
CP 77           006C  FE 77     Compare 77 with the accumulator. We are looking for an ACE
JR NZ,0072      006E  20 02     If the accumulator is not 77, the micro will jump to LD A,I. If the
SET 7,C         0070  CB F9     accumulator is 77, the program will advance to the next instruction
LD A,I          0072  ED 57     and SET bit 7 of the C register
INC A           0074  3C        The I register counts the number of presses of the B button. We are
LD I,A          0075  ED 47     looking for 2 or more presses so that the tally can be displayed. This is
CP 02           0077  FE 02     the advantage of using the CARRY command
JR NC,007D      0079  30 02     The micro jumps when I is 2 or MORE
JR 0009         007B  18 8C     Jump to start of program (button B has been pressed once.)
LD D,60         007D  16 60     Register D produces the time for the tally to appear
XOR A           007F  AF        Blank the display
OUT (02),A      0080  D3 02     Output to latch
LD B,FF         0082  06 FF     Load B with maximum delay value
DJNZ 0084       0084  10 FE     Perform FF loops of decrementing register B
LD HL,0100      0086  21 00 01  Load HL with start of display values
LD A,E          0089  7B        Load the tally register into the accumulator
CP 21           008A  FE 21     Compare 21 with the accumulator
JR Z,00D4       008C  28 46     If accumulator is 21, the micro jumps to 'BLINKING 21'
AND 0F          008E  E6 0F     If not 21, remove high nibble by ANDing with 0F
ADD A,L         0090  85        L contains '00' from address above. ADD 00 to accumulator
LD L,A          0091  6F        Load the result back into L so that the micro looks at one of the
LD A,(HL)       0092  7E        addresses of the table. Load the value it finds into A
OUT (02),A      0093  D3 02     Output the byte to the latch
LD B,10         0095  06 10     Load B with a low value
DJNZ 0097       0097  10 FE     Create 16 loops of decrementing register B
LD A,E          0099  7B        Load the tally register into the accumulator
RRA             009A  1F        Rotate the accumulator right, effectively bringing the 4 bits of the
RRA             009B  1F        HIGH nibble to occupy the 4 lower places.
RRA             009C  1F
RRA             009D  1F
AND 0F          009E  E6 0F     Remove the 4 high bits by ANDing with 0F
LD HL,0100      00A0  21 00 01  Load the HL register with start of display table
ADD A,L         00A3  85        ADD L to accumulator to create a new value for L so that we look at one
LD L,A          00A4  6F        of the addresses in the table
LD A,(HL)       00A5  7E        Load the byte from the table into the accumulator
SET 7,A         00A6  CB FF     Set bit 7 of the accumulator so that the LH display turns on
OUT (02),A      00A8  D3 02     Output this value to the latch
LD B,08         00AA  06 08     Load B with a short delay value
DJNZ 00AC       00AC  10 FE     Create 8 loops of decrementing register B
DEC D           00AE  15        Decrement D and go to start of multiplexing loop. When D is zero,
JR NZ,0086      00AF  20 D5     increment to next instruction in program
LD A,E          00B1  7B        Load the tally register into the accumulator
CP 22           00B2  FE 22     Compare with 22
JR NC,00B9      00B4  30 03     If tally is 22 or MORE, increment to next instruction.
JP 0009         00B6  C3 09 00  Jump to start of program. If tally is less than 22, jump to BIT 7,C.
BIT 7,C         00B9  CB 79     Test bit 7 of the C register to see if an ACE has been dealt
JR Z,00C4       00BB  28 07     Jump if no ACE. Increment if an ACE is present
SUB 10          00BD  D6 10     Subtract ten from tally, making ACE equal to ONE
LD E,A          00BF  5F        Load the new tally into the tally register
RES 7,C         00C0  CB B9     Reset bit 7 to show ACE has been turned into ONE
JR 007D         00C2  18 B9     Jump to displaying new tally
XOR A           00C4  AF        Zero the accumulator
OUT (02),A      00C5  D3 02     Output to latch for a delay period equal to 4 DJNZ's (with B = FF) to
LD B,FF         00C7  06 FF     indicate END OF GAME
DJNZ 00C9       00C9  10 FE
DJNZ 00CB       00CB  10 FE
DJNZ 00CD       00CD  10 FE
DJNZ 00CF       00CF  10 FE
JP 0000         00D1  C3 00 00  Jump to start and re-load all registers
LD C,10         00D4  0E 10     Load C with 10 for 16 loops of multiplexing '1' and '2'
LD A,06         00D6  3E 06     Load A with 06 to create '1' on RH display
OUT (02),A      00D8  D3 02     Output this to latch
LD B,10         00DA  06 10     Create short delay
DJNZ 00DC       00DC  10 FE     Decrement B to zero
LD A,DB         00DE  3E DB     Load the accumulator with DB to create '2' in LH display
OUT (02),A      00E0  D3 02     Output to latch.
LD B,10         00E2  06 10     Create short delay
DJNZ 00E4       00E4  10 FE     Decrement B to zero.
DEC C           00E6  0D        Decrement C and if not zero, jump to start of multiplexing the displays.
JR NZ,00D6      00E7  20 ED
XOR A           00E9  AF        Zero A
OUR (02),A      00EA  D3 02     Output to latch to turn off both displays
LD B,FF         00EC  06 FF     Load B with FF to produce a short delay for the OFF time
DJNZ 00EE       00EE  10 FE     The only way of jumping out of 'BLINKING 21' is to push button B or
IN A,(01)       00F0  DB 01     reset the computer. The program inputs from the set of buttons and if
CP 40           00F2  FE 40     B is pressed, the program jumps to 0000. Otherwise the program
JP Z,0000       00F4  CA 00 00  keeps looping
JR 00D4         00F7  18 DB
```

Before we continue our dissection 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.

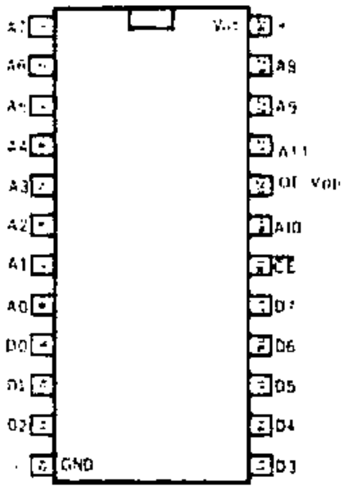
Burning a program is only done after you are thoroughly satisfied with its performance, as it is very difficult (if not impossible) to change the program . once it is burnt. For this reason it is best to get the program up and running via a medium which can be easily altered, as a program quite often has to go through lots of changes and modifications before you ere 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 writtan for both the lower half and upper helf 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 progrem 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, oceipied 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 seme 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 8URNER 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 diegram explaining what is happening at various locations.

A description of the progrem 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 tn 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 replenishad 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 nnly 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 deta can ba 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 SCRATCHPAO and STACK.

The scratchpad or BUFFER zone needs only a few bytes where such data as displays velues are kept. This frees ragisters for carrying out program commands.

The other area is STACK and this is whera 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 kaep removing bytes at the same rate as they are added so that the stack does not grow too large.

The othar peculiar featura about the stack is the access you have to its contents. It is a LAST-ON FIRST-OFF arrangement end 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 tha programs. This severly limits programming as the working registers are held-up as memory cells

Without a stack, programs have to be designed differantly 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 arnund 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 mamory. One is programmed by the manufacturer and cannot be changed, the other is erasable memory and can be programmod 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 nf 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 e 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 wo want to fully explain the lower half and gat you aquainted with tha concepts.

One question we have been asked is why the Micrncomp has only 11 address lines whereas the 2732 requires 12!

The answer is we are creating the 12th address line via the jumper lead. Whan 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 oparation, 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 tha 2k boundary, place the jumpar on A11).

Because of our arrangement, the 2732 can be cansidered as two separate 2k blocks, each of which is equivalent to a 2716 EPROM. In fact you can use 2716's without tha need for any modifications.

Each 2k block is addressed in hexadecimal notation. It starts at **0000** and goes to **07FF**. Tha next 2k starts at **0800** and finishes at **0FFF**. There are 8 pagas 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 deservas its reputation for being the best language for teaching computers to beginners

And true enough, Basic has eneblad 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 represant 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 impressad with the results.

We are not rubbishing Basic but let's say it is completely removad from the field we are covering. Machine code is is up to 10,000 times fast and takes up to 500 times less mamory. 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 instructiens for you to be able to complete anything from a sequencer to a robot.

HOW TO START PROGRAMMING

All programs start with an idaa. The idea mey 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 papar and jotting down notes.

Start with sketches, scribbles and bits of data.

Put a date en tha sheet and think up a name for the project. Names and labals 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 togethar much better after you have cleared it first by writing down all the preliminaries.

Don't be efraid to use paper. It will take about 6-10 pages to produce one page of finished work.

At first the best idee is to use perts 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. Thay rarely work first time.

But that's the wonderful part about programming. The micro picks up your mistakes end feils to operate.

When this happens, you can spend hours trouble-shooting the fault. Tha 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 mistaka.

If you don't have anyona in this category, you will have to work through it yoursalf.

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 displey still fails to light up, you will have to investigate each of the steps and instructions very carefully. Work backwerds 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 eddress 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 ba inserted as you nead to count tha 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 thet one byte must be counted when working out the displacement values.

If the display still fails to illuminate, you can createa your own display value by loading the accumulator and outputting it to the displey and then adding a HALT instruction. This is a last resort! and lets you know how far the progrem 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 et address 0000. This is whare 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 eppropriate 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.

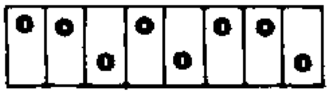
Tables should be placed well away from the program so that you don't run out of room. When everything works perfectly, they can be moved up and the pointers changed accordingly.

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 neetening it up tekes 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 en input switch ON, the corresponding LED, in the row of 8 LEDs, will be illuminated.

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

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

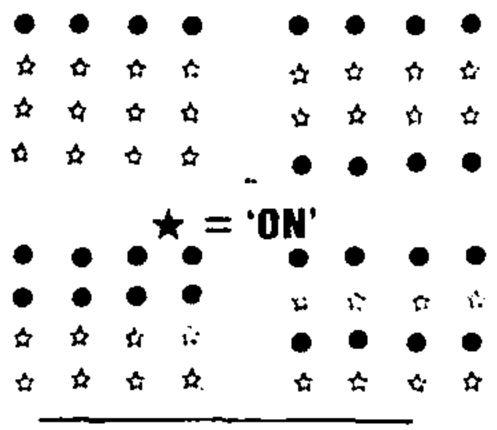
This means it is rapidly looping around the program and will update the displays as soon as the input values are changed.

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

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

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

Create these effects on the 4x4 matrix:



FROM INPUT PORT TO 8LEDS

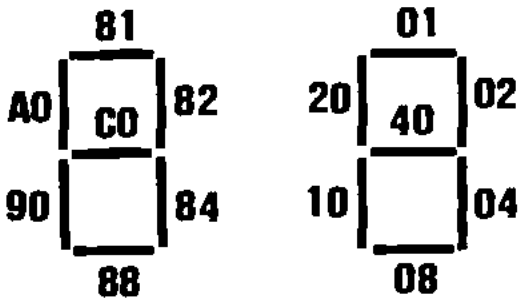
```
IN A,(01)      0290   DB 01   Looks at input switches and places the value in the accumulator
OUT (02),A     0292   D3 02   Outputs accumulater to the latch
JR 0290        0294   18 FA   Jumps to start af program.
```

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.

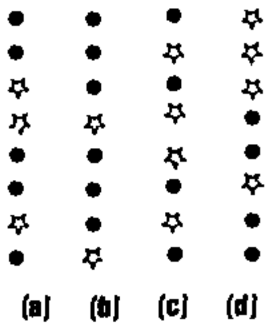
Button 'A' is connected to **80** and will make the figures jump from one display to the other.

3. The 4x4 matrix has been wired so that each column is turned on by a LDW value. These values are: 01, 02, 04, and 08. This will causa 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.

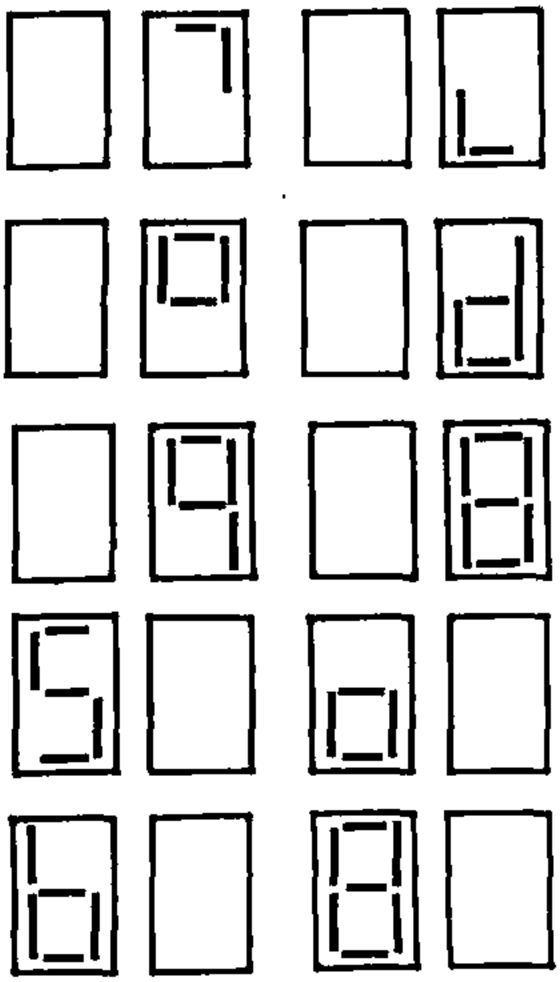
There are some limitetions as to what combinations of LEDs can ba turned on and thia is something you must be aware of.

Experiments:

Create these effects by using the input switches:

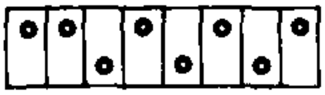


Create these on the 7-segments displays:



INCREMENT via BUTTON A

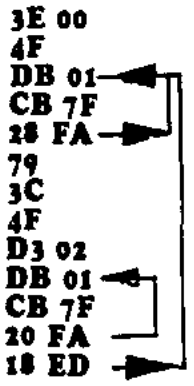
This program at 02A0 increments the display each time button A is pressed.



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.

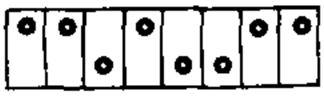


```
3E 00
4F
DB 01
CB 7F
28 FA
79
3C
4F
D3 02
DB 01
CB 7F
20 FA
18 ED
```

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 progrem is locatad 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.

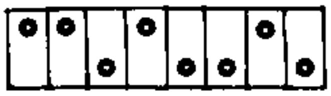
```
LD A,00        02C0   3E 00
INC A          02C2   3C
OUT (02),A     02C3   D3 02
DJNZ 02C5      02C5   10 FE
DJNZ 02C7      02C7   10FE
DJNZ 02C9      02C9   10 FE
JR 02C2        02CB   18 F5
```

```
LD A,00        02A0   3E 00   Load the accumulator with zaro.
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 tha accumulatar to sae if buttan 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 tho eccumulator.
INC A          02AA   3C      Incremant tho occumulator
LD C,A         02AB   4F      Load the answer into tho TALLY register 'C'.
OUT (02),A     02AC   D3 02   Output the accumulator to the disploys.
IN A,(01)      02AE   DB 01   Input the switches to the accumulster.
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 ta 2A3.
```

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 valua.

AUTO INCREMENT (variable)

This program is located at 02D0 and the speed with which the computer



02D0

completes ona cycle depends on the satting of the input switches.

```
LD D,01        02D0      16 01     Loed the TALLY register with 01.
IN A,(01)      02D2      DB 01     Input the switch velue to the eccumulator.
LD C,A         02D4      4F        Load the accumulator into 'C' for the delay veluo.
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 ragister 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 tha screen, so that the accumulator can ba used for other things.

'C' is the delay ragister and it is dacremented very similar to a **DJNZ** statemant, where FF produces tha longest delay and 01 the shortest delay.

This is not quite correct, however, as you will find out for yourself.

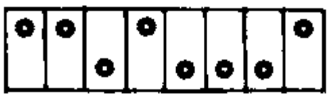
Load the value **01** end compare it with **00.** **00** is a much longer delay and it appeers to be as long as **FF!** In fact this is the case! The longest deley is produced when a register is loaded with **00** since the first operation to be performed on the register is to decrement it. The result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

The program is designed to start with en 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 tha velues 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

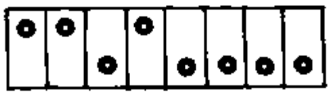
counter. The best effect of decrementing can be seen on tha 8 LEDs. Adjust the speed control to viaw the effect in slow motion.

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 OECREMENT (variable)

This routine is located at 02F0 and decrementes the display when button A is pressed. It has a fixetl rate of decrementing and is not variable.

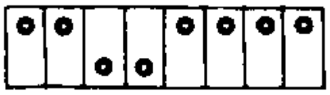


02F0

```
LD E,FF       02F0   1E FF   Load the COUNT HOLD register with FF.
LD A,E        02F2   7B      Load the Cuunt Hold registar into tho accumulator.
OUT (02),A    02F3   D3 02   Output the eccumulator to the latch.
DJNZ 02F5     02F5   10 FE   Create a short delay with the B ragister.
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 seo if A is pressed.
JR Z,02F2     02FB   28 F5   Jump to 02F2if it is not pressed. Go to next line if pressed
DEC E         02FD   1D      Decrement registar E.
JR 02F2       02FE   18 F2   Jump to 02F2.
```

4x4 DISPLAY

As the name suggests, the program at **0300** is designad for the 4x4 DISPLAY. It



0300

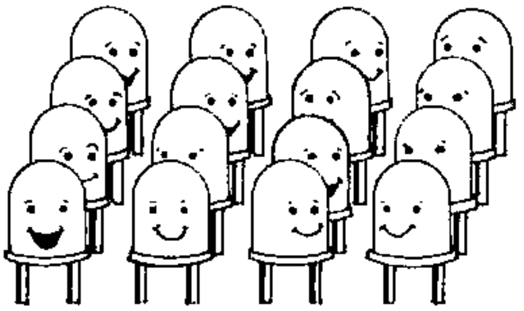
will produce almost no interpretable effects on either of the other displays.

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 en enormous array of LEDs or globes and obviously tha ultimate is to produce a video screen with coloured globes to duplicate a TV. But the cost of this kind of venture is anormous as the parts alone would cost a fortune and the tima taken to wire it up would ba too much for an individual constructor.

That's why we have concentrated on a manegeable module.

One of the decisions you hava to make when outputting to LEDs, is the method of turning them ON. One is to connect each output of a latch diractly to a LED. The other is to multiplex the display and scan it. The multiplex method uses tha least number of chips and is obviously the cheeper.

The relative merits of each will be covered in future articles and for the moment we will study the effects which cen be produced with a display connected in MULTIPLEX mode.



The program at **0300** is an DUTPUT ROUTINE in which a value is loaded from a table into the accumulator and outputted to the display. The display remains illuminated for a deley period and then the next byte is picked up from the table. This is done until all tha bytes have beeit used.

When the end of the table is reeched, the program starts again. This is rapeated for 8 loops and then the micro advances to the second part. This is identicel to the first except for the byte table. It has antiraly 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 end the first pattern is outputted.

The speed of presenting a pattern is controlled by the clock end the inbuilt delay value. The delay is fixed but the clock cen be adjusted to slow-down or speed-up the effect.

```
LD B,08        0300  06 08     B is the COUNT REGISTER for the number of loops in the first pngram
LD HL,0338     0302  21 38 03  Load HL with the address of the start of the BYTE TABLE
LD C,18        0305  0E 18     Load C with the number of bytes for the program (There are 24 bytes )
DEC C          0307  0D        Decremant the number of bytes remaining in the table to detect the end nf table.
JR Z,0318      0308  28 0E     If no bytes remain, decrement the numhor of loops and start program again
LD A,(HL)      030A  7E        Load the accumulator with the byte pointed to b the HL register pair
OUT (02),A     030B  D3 02     Output this value to port 2
INC HL         030D  23        Increment HL to point to the next byte in the table
LD DE,0880     030E  11 80 00  Load DE with a short delay value.
DEC DE         0311  1B        Decrement DE
LD A,D         0312  7A        Load D into A.
OR E           0313  B3        Logically DR the accumulator with E to see when BOTH D and E arezero
JR NZ,0311     0314  20 FB     Jump to 0311 if the answer is NOT ZERO
JR 0307        0316  18 EF     Jump to DEC C and repeat for the second byte in the table
DJNZ 0302      0318  10 E8     Decrement the number of luups and start the byte table again
LD B,08        031A  06 08     Load B with 8 fnr the second part of tho program
LD HL,0350     031C  21 50 03
LD C,20        031F  0E 20
DEC C          0321  0D
JR Z,0332      0322  28 0E
LD A,(HL)      0324  7E
OUT (02),A     0325  D3 02
INC HL         0327  23
LD DE,0080     0328  11 80 00
DEC DE         032B  1B
LD A,D         032C  7A
OR E           032D  B3
JR NZ, 032B    032E  20 FB
JR 0321        0330  18 EF
DJNZ 031C      0332  10 E8
JR 0300        0334  18 CA
```

This part is identical with that above except the byte table is longer and located at a different address. When 8 loops of this part have been executad 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 eccess 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 tho hex value required to illuminete an individual LED:

71	72	74	78
B1	B2	B4	B8
D1	D2	D4	D8
E1	E2	E4	E8

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

The following diagrams give you tho values required tn turn DN one or more LEDs:

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

LEGEND:

○ = ON.
● = OFF.

```
○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○
```
0F

```
○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○
● ● ● ●
```
1F

```
○ ○ ○ ○
○ ○ ○ ○
● ● ● ●
○ ○ ○ ○
```
2F

```
○ ○ ○ ○
○ ○ ○ ○
● ● ● ●
● ● ● ●
```
3F

```
○ ○ ○ ○
● ● ● ●
○ ○ ○ ○
○ ○ ○ ○
```
4F

```
○ ○ ○ ○
● ● ● ●
○ ○ ○ ○
● ● ● ●
```
5F

```
○ ○ ○ ○
● ● ● ●
● ● ● ●
○ ○ ○ ○
```
6F

```
○ ○ ○ ○
● ● ● ●
● ● ● ●
● ● ● ●
```
7F

```
● ● ● ●
○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○
```
8F

```
● ● ● ●
○ ○ ○ ○
○ ○ ○ ○
● ● ● ●
```
9F

```
● ● ● ●
○ ○ ○ ○
● ● ● ●
○ ○ ○ ○
```
AF

```
● ● ● ●
○ ○ ○ ○
● ● ● ●
● ● ● ●
```
BF

```
● ● ● ●
● ● ● ●
○ ○ ○ ○
○ ○ ○ ○
```
CF

```
● ● ● ●
● ● ● ●
○ ○ ○ ○
● ● ● ●
```
DF

```
● ● ● ●
● ● ● ●
● ● ● ●
○ ○ ○ ○
```
EF

```
● ● ● ●
● ● ● ●
● ● ● ●
● ● ● ●
```
FF

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

To usa these values salect from the first 16 diagrams to give the row(s) and from the following 16 diagrams for tha column(s).

○ ● ● ●
○ ● ● ●
○ ● ● ●
○ ● ● ●

01

● ○ ● ●
● ○ ● ●
● ○ ● ●
● ○ ● ●

02

○ ○ ● ●
○ ○ ● ●
○ ○ ● ●
○ ○ ● ●

03

● ● ○ ●
● ● ○ ●
● ● ○ ●
● ● ○ ●

04

○ ● ○ ●
○ ● ○ ●
○ ● ○ ●
○ ● ○ ●

05

● ○ ○ ●
● ○ ○ ●
● ○ ○ ●
● ○ ○ ●

06

○ ○ ○ ●
○ ○ ○ ●
○ ○ ○ ●
○ ○ ○ ●

07

● ● ● ○
● ● ● ○
● ● ● ○
● ● ● ○

08

○ ● ● ○
○ ● ● ○
○ ● ● ○
○ ● ● ○

09

● ○ ● ○
● ○ ● ○
● ○ ● ○
● ○ ● ○

0A

○ ○ ● ○
○ ○ ● ○
○ ○ ● ○
○ ○ ● ○

0B

● ● ○ ○
● ● ○ ○
● ● ○ ○
● ● ○ ○

0C

○ ● ○ ○
○ ● ○ ○
○ ● ○ ○
○ ● ○ ○

0D

● ○ ○ ○
● ○ ○ ○
● ○ ○ ○
● ○ ○ ○

0E

○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○
○ ○ ○ ○

0F

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 illuminatad at tha same time.

Brightness can be improved by tuming 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%.

Wa 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 rete 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 spead of the execution of machine code instructions, some parts of the program must be slowed down so that humans can ba involved. This may be for the video aspect, so that the aye can see what is being outputted on a display or for tha 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-movemant for objects moving across the screen.

The dalays 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 ragister pair with a large numbar and gradually decrement it to zero.

By decramenting a single register, the maximum number of loops which can be executed is 256. Each loop may taka 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, tha time can be increased 256 times. The delay becomes more noticeable and will be about 2 seconds.

If we require longar delays we can add another register-pair and incraase 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 hare: When a micro is performing a very long delay, the entire computar tima is being taken up with a COUNT OOWN 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 saen 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      Seve the 'TALLY' in the I register (Not part of IX)
LD DE,FFFF     03F4   11 FF FF   Load OE with the moximum value
LD HL,FFFF     03F7   21 FF FF   Lood HL with the maximum value.
DEC HL         03FA   2B         Oecroment HL.
LD A,H         03FB   7C         Lead register H inte the accumulaler.
OR L           03FC   B5         Legicelly OR the aceumulator 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 net zero, jump te 03FA.
DEC DE         045D   1B         When HL (the inner leep) is zero, decrement DE.
LD A,D         045E   7A         Load register O into tha eccumulater.
OR E           045F   B3         Logicelly OR the eccumuleter with register E.
JP NZ,03F7     0460   C2 F7 03   If result is net zaro, JUMP te 03F7 and OEC HL!
LD A,I         0463   ED 57      When beth HL end OE are zero, time is UP!
OUT (02),A     0465   D3 02      Lead the TALLY register into A and output it.
INC A          0467   3C         Increment A.
LD I,A         0468   ED 47      Loed the new tally inte the TALLY register.
JP 03F4        046A   C3 F4 03   Lead the ragister pairs and start again!
```

When we usa two register pairs to create a very long tima delay, we do not place one pair after the other as this would only double the time dalay. We place them ON TOP of each other so that the effect is MULTIPLICATIDN. This means one pair is INSIDE the other and we say it is HIDDEN or NESTED. This arrangement gives rise to the term NESTED LDOP. This is what wa are creating in this section.

The simplest method of incraasing the delay is to add the instruction: **10 FE.** This will heve the effect of adding 256 cycles to the delay time. This is a **DJNZ** instruction end operates with the B register. The advantage of a **DJNZ** is it does not effect the accumulator. In the Microcomp we do not have any RAM and we cannot save the accumuletor via a PUSH oparation since we do not have any STACK. Thus it's an advantage not to alter the contents of the accumulator.

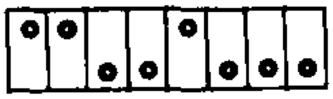
DJNZ loops are not nested loops but are additive and require tha **B** register to be zaro at the stert of the delay routine to create the longest delay. At the end of a **DJNZ** tha B register is zero and this is ideel for the next **DJNZ.**

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 CDUNTER. This is located at addrass **0370** and will show us how to



output numbers onto the display and how the INCremant oparation is parformed.

The main fact to ramamber with the program is the computer is NOT adding numbers. It is simply going through a tabla of velues and it is the values it fetches that create the incremants on the screen.

The table could be designed to produce letters or symbois and we would losa 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 LDDPING at all times ready to instantly pick up an input value.

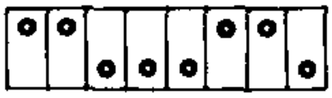
Bacause the program is running at high speed, we must include a DEBOUNCE feature to prevent more than ONE COUNT being registered when a button is pressed.

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.

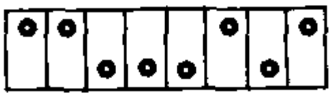


The main program for producing the letters on the display is located at **03A8** and the micro jumps to this address via the instruction **JR 03A8.** The main program is also usad by the **A -Z, 0 - F** counter and shows how the sama table and output program can be accessed by two different START-UP 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 wili be awere that certain combinations of hex values produca letters and numbers on the display.

Use the program at **0290** to produce the numbers 0 - 9 end letters A - F, by switching DN tha correct switches. Use the output display values on P68 to assist you in this. Add the value on the switches and compere with the table et **03C6.**

0 - 9 COUNTER

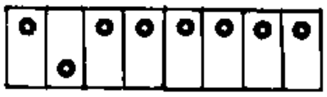
```
LD C,0A      0370   0E 0A      Register C is the counter for the BYTE TABLE There are ten bytes
LD DE,03DF   0372   11 DF 03   The DE register pair is loaded with the start-address of the byte tabls
IN A,(01)    0375   DB 01      The input latch is looked at and the value it holds is placed inte the accumulator.
BIT 7,A      0377   CB 7F      The only line (er 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 accumulatar
OUT (02),A   037D   D3 02      Output this byte to the diselay.
IN A,(01)    037F   DB 01      Look at the input part
BIT 7,A      0381   CB 7F      Test bit 7 of the accumulater
JR NZ,037F   0383   20 FA      Jump to address 037F if button A is prassed. When button is relessed, advance to next line.
DEC C        0385   0D         Decrement the SYTE 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 kapt 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.

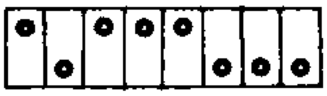
This is necessary to keep the displays illuminated while et the same time preventing tha program from incrementing the displays if a button is kapt pressad.

```
LD E,00        0400   1E 00      Registar E halds 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 tham.
LD HL,03E0     0405   21 E0 03   Load HL with the stert 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 pointad 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 produca the 10's value.
RRA            040E   1F         Shift the bits in the accumuletor 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 stert of the byte teble.
ADD A,L        0417   85         Add the value of L to the velue in the accumulator.
LD L,A         0418   6F         A new pointer value is created.
LD A,(HL)      0419   7E         Load the aceumulator with tha 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 displey.
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 tha accumuletor.
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 zoro, jump to 024A. If it prassed,increment to next instruction.
LD A,E         0424   7B         Load E into the accumuletor, ready for an INCremant operation.
INC A          0425   3C         Increment the accumulator.
DAA            0426   27         Decimal adjust the accumuletor. This means an A will ba changad 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 0421, 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 accumulater.
DEC A          042F   3D         Decrement the accumulator.
DAA            0430   27         Decimal adjust the accumulator. This will change a zero inte a 9.
LD E,A         0431   5F         Save the count value by leading 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 remeinder of tho program keeps both displays illumineted by looping from **0432** to **0456** while either of the buttons remeins pressed. As soon as the button is released, the program jumps back to **0402** and executes the top loop.

DICE

The DICE Program at0470 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 relieble way to get a random number is to introduce the human element.

This is what we have done in this progrem.

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 ragistars. 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 ragister 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 fest 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 gredually get to understand that pressing et the beginning of cycle would generate a low number end at the end of a cycle, a high number.

Owing to the unpredictability of the human reaction, an even spread of numbars 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 obteined, a number of flashes are created on the screen and then the accumulator is compared with the random number before jumping to the displey routine.

This routine is e very simple multiplexing routine in which three bytes are outputted for e 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-chack was not made, the sama numbar would appaar on the displeys due to a constant number of cycles occuring in the program for eech game.

```
LD D,0C          0470   16 0C      C is the byto 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 thn value of the first byte in the table.
OUT (02),A       0476   D3 02      The accumulator is outputted ta port 02
INC HL           0478   23         The byte table pointor is incremented
LD B,15          0479   06 15      Load B with 15, for a delav valun of 21 loops
LD C,06          047B   0E 06      Load C with 6, for tha dice values: 0-6
IN A,(01)        047D   DB 01      Input from tha input port to the accuroulator.
BIT 7,A          047F   CB 7F      Check to soe if button A has been prossed
JR NZ,048D       0481   20 0A      If pressed, jump eut of the dolay routine
DEC C            0483   0D         Decromant registor C
JR NZ,047D       0484   20 F7      If C is not zero, jump up. If C zero, advonce.
DJNZ 047B        0486   10 F3      Dacremeat B and if not zaro, jump up
DEC D            0488   15         Oecrement the byto table register O
JR Z,0470        0489   28 E5      Whon D is zero jump to start of pingram
JR 0475          048B   18 E8      If nat zero, continue OELAY ROUTINE
LD D,06          048D   16 06      Load O with 6 for six flashas of the display.
LD A,0F          048F   3E 0F      Load A to turn on the whole 4x4 display
OUT (02),A       0491   D3 01      Output to part 02
DJNZ 0493        0493   10 FE      Ragister B is decromonted to create a doluy
LD A,FF          0495   3E FF      Load A with n value to turn 4x4 OFF
OUT (02),A       0497   D3 02      Output to purt 02.
DJNZ 0499        0499   10 FE      Create a short delay with register B
DEC D            049B   15         Oecrement the flesh-count registor
JR NZ,048F       049C   20 F1      Loop for 6 flashos
LD D,80          049E   16 80      Load O for 80 loops for multiplexing routine
LD A,C           04A0   79         Load aur random numbor 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 tha accumulator with 1
JP Z,045F        04A6   CA F5 04   If ibe accumulator is 1 jump to multiplex routine.
LD HL,04E3       04A9   21 E3 04   Load HL with start address for disploying '2'
CP 02            04AC   FE 02      Compare the accumulatar with 2
JP Z,045F        04AE   CA F5 04   If accumuletor is 2, jump to multiplex routine.
LD HL,04E6       04B1   21 E6 04   Load HL with stert-address for displaying '3'
CP 03            04B4   FE 03      Compara accumulator with 3
JP Z, 045F       04B6   CA F5 04   If accumulatnr is 3, jump tn multiplex routine.
LD HL,04E9       04B9   21 E9 04   Lnad HL with stort-address for displaying '4'
CP 04            04BC   FE 04      Compara thu accumulatur with 4
JP Z,045F        04BE   CA F5 04   If accumulator is 4, jump to multiplex routine.
LD HL,04EC       04C1   21 EC 04   Laad 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 reutina.
LD HL,04EF       04C9   21 EF 04   Load HL with start-addrass for displaying 6.
CP 06            04CC   FE 06      Compare accumulator with S
JP Z,04F5        04CE   CA F5 04   Jump tn multiplex rautine is accum is 6
```

A jump value muat be found and the micro jumps to the multiplexing routino balow and produces a display on the 4x4 that is similar to tho spots on tha faca of a dico. Tha routine runs for 80 loops, makes sure butten A is nat pressad, than jumps te tha start of tha OICE program

At 04E0:

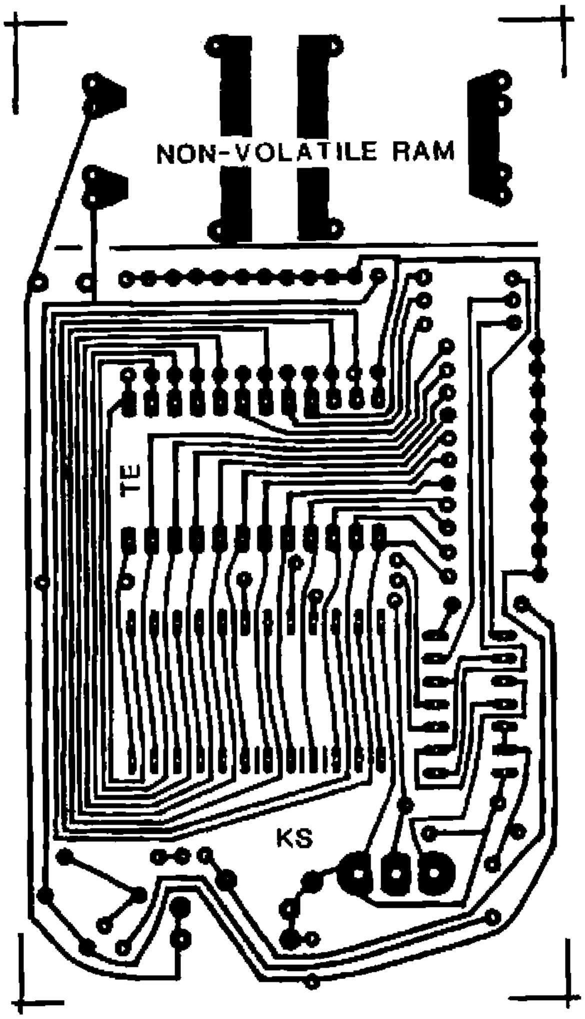
B4	D2	72	52	52	52
00	00	B4	00	B4	54
00	78	D8	58	58	58

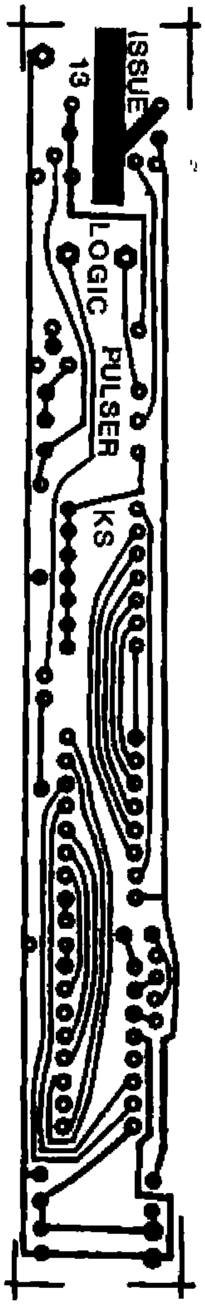
```
LD A,(HL)        04F5   7E         Load A with the vnlue pointod to by HL
OUT (02),A       04F6   D3 02      Output the value to port 02
LD B,0A          04F8   06 0A      Load 8 with a short delay value.
DJNZ 04FA        04FA   10 FE      Create a short delay with rogister 8.
INC HL           04FC   23         Peint ta 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 8 with a shart delay value.
DJNZ 0502        0502   10 FE      Crente a short delay with register 8
INC HL           0504   23         Ioc HL to look at next address
LD A,(HL)        0505   7E         Lond value pointed to by HL in the accumulator.
OUT (02),A       0506   D3 02      Output to port 02
LD B,0A          0508   06 0A      Lead register B with a short delay value
DJNZ 050A        050A   10 FF      Decremant registar B to zero.
DEC HL           050C   2B         Oec HL to look at start of display tabln
DEC HL           050D   2B
DEC D            050E   15         Decrement multiplex routine loep counter.
JR NZ, 04F5      050F   20 E4      Loop again if D is nat zero.
XOR A            0511   AF         Zero the accumulator and output to port 02 to
OUT (02),A       0512   D3 02      blenk tha display.
IN A,(01)        0514   DB 01      Look at the output port to see if button A is NOT
BIT 7,A          0516   CB 7F      pressed before re-starting the DICE program.
JR NZ,0514       0518   20 FA      Loop is A is preseed.
JP 0470          051A   C3 70 04   Jump to start of OICE program.
```

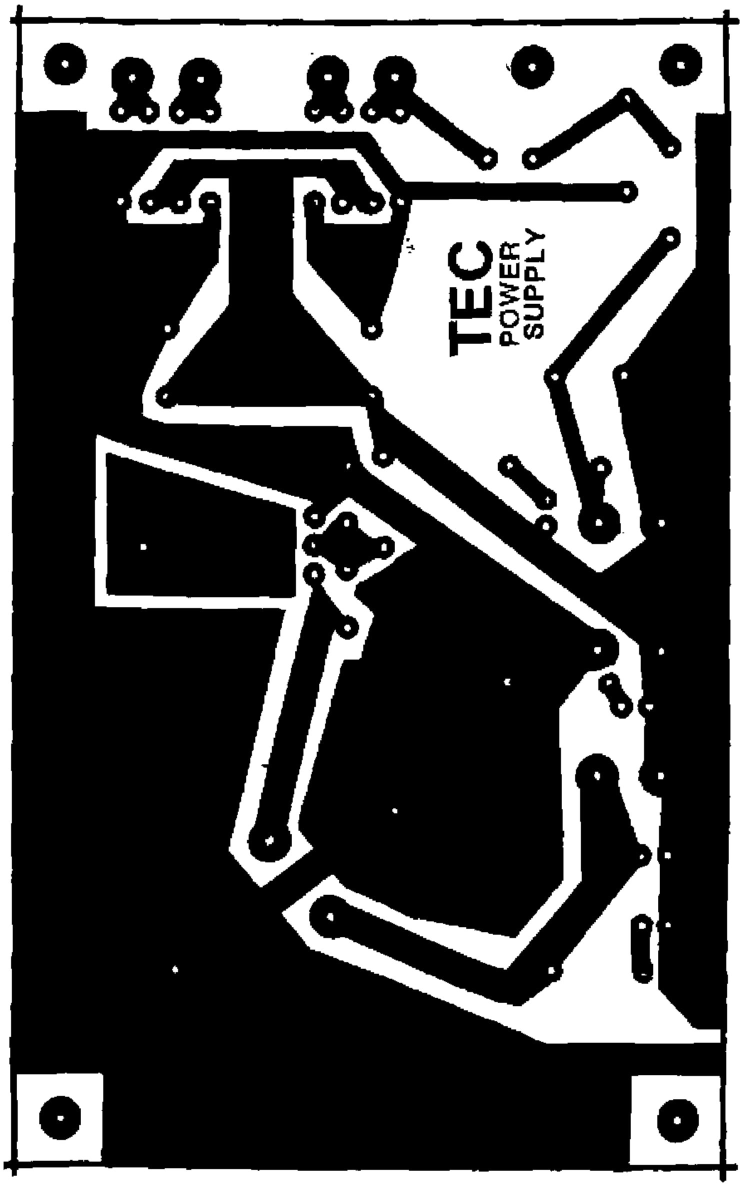
At 04D3:

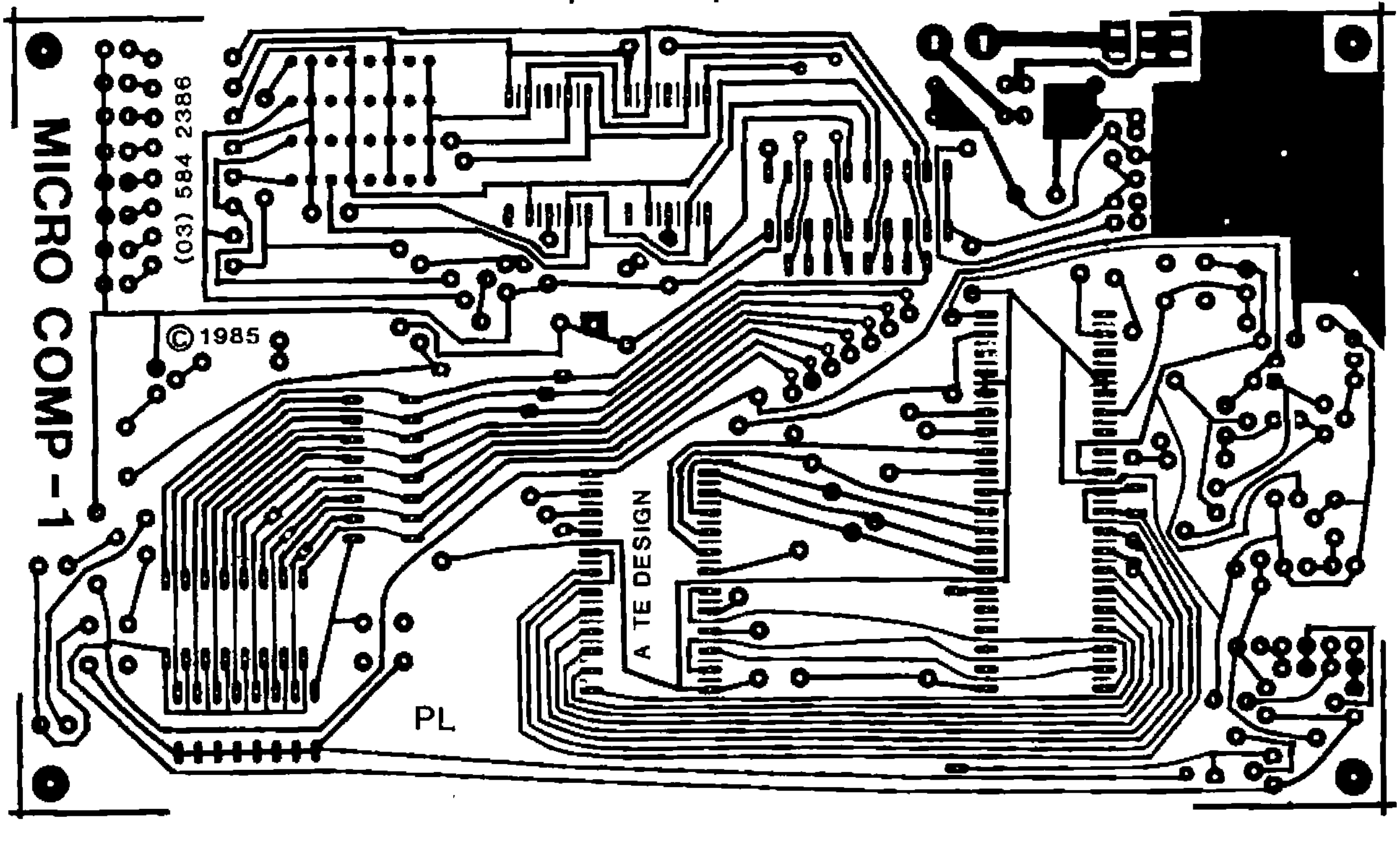
71	E8
72	E4
74	E2
78	E1
B8	D1
D8	B1

PC ARTWORK:

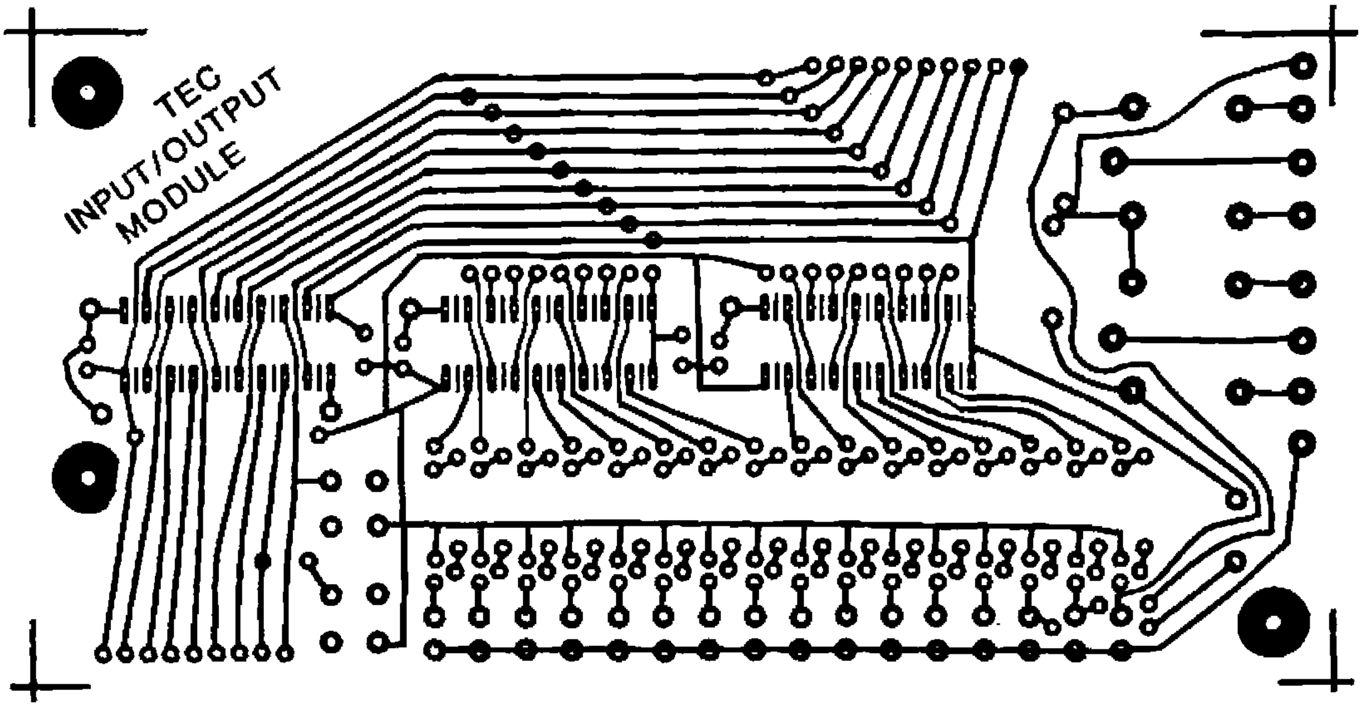

NON-VOLATILE RAM
TE
KS

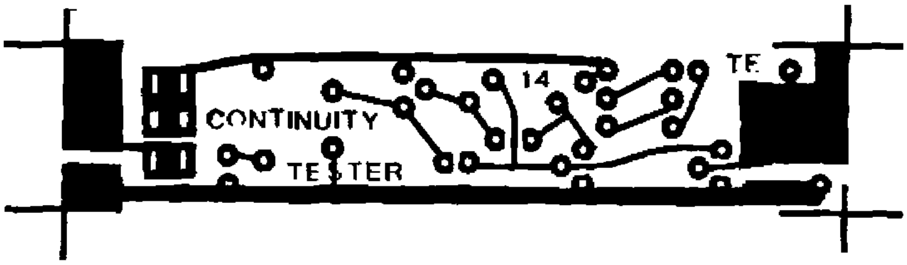

ISSUE 13
LOGIC PULSER
KS

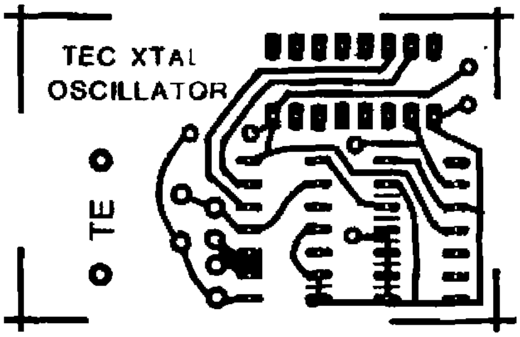

TEC POWER SUPPLY

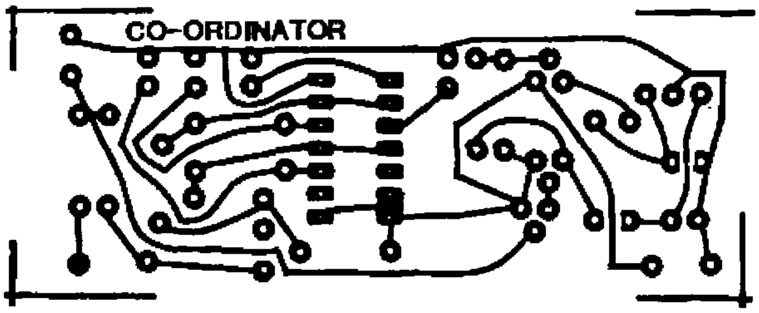

MICRO COMP - 1
(03) 584 2386
© 1985
A TE DESIGN
PL

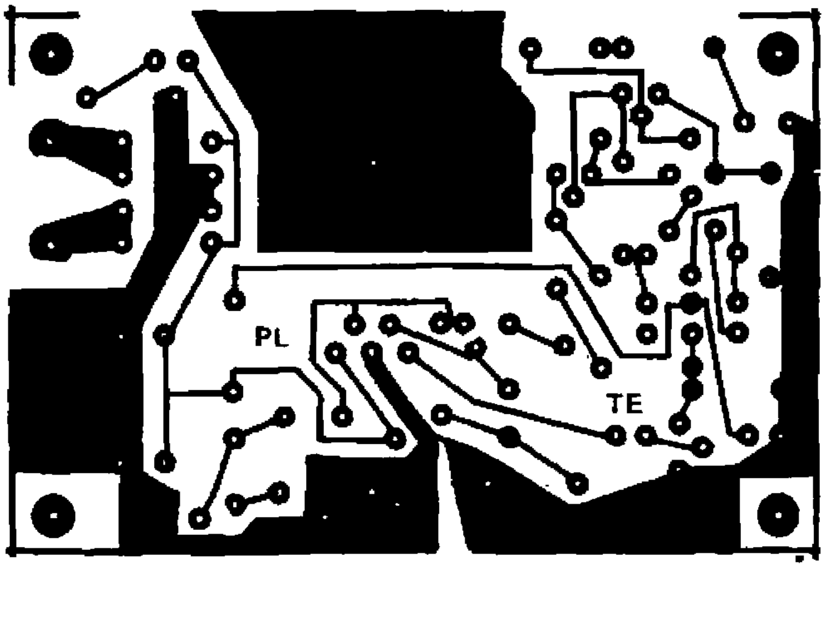
PC ARTWORK:


TEC
INPUT/OUTPUT
MODULE


CONTINUITY
TESTER
14
TE


TEC XTAL
OSCILLATOR
TE


CO-ORDINATOR


PL
TE

TALKING ELECTRONICS®

$3.00
$4.60NZ

Issue No 15

May '89
Reg by Aust. Post.
Publication No: VBP 4256

☆CAR ALARM
☆ULTIMA
(A 1km TRANSMITTER)
☆SPEECH MODULE

☆ DAT BOARD

☆ FM RADIO

☆ ORGAN

TEC TALK

This page is for TEC owners. Through this, we can conduct a forum on the uses and future of the TEC. As we cannot reply to every letter sent in, we will attempt to answer letters of common interest through this page.

When writing in, put your letter on a seperate page if you are ordering kits etc. This helps us file things in some sort of reasonable order.

SENDING IN PROGRAMS VIA TAPE

We are looking forward to readers sending programs to us. There aren't any in this issue due mainly to the shortage of space. Hopefully, issue 16 will contain several pages or more.

A big factor in deciding weather or not we publish a "reader send-in" is the way the program is sent to us.

If you do have something to send, here is what we want you to do.

Provide us with a copy of the program Save it on tape with a crystal speed of half 3.58MHz. Put your name and address on the tape so we can send it back, if requested.

We also need documentation on the program. Write what it does and where it runs in memory and include any notes you may have generated. The first thing we will do is disassemble it and load it into our IBM clone. Here we can format it for publishing.

For the sake of our disassembler, please, if you can, put tables at the end of the program code and write down where the tables are located. This way we can use our HEX dump routine and tack the tables on at the end of the code.

JMON UP-GRADES

JMON has been designed to be upgraded without losing software compatibility.

Some likely changes are the removal of the low speed tape save (unless there is a storm of protest). This will decrease the software overhead in the tone routine and even-up the period measurement. The result will be an increase in the tolerance of different TEC frequencies and different tape speeds. This should make it possible to freely interchange half 3.58MHz and half 4.MHz tape software as well as allowing poorer quality tape players to be used.

The single stepper, which has no effect on the MONitor at all, may be shifted to a more specialized ROM to increase the stepper's abilities.

The keyboard and LCD RST's will not be changed, so any routine you write using these will run on future up-grades. The same cannot be said if you directly call into JMON. So don't do it!

ISSUE 15 CONTENT

Missing from the TEC section are two usual features. The reader send-ins and tutorial section.

The reason for this is mainly due to lack of room. Already the TEC section is the largest ever and the material left over will be a good start for issue 16.

A different direction is planned for issue 16. The basic lay-out will be two MAJOR add-ons and the rest of the article will be filled with programs (mine and yours, so send them in). There are a couple of reader send-ins that we intend to publish, so if you have sent something in already, we haven't just tossed it in the bin!

JIM's PACKAGE

This package is centered around JMON. The main feature is a complete line-by-line disassembly of the JMON ROM. I hope that, with careful study, you will be able to look at any instruction and understand its role.

My programming style is very optimized. Generally my programs are short and to the point. This does make them a little difficult to read but at the same time by studying and learning my ideas, your own programming abilities will be improved.

If the role of every instruction escapes you, you will still learn important concepts and a better way to do some things.

I wish I had a Jim's package when I was just starting out!

The package will be 20 pages long as this is the limit of our collating photocopier. If there is enough room, some other notes and programs will be included.

It is a pity that such a listing was not available for the earlier MONitors.

Because Jim's package contains every byte in JMON, you can actually burn your own JMON ROM.

Keep in mind that this means typing in 2k worth of program and one mistake will ruin the whole MONitor.

If you feel up to it then go to it. We don't mind you doing this ONCE for YOUR OWN USE.

This offer does not apply to schools or commercial buyers.

If you don't wish to try to type out JMON, I present you with this offer:

Purchase and pay for JMON and Jim's package together, and save $3.50.

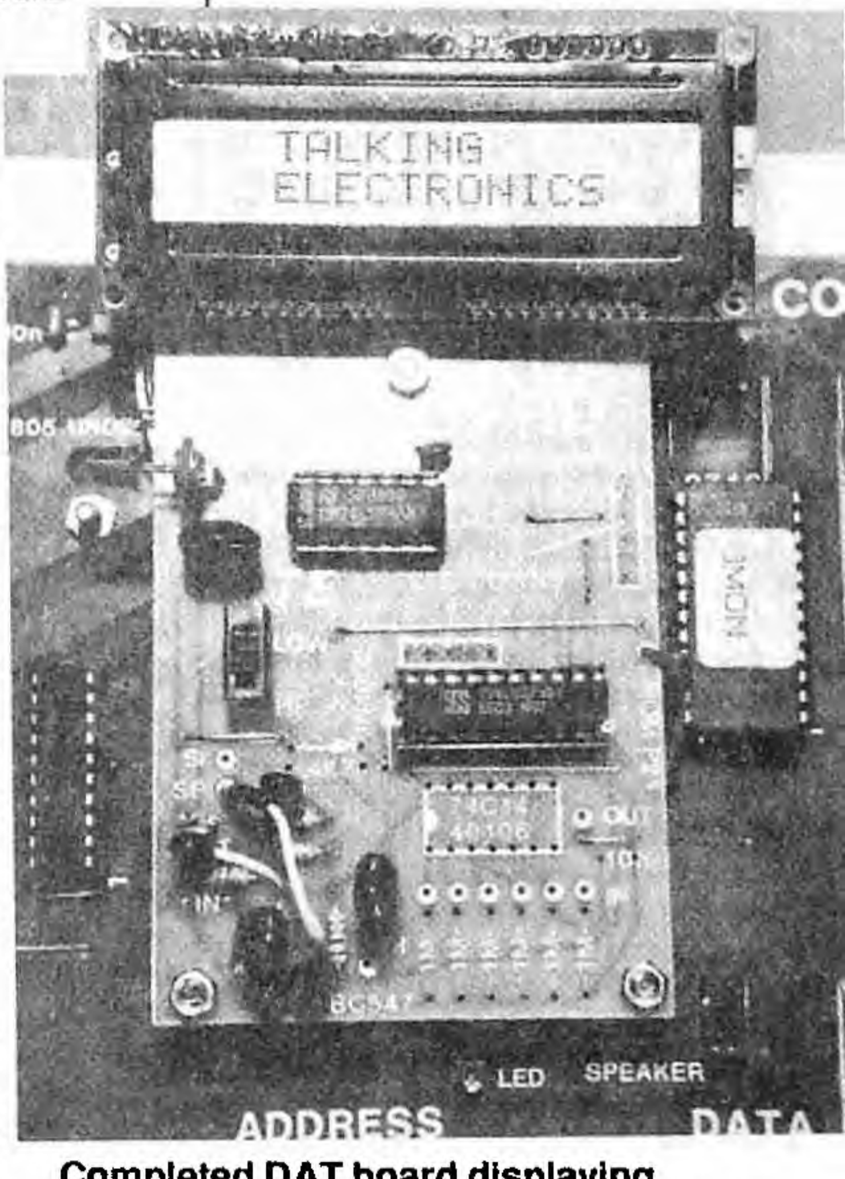
This means the total price for both is $27.50 instead of $31

ISSUE 16 and the TEC

Most of issue 16's TEC pages have been allocated and about half of these are already finished.

If you have some thing for us, don't waste time if you want the chance to see it in print.

- Jim



Completed DAT board displaying "TALKING ELECTRONICS." See the DAT board article starting on P. 47, and LCD article on P.52. Read the whole TEC article before starting anything!

INTRODUCTION

A discussion on Talking Electronics latest monitor for the TEC computer

JMON is a big step ahead for the TEC.

Some of the contents of JMON are: a highly improved Monitor Program, a versatile Tape Storage Program, software for driving a liquid crystal display, a Menu Driver for utilities, a Perimeter Handler, User Reset Patch, Single Stepper and Break Pointer with register display software, and simplified access to utilities and user routines.

JMON also uses indirect look-up tables stored in RAM. This idea leaves the door open for many possibilities.

All the above and more is contained in 2k bytes.

The following is a description of the major blocks in the ROM.

THE MONITOR PROGRAM

To support new features added to the TEC, a new interactive monitor program has been written. The new monitor is, by itself, a considerable upgrade over previous monitors and when combined with other software in the monitor ROM, gives great features for the TEC user.

Major improvemeots have beeo made in the MONitor software, to allow quicker entry and editing of code. This has been achieved by adding such features as auto key repeat and auto increment. If you add the LCD, its larger display and cursor control software open up a second level of improvement.

THE TAPE SOFTWARE

The TAPE SAVE facility is versatile and reliable.

Some of the functions include: 300 and 600 baud tape SAVE, auto execution, LOAD selected file, LOAD next file, LOAD at optiooal address, TEST tape to memory block and TEST tape check sum. Both tests may be combined with other options.

The tape software uses the universal MENU driver and perimeter handler. These rontines allow easy selection of cassette functions (e.g. Load, Save, etc.) and easy passing of variables to the tape software.

Article and monitor by Jim Robertson

The tape software contains check-sum error detection that allows the user to know if the load has failed. A check-sum compare is performed after every page (256 bytes) and also after the leader is loaded. This means the user does not need to wait until the end of a load or test for error detection.

Each full page to be loaded, tested or saved, is displayed on the TEC LED display. Up to 16 pages are displayed.

Upon completion of a tape operation, the MENU is re-entered with an appropriate display showing:-END -S (END SAVE); PASS CS (CHECK SUM); PASS Tb (TEST BLOCK); PASS Ld (LOAD); FAIL CS (CHECK SUM); FAIL Tb (TEST BLOCK); FAIL Ld (LOAD).

The one exception is when an auto execute is performed after a successful load.

The tape software will display each file as it is found and also echo the tape signal.

LIQUID CRYSTAL SOFTWARE

This software is called from the monitor program. It is possible to de-select this software to allow the liquid crystal display to perform a nser-defined purpose while the monitor is being used.

The Liquid Crystal Display is being accessed as a primary output device to the user during the execution of the monitor. Eight data bytes are displayed at a time and a space between each for the prompt (it appears as a "greater than" sign). Four digits in the top left hand corner show the address of the first byte.

In the bottom left hand corner is a current mode indicator and this lets you know which particular mode JMON is in. E.g. Data mode, Address mode etc.

The prompt points to the next location to have data entered, or if at the end of the 8 bytes being displayed, the prompt parks at the top left corner indicating a screen change will occur on the next data key press. This allows revision before proceeding.

It is possible to use the monitor with only the LCD unit, the only drawback being the actual current value of the address pointer is not displayed (the value showo in the address portion of the LED display). However this is only minor.

MENU DRIVER

This is a universal routine used to select various utilities routine from JMON. It is already used by the tape software and the utilities ROM. It may also be easily used by the TEC user.

The Menu Driver displays names of functions in the TEC LED display. The number of different names is variable and may be user defined. It is possible to step forward and backward through these names.

A 3-byte jump table with an entry for each name provides the address of the required routine. A jump is performed upon "GO."

To have a look, call up the cassette software by pressing SHIFT and ZERO together. If you have not fitted a shift key, the cassette software can be addressed by pressing the address key, then the plus key, then zero.

To move forward through the MENU, press "+". To move backward, press "-". Notice the automatic FIRST-TO-LAST, LAST-TO-FIRST wrap around.

Pressing "GO" will take you into the perimeter handler.

PERIMETER HANDLER

Like the Menu Driver, this is a universal program and may be easily used by the user.

This routine allows variables to be passed to routines io an easy manner. The variables are typically the start and end address of a block of memory that is to be operated on, such as a load, shift, copy, etc.

A 2-character name for each 2-byte variable is displayed in the data display while the actual variable is entered and displayed in the address display.

The number of variables may be from 1 to 255 and is user definable.

The data display is also user definable.

It is possible to step forwards and back through the perimeter handler in the same fashion as the MENU driver.

When a "GO" command is received, control is passed to the required routine via a 2-byte address stored at 0888 by the calling routine.

The SINGLE STEPPER and BREAK POINT handler.

A single stepper program can be important when de-bugging a program. It effectively "runs" the program one step at a time and lets you know the contents of various registers at any point in the program.

If you have ever produced a program that doesn't "run", you will appreciate the importance of a single stepper. Many times, the program doesn't run because of an incorrect jump value or an instruction not behaving as the programmer thinks.

The single stepper runs through the program one instruction at a time and you can halt it when ever you wish. By looking at the contents of the registers, you can work out exactly what is happening at each stage of the program.

The single stepper operates by accessing a flip flop connected to the Maskable Interrupt line of the Z-80. It can be operated in the manual mode, in which a single instruction is executed after each press of the "GO" key. In the auto mode, 2 instructions are executed per second.

BREAK POINTS

Break points work with groups of instructions. They allow register examination in the same way as a single stepper. The advantage of break points is that there is no time wasted stepping slowly through a program. This is particularly important as some programs contain delay loops and they may take weeks to execute at 2Hz!

Break points are one of the most effective ways to debug a program!

STARTING WITH JMON

JMON is straight forward to use. Some new habits must be learnt, however they are all quite easy.

JMON has 4 modes of operation. They are:

DATA MODE, ADDRESS MODE, SHIFT MODE and FUNCTION MODE.

The data address and shift modes are not new but have been, in part, changed in their operation. The function mode is new to the TEC and I am sure you will find it useful. Below is a description of each mode.

THE DATA MODE

The data mode is used to enter, examine and edit, hex code into RAM memory. It is identified by one or two dots in the data display and the word "DATA" in the bottom left hand corner of the LCD display. It is similar to the data mode on all previous MONitors.

The data mode has a sub-mode called AUTO INCREMENT. This is a default setting, meaning that it is set to auto increment on reset. The user may turn off the auto increment sub-mode if desired.

When in the auto increment mode, the current address pointer in the address display is automatically pre-incremented on each third data key press.

A SINGLE DOT in the RIGHT-MOST LED display indicates the current address will be incremented BEFORE the next nibble received from the keyboard is stored.

This allows the user to review the byte just entered. If an incorrect nibble is entered, the internal nibble counter MUST BE RESET by pressing the ADDRESS KEY TWICE. Then two nibbles may be entered at that location. This is a slight annoyance at first, but it is a small price to pay for such a powerful feature as auto increment!

After two nibbles have been entered, the prompt on the LCD is IMMEDIATELY updated and points to the next memory location, or in the case of the last byte on the LCD, the prompt PARKS AT THE TOP LEFT CORNER signifying an entire screen update UPON THE NEXT DATA KEY PRESS.

This allows the user to revise the entered code before continuing.

You must be in the data mode to perform a program execution with the "GO" key. (Actually, you can be in the SHIFT mode also.)

Because of the auto key repeat, and "auto increment", it is possible to fill memory locations with a value by holding down a data key. This may be useful to fill short spaces with FF's or zero's.

Because the LCD prompt is advanced immediately after the second nibble being entered while the LED display is advanced on the third nibble received, the "+" key will advance only the LED display while the "-" key will shift the LCD prompt back two spaces, if either are pressed immediately after the second nibble is entered. This may seem strange but is the result of a clever design which allows for revision of entered code on either display before proceeding.

ADDRESS MODE

This is identified by 4 dots appearing in the address display of the LED display and "ADDR" in the LCD bottom corner.

The address key is used to toggle in and out of this mode.

TEC INVADERS AND MAZE

These two games come on a 10 minute tape with instructions and a detailed diagram of the "invaders" screen showing the various characters.

The instructions are basic but sufficient. One VERY IMPORTANT omission is the 8x8 is connected to PORTS 5 and 6 for both games.

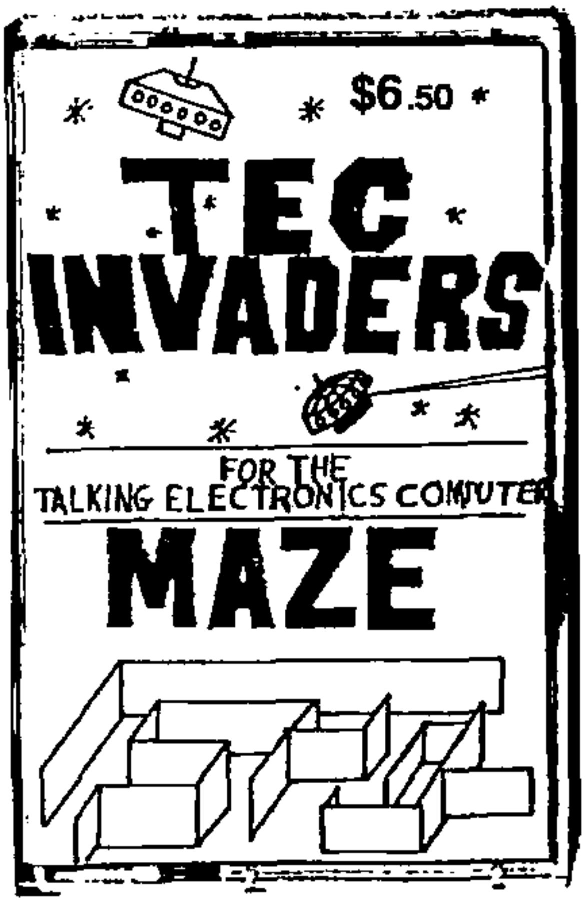
Both games are very entertaining but invaders suffers a little by the limitations of the 8x8.

However it does impose a challenge and you can constantly improve on your score.

Maze dose not suffer one bit by the limits of the 8x8. In fact the 8x8 is perfect for the Maze. The scrolling effect has to be seen to be appreciated.

Maze is a game to keep you occupied for hours.

See Camerons tape #1 on P. 39.



The address mode will be entered by an address key press from either the data or function mode. An address key pressed while in the address mode will result in a return tn the data mode.

While in the address mode, data keys are used tn enter an address while the control keys (+, -, GO) are used to enter the functinn mode. Nn auto zeroing has been included, therefore 4 keystrokes are required to enter any address.

SHIFT MODE

This mode allows easy manual use of the cursor. The shift works by holding down the shift key and at the same time, pressing a data key.

The monitor must be in the data mode and only data keys work with the shift.

Sixteen functions are available but nnly ten have been used in this monitor.

The shifts are:

Shift-zero: Cassette MENU is displayed.

Shift-one: Cursor back one byte.

Shift-two: Start single stepper at current address.

Shift-four: Cursor forward 4 bytes.

Shift-five: Break from shift lock (see function mode).

Shift-six: Cursor back 4 bytes.

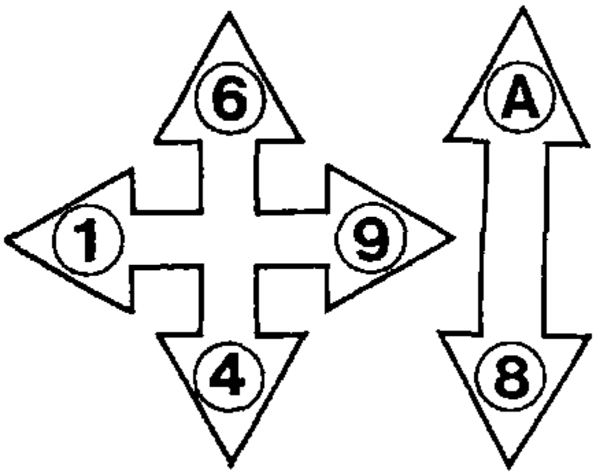
Shift-seven: Enter register examination routine.

Shift-eight: Cursor forward 8 bytes.

Shift-nine: Cursor fnrward 1 byte.

Shift-A: Cursor back 8 bytes.

Nnte that 1, 4, 6 and 9 form a cross and 8 and A form an arrow and each is positioned to cnrrespond to their cursor movement.



Keys 1, 4, 6 and 9 move the cursor LEFT, RIGHT, UP AND down on the LCD.

Key "A" shifts the screen back to display the previous eight bytes.

Key "8" shifts the screen forward eight bytes.

When editing a program, the shift enables fast movement through the memnry. Data entry is achieved by releasing the shift key.

The shift mode is not identified explicitly nn either display.

THE FUNCTION MODE

This has been provided to enable a quick way tn call commonly used routines. Only three keystrokes are required invoke up to 48 different routines.

The function mode is broken up into 3 sections.

They are: Function select-1, Function select-2 and Function select-3.

Each is identified by a single dot in the address display: right-most for function 1, second right for function 2 and third right for functinn 3. On the LCD display, the functions are identified by: Fs - 1, Fs - 2, or Fs - 3 in the bottom left corner.

Fs - 1, Fs-2 and Fs-3 are entered FROM THE ADDRESS MODE by pressing the "+" key for Fs-1, the "-" key for Fs-2, the "GO" key for Fs-3.

It is possible tn swap between sections without coming out of the current function mode by pressing the required function select key. After entering the required section, A DATA KEY IS THEN USED TO SELECT ONE OF SIXTEEN ROUTINES.

The address of these routines are stored in a look-up table.

SECTION-1 - the SHIFT-LOCK FEATURE.

Section-1 is selected FROM THE ADDRESS MODE by pushing the "+" key. The keys 0, 1, 2, 4, 5, 6, 7, 8, 9 and A then have the functions as listed in the shift mode. (Key 5 has the function of returning to the data mode.)

Cursor control rnutines return back to section-1 to enable continuous cursor mnvement (shift-lock).

The look-up table fnr the jump addresses for section-1 is at 07E0.

SECTION-2

Section-2 is selected from the address mode by pushing the "-" key. This is unused by any existing software and is available to the user.

HERE'S HOW TO USE IT:

Using the section-2 is very easy. All that is required is to enter the address(es) of the required routines in a table. The table begins at 08C0. The first two bytes at 08C0 correspond to the zero key in sectinn 2. While the second two (08C2) correspond to key nne etc.

Here is a shnrt program as an example:

```
08C0: 00 09 04 09 08 09
```

(These are the addresses of the routines).

```
0900: 3E EB 18 06 3E 28 18 02
0908: 3E CD D3 02 3E 01 D3 01
0910: 76 C9
```

Now push ADdress, "-", "0" and the routine at 0900 will be CALLED from the MONitor. Reset the TEC and try ADdress, "-", "1" and ADdress, "-", 2.

Because these routines are CALLED from the MONitor, you may use a return (RET, C9 or RET NZ etc.) instruction to re-enter the MONitor in the same state as you left it. e.g. in the function select-2 mode.

SECTION-3

This has been reserved for the utilities ROM at 3800. The table for Section-3 is at 3820.

USING THE SINGLE STEPPER

Getting the single stepper to work is simple enough, however there is some skill required to understand its limitations and knowing how to avoid them.

To start with, you need a program that you require tn be SINGLE STEPPED.

This program may be anywhere in memory except in the lowest 2k (the MON ROM).

This is because the MON select line is used as part of the timing. You may call intn the MON ROM but only the first instruction will single step, and when returning out of the ROM, the next instruction will also not be stepped. (However they will be executed at normal speed.)

Programs that use the TEC's keyboard require careful attention as you cannot step them in the normal way. This is because there is no way to distinguishing between key-presses for the single stepper and those for the subject program.

This reduces the usefulness of the single stepper a little however thoughtful snftware design enables a fair degree of flexibility and this problem may be side-stepped.

The key use of the single stepper is as a de-bugging aid. When you are writing programs. effective use of the single stepper usually requires that while writing your programs, you allow for the use of the single stepper by leaving room to place one byte instructions that turn ON and OFF the single stepper.

Programs using the keyboard may be stepped by turning OFF the stepper. This allows areas requiring use of the keyboard to run in real time while other areas may be single stepped. This applies only to programs that use the keyboard routines provided inside JMON.

The only disadvantage here is that after completing your program you may have NOPs left.(from where you blanked over the single stepper control bytes).

The keyboard controls for the single stepper are as follows:

To start single stepping from the current address, this is what to do: From the data mode, press shift-2. This will start the single stepper. The first instruction will be performed and the address will be displayed as "PC" (Program Counter) on the single stepper. To examine the registers, press "+". The left two nibbles correspond to the high order byte and in the case of register pairs, the left-hand register. You may go backwards also by pressing "-". The registers displayed are : PC, AF, BC, DE. HL. IX, IY, AF', BC', DE', HL' and SP, in this order.

To step the next instruction, press GO. You can also step continuously at about 2Hz by pressing any data key.

When in the auto step mode. you can stop at any time and examine the registers by pressing "+" or "-", or bring it back to the manual mode by pressing GO.

The address key resets back to the MONitor unconditionally. The control bytes for the single stepper are as follows:

To stop single stepping in a program: F3 (disable interrupt).

To restart in a program: EF (restart 28). This causes a restart to 0028 where a routine passes the start address (which is actually the return address of the restart 28 instruction) to the single stepper. It also enables the interrupts and then returns to the next instruction which is then single stepped.

This SINGLE STEPPER is only a first model. Hopefully, when more room is available, some improvements can be added. One imprnvement on the "cards," is allowing it to be interfaced with a utilities ROM. This ROM will extend the display capabilities, allow editing while stepping and to disassemble on the LCD each instruction as it is stepped. If you have any ideas or requirements, write in and tell us.

BREAK POINTS

Break points are locations in a program where execution is stopped and the registers are examined in the same fashion as the single stepper. The advantages over single stepping include real time execution and less or no control bytes in a program. They also usually allow much quicker fault finding.

As a trade-off move, only a simple (but effective!) form of break-point is available with JMON. This allows for more MONitor functions and also eliminates the need for extra hardware.

More complex methods automatically remove the break-point control byte and re-insert the correct op-code and allows re-entry to the program.

USING JMON BREAK-POINTS

Break points are achieved by using a restart 38 instruction. The op-code for this is FF and all that is required is for it to be placed where ever you require your break point.

Before running your program, make sure the TEC is reset to 0900. This is necessary to clear the auto-repeat on the stepper/break-point register display. (This is explained in the LCD section).

Simply run your program as normal. When the break point is reached, the register display routine is entered. The value of the program counter display WILL NOT BE VALID on the first occurring break unless you provided the address of the break point at 0858. This minor flaw was unavoidable without considerable additional software which would have "eaten" memory like there's no tomorrow!

If you allowed for break commands in your program, you may then have multiple breaks and step to the next break with the GO key.

However if you placed a break command over an existing instruction then no further breaks will be valid and you should never try multiple breaks in this case AS YOU MAY CRASH THE MEMORY.

In the above case, make a note of the contents of the registers and return to the monitor via the address key and then examine memory locations, if required. (You may enter the register examination routine via shift-7). Further breaks should be done by removing the existing break and placing it where required and re-executing the program from the start.

Some other good ideas are to load the stack away from the MONitor's RAM area. (08F0 is good but make certain that 08FF does not contain AA - as this prevents the MONitor rebooting its variables on reset and your stack may have accidentally crashed these variables.) Also, if you are using the LED display scan routine in the MONitor ROM, shift your display buffer to 08F0 by putting this address into 082C/2D. Now you can examine your stack and display values after returning to the MONitor.

There is a conditional way to cause breaks. To do this requires a conditional jump relative with FF as the displacement. If the condition is met, the jump is made and jumps back onto the displacement which then becomes the next op-code! Remember this as it is a very useful idea. You cannot continue on with multiple breaks after a break caused by this method.

Break points are a quick way to debug a program. It is very important that you familiarize yourself with them. They bave been the single most important programming aid used when writing most of JMON and the utilities ROM.

SUMMARY:

Clear the auto-repeat via the reset.

: Use FF to cause a break

: PC not valid on first break.

: For multiple breaks, provide spaces for the break control byte.

: Shift stack and display buffer (optional)

: Use FF as displacement for conditional breaks.

Finally, make sure you write down when, where and why, each time you insert a break-point.

ACKNOWLEDGMENT

Thanks to MR. C PISTRIN of Traralgon VIC. His SINGLE STEPPER program for the MON-1B inspired me to include one in JMON and provided me with a circuit for the hardware section.

See page 47 for the circuit

THE TAPE SYSTEM

This discussion covers all the areas needed to use the tape save and its various options.

TEC CONSIDERATIONS

The tape software works on any type of TEC, the only consideration is the various different clock speeds.

The following description generally applies to TEC's with a crystal oscillator that is fitted with a colour burst (3.58MHz) crystal and divide-by-two stage.

If you are still using a 4049 based oscillator, the tape system will work ok, but it will be very important to note the TEC clock speed when saving as the TEC must be set to the same speed when re-loading. Another problem can be the drift in frequency over a temperature range and the different oscillator frequencies between TEC's.

When saving a tape, the best idea is to wind the clock up to full speed, and then turn back the speed control pot one quarter of a turn. This will allow you compensate for speed drift if ever required.

The tape also works very reliably with a 4MHz crystal and divide by two stage, however a tape written using a 3.58MHz oscillator cannot be loaded by a TEC that uses a 4MHz oscillator, and vice versa.

If you are sending programs into TE on tape, they must be recorded with the 3.58MHz crystal. (divided by two).

The tape system has been extensively tested and found to be very reliable under a wide range of conditions. We don't expect you to have any trouble in getting it to work reliably for yourself.

LET'S BEGIN

To start with, you need a JMON monitor ROM as the tape software is inside this ROM.

Secondly, you will need a cassette recorder with both "mic" and "ear" sockets. Any audio cassette player of reasonable quality should be ok, provided it has the two sockets mentioned above.

We have tested more than six types, and found them to be quite suitable.

Thirdly, you will need to have constructed the cassette interface on the LCD interface board and have made up the two connecting cables, with 3.5mm plugs on each end. Finally you will need a new C60 or C90 cassette of the better quality types, such as TDK or Sony. We found the cheap tapes from the junk shops or supermarkets to be unreliable. (Some of them didn't work AT ALL, so don't take the chance).

Now connect the "mic" on the tape recorder to the "tape out" from the TEC and "ear" socket to the "tape-in" on the TEC. (It's a good idea to mark the cables between the recorder and the TEC to prevent incorrectly connecting the leads).

Insert a tape and we are ready to learn how to operate the system.

HOW TO OPERATE THE TAPE SYSTEM.

We will start by saving a few bytes at 0900. Enter at 0900 the following: 01 02 03 04 05 06 07 08 09 0A.

OK. Now connect up the tape recorder as described above and call up the tape software by pushing shift and zero at the same time or Address, "+","0" consecutively.

The TEC display will now show "SAVE-H" and this is the heading for SAVE at HIGH SPEED. Now select this by hitting "GO"

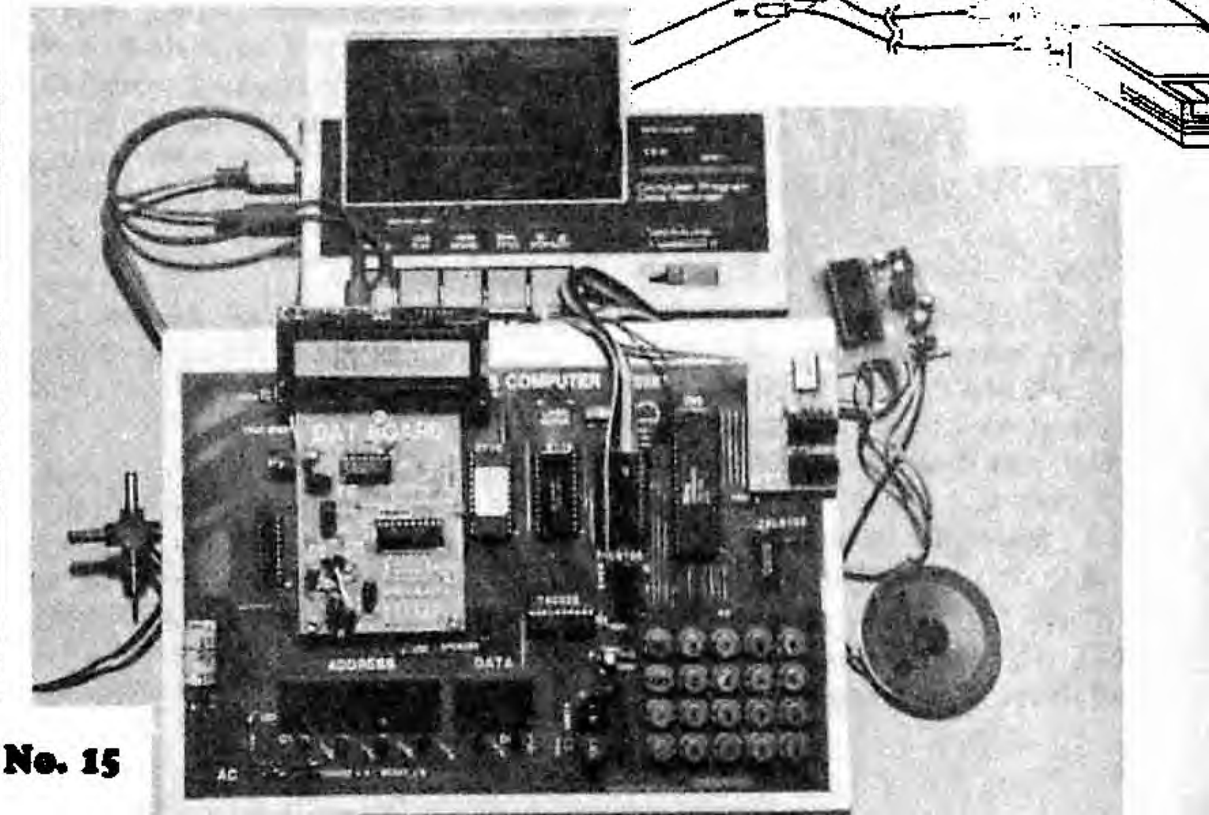
The display will now have a random two-byte value in the address display and "-F" in the data display. The "-F" in the data display is for the file number, while the address number is just junk from the RAM. You can enter a file number by pressing the data keys. Enter anything you want. The numbers you enter will shift across in the same fashion as when entering an address on the MONitor. Then when you have entered a file number, press the "+" key. The data display will now show "-S". This is where you enter the start of the block you wish to output. Enter 0900, and then press "+". The data displays now show "-E." This is where you enter the address of the last byte of the block to be saved.

Enter 090A and press "+". The next data display is "-G". This is the OPTIONAL AUTO-GO address, this is always set to FFFF by the software as this is its NON-ACTIVE state, i.e. NO AUTO-GO upon a re-load. ANY other value entered here will result in an automatic execution upon a SUCCESSFUL LOAD AT THE ADDRESS ENTERED HERE.

We don't require auto execution, so leave this at FFFF.

Now press play and record on the tape recorder and wait for the clear plastic leader to pass if at the start of a tape. Incidentally, it is not a good idea to remove this leader as has been advised in another magazine, as it protects the tape from stretching and possibly breaking, when rewound.

When the tape is right, press GO. The display will blank and a continuous tone will be heard from the speaker. After a few seconds the file information will be outputted and then a period of high frequency tone. This "middle sync" tone is to cover the time that the filename is displayed when re-loading.



After the high tone, the code will be nutputted and also a digit will appear on the TEC LED display. This is the number of COMPLETE pages to be saved. In this case it will be zero.

A point to raise here is that if you ever accidentally enter a start address that is HIGHER than the end address, when GO is pressed, the software will detect this and display "Err-In". In this case, Push "+" or "-" to go back to the perimeter handler where you can correct the error.

When the code has been saved, a short end tone will be heard and then the menu will re-appear with "-END-S", meaning end of save.

Once the code has been saved, rewind the tape.

To re-load the tape press the "+" key and you will see "SAVE-L" on the display, then "TEST-BL", "TEST-CS", then you will come to "LOAD-T" (for load tape). Note that there is no "TEST-H" or "TEST-L" for low and high speeds as the test and load routine will load either speed automatically.

Press GO. The data display shows "-F" for file number. This will be as you left it when you saved. When loading or testing from tape, the file number here determines which file will be subject to the selected operation. If you enter FFFF here, the next file found will be used, regardless of its file number.

For now, we will leave it as it is.

Next push "+". The data display will show "-S", meaning Start address. This is always set to FFFF by the software. The start address allows you to optionally load a file or test a file at an address different to the one on the tape, (which is the address from which it was saved). To demonstrate its operation and to make it a more convincing trial, we will enter 0A00. The file will now be loaded at 0A00. If you press the "+" key again, you will be back at the file name. (This last point demonstrates the programmable number of "windows" feature of the perimeter handler. It was set up for 2 "windows" by a short routine entered from the Menu driver before passing control to the Perimeter handler, remember that there was 4 "windows" when you saved the file).

Now press GO.

The display will blank. Now start the tape playing. The sound from the tape will be echoed on the TEC speaker. Soon the leader will be heard and it should sound as crisp as when it was saved. If not, experiment with the volume. The interface allows for a wide variation of volumes but 3/4 volume is a good place to start.

After the leader has passed, the file name is loaded and should appear on the display. If it was not correctly loaded, "FAIL-Ld" will appear. In this case experiment with the volume and retry. After a few seconds the file name disappears and the number of complete pages to load are displayed on the middle digit. The code is now being loaded.

The code is loaded very quickly and hopefully a "PASS-Ld" will appear. If not, re-try with a different volume setting. After you have successfully loaded, hit reset and ADdress 0A00 and 01, 02 etc. will be found.

If you are unable to get a successful load after many attempts, then skip ahead to the trouble shooting section.

Now we have a successful load, we will experiment with the TEST BLOCK function.

Change a byte in the 0A00 block. Now call up the tape software (Shift-0, or ADdress, "+" ,0), select "TEST-BL", and LEAVE FFFF at the optional start. ("-S") Then rewind the tape and play it back like you did when loading.

At the end of the test, the display comes back with "PASS-TB". Now do this again, but this time enter 0A00 at the optional start and FFFF for the file. This will demonstrate the load/test next file feature.

Because 0A00 has been entered in the optional start "window", the test will be between tape and the code at 0A00.

Rewind the tape and press "GO" on the TEC, then play the tape.

Because a byte has been changed, the test this time will fail and the display will show "FAIL-TB."

Use the test-block feature whenever you wish to compare a tape file with a memory block or test that a save operation was successful.

If ever revising software on a tape of which you do not have a copy in memory, use the test checksum (TEST-CS) to ensure that the file is good. By use of the "LOAD NEXT FILE" feature (FFFF in the file number window) you can go through a tape completely, checking each file.

THE "AUTO-GO"

To use the Auto-GO feature, you must enter the required GO address WHEN YOU SAVE THE FILE. The go address is entered under the "-G" data display.

Experiment with the following:

```
0900: 21 10 09 11 00 08 01 06
0908: 00 ED B0 CD 36 08 18 FB
0910: 6F EA C6 EB E3 EB.
```

Save this as described above, but this time enter 0900 under the "-S" heading, 0921 under "-E", and 0900 under "-G".

Now re-load it and if the load is successful, the program will start automatically and an appropriate display message will appear.

USING THE TAPE SYSTEM.

The primary use for your tape save system is as a mass storage device for your files.

Files may be saved and loaded as described previously, the important addition here is good paper-work habits. It is very important to keep a log of your files or you will quickly forget what you have, where it is located, and you will end up writing over your files!

Your log system must include identifying each cassette and the side of the tape, the files on the cassette in the correct order, how many of each file, the date and any notes on the file. If your recorder has a tape counter facility, it makes good practice to record the readings from this, so that files may be quickly found anywhere on the tape.

Also a great aid is to log approximately the location of each file e.g. half-way, 30 seconds from rewind from end etc.

Apply the above idea to the start of vacant area on the tape also.

Another very good way to use the tape system is as a "RUNNING LOG", where a whole side of a cassette is used to save a developing program, stage-by-stage. If you crash your program, you can re-load it back from tape. A good idea here is to use the high byte of the file number as the program identification and the low byte as the progressive count or version number on the tape.

When you have a final version, then save that on a permanent cassette. The "RUNNING LOG" cassette can then be used over and over.

Once again, paper work is very important. Make sure you document any differences between successive files. This may help later in de-bugging. Also, always include the date and time as this will give a chronological order to your work.

If you are wondering how many times you should save a file, and at what speed, the answer really depends on the reliability of your system.

The major factors in reliability are your tape player and cassette quality and how well you constructed your interface. If any of these are horderline, the system may work but you may have a higher than normal failure rate. Our tests show reliability at better than 98% on saves of 2K blocks. Different cassettes and players were used over many months and rarely did a fault creep in.

You can test your system out by saving the monitor 10 times on each speed and then perform a BLOCK TEST. You should get at the very least, 17 out of 20 passes. If not, some trouble-shooting may be required. If you get 19 or 20 you could probably get away with high speed saves and not have to worry about checking them on your running log. For permanent storage, a good system is a high speed save, then two low speed saves and check each afterwards.

The low speed save should be more reliable than the high speed save as the low speed save will tolerate the occasional hiccup. However, this extra reliability does not cover all possible causes of failure, e.g. problems related to frequency or bandwidth restrictions of your tape player as the period is not changed only the ratio of pulses.

Finally, a file that is absolutely necessary to be retrieved from tape must be stored on two tapes. This provides a double back-up facility against accidental erasure or damage.

"OH NO!" IT DOESN'T WORK.

If your tape system fails to work correctly, then check the interface board or better still, have a friend check it.

Eliminate any problem and re-try.

If problems still exist, test the cassette player with a normal pre-taped audio tape. The music should sound normal and not flutter. If it flutters, the tape player is due for a service or replacement, or if battery operated, the batteries may be flat.

Various sections may be eliminated by listening to the tape signal. If the signal saved on the tape sounds ok when played back on the player, but is not heard on the TEC, check the input section of the interface board and also the "E" output of the player with a pair of Walkman-type headphones.

It is possible that the volume output is not high enough to be amplified on the interface board.

This is very unlikely though on ordinary tape players but we found this to be the case with our VZ-200 data cassette player.

If no signal is getting to the TEC and everything else seems to be ok, test the input buffer by setting the tape software to load and taking the input high and low with a jumper lead.

The LED on the speaker should echo the inverse of the input. If not, shift the jumper to the collector lead of the input transistor and repeat the process.

If the speaker LED now toggles, the input transistor is faulty.

If not, investigate the latch chip. Make sure all the pins are well soldered and the feed-throughs are connecting properly.

If the tape signal is heard, in the TEC speaker, but the file number is not recognised, loaded correctly, or the tape fails to load the data blocks consistently, try a better quality cassette tape. If problems persist, try a different player as the signal may be distorted or not have enough amplitude.

If you still can't get it to go, a repair service is available for $9.00 plus $2.50 postage.

RUNNING OLD PROGRAMS WITH JMON

Most old programs will run with JMON without to much alteration.

For most, they will only need to be relocated from 0800 to 0900 and that's all. Those that use old MON-1 routines such as the running letter program or the tune player can't, of course, run with JMON.

Of those which use the keyboard, most can be easily altered but some require a complete overhaul.

These ones listed below cannot run on JMON, or require more extensive changes that those presented here:

SPIROID ALIENS, HALILOVIC'S PIANO, BIG BEN CHIMES, WINNERS CALL, YOU'RE DEAD FUNERAL DIRGE, TOCCATA, THE STRIPPER, ADDING AUTO REPEAT, AUTO RETURN AND STOP, AUTO MOVEMENT AND HALT, THE ROMMED PRINTER SOFTWARE and SPACE INVADERS SHOOTING.

Those not mentioned should run ok if re-located to 0900 and the mods listed below are done, if required. These mods apply only to routines which use the keyboard.

TEC KEYBOARD THEORY

Basically the keyboard usage of earlier routines is broken up into two types: Those which halt and wait for a input via the interrupt, and those which intialize the input buffer (the interrupt vector register or I register) and read it "on the fly."

The first type may again be broken into two groups. Those which explicitly read the value from the input buffer, with a LD A,I instruction (ED 57), and those which assume the input to be inside the Accumulator after the interrupt has occurred. (Remember the earlier MON-1 series did not save the accumulator during the NMI routine but instead returned with the input key value inside the accumulator. A disastrous state of affairs)!!

JMON

The specially-provided routines in JMON will work for all the above types, the only difference being the way you alter the program in question. Here's how to alter the routines:

To up-date any type which uses a HALT instruction (76H) as part of the keyboard input section, change the HALT instruction (76H) to a RST 08 (CF).

The RST 08 routine SIMULATES the halt instruction by first looping until NO key is pressed then looping until a key IS pressed.

After a key press is received the input value is masked to remove unwanted bits and stored in the input buffer, (the interrupt vector register), in identical fashion to the old interrupt routine.

Now if the halt instruction is immediately followed by a LOAD A,I (ED 57), you may leave it as it is or remove it as it is not required any more as the input value is returned in A.

If a program doesn't have a HALT instruction but uses the keyboard, then look for the LOAD A,I instruction (ED 57). Change this to a RST 20 (E7) and place a NOP over the unused byte. Notice that this IS NOT the same RST instruction as above.

Be careful not to mistake the LOAD A,I (ED 57) with a LOAD I,A (ED 47) otherwise you program may get upset and go on strike.

Programs which have neither a HALT or LD A,I instruction cannot be altered by any of the above methods because they enter a continuous loop and require the interrupt to force an input value into the accumulator. A classic example of this is the "space invaders shooting" on page 14, issue 14. This above loop is located at 0821. (while you're looking at this, grab a pen and change the byte at 0812 from 02 to 01, at least it will run correctly with MON-1)! All the above types are among those listed as not being suitable for modification via these methods.

FINALLY

If you find a program which doesn't work (we haven't tried them all) or something else interesting, please write and let us know.

USING THE KEYBOARD IN YOUR PROGRAMS

The new keyboard set-up is no more difficult to use now than before. In actual fact it is easier and requires less bytes than before thanks to the use of the RST instructions.

Four RST's are provided to handle the keyboard in different ways. The first RST we shall look at is RST 08 (CFH). This RST is a "loop until a NEW key press is detected" routine. If you refer to the section on running old programs, you will see that this RST is used to simulate/replace the HALT instruction. (You know how to use it Already!)

An important feature of this RST is that it ignores any current key PRESSED, that is if a key is being pressed when this RST is performed, it will not be recognized. This mimics the NMI which only recognized a key press once. (This is why the auto-repeat feature could not be done with the keyboard hooked up to the NMI).

When this RST detects a valid key press, it inputs the value from the key encoder aod masks the unwanted bits and stores the input in the interrupt vector register (as did the MON-1 series). The input value is also returned in the accumulator. The shift key can not be read from this (or any other MONitor keyboard routine) as the shift input bit (bit 5) is masked off.

Here is an example of its use:

```
0900  CF      RST 08
0901  FE 12   CP 12
0903  20 04   JR NZ,0909
0905  3E EA   LD A,EA
0907  18 06   JR 090F
0909  FE 01   CP 01
090B  20 F3   JR NZ,0900
090D  3E 28   LD A,28
090F  D3 02   OUT (02),A
0911  3E 01   LD A,01
0913  D3 01   OUT (01),A
0915  18 E9   JR 0900
```

The first thing you should notice when you enter and run the above, is that the "go" key is not detected when the routine is first started, even though it is being pressed. This is because the first part of the RST loops until the key being pressed is released. The RST then loops until a new key press is detected. When the RST returns, the input value is both in the interrupt vector register and the accumulator. The rest of the routine tests for either a 01 or "GO" key and outputs to the display.

Use this RST when ever you want the TEC to go "dead" and wait for a key press.

The second RST is RST 10 (D7). This is similar to the first RST but has one very important difference. The difference is that this RST DOES NOT wait for a key being pressed to be released before returning. While this is not as likely to be used as much as the first RST, it does have some good uses. Any program which requires some action to take place while there is a key pressed, but do nothing when there is not, may make good use of this RST. Some possible uses include random number generation on the time the key is held down; count while a key is pressed; turn on a relay while a key is pressed etc. As you can see, this RST simulates momentary action switches.

This RST exits with the input stored in the same fashion as the above RST.

The third RST is RST 18 (DF). This is a LED scan loop and keyboard reader. The scan routine will scan the 6 TEC LED displays once with the display codes addressed by the address at 082A. (0800 is stored here by JMON. You can leave it as it is, just store what ever you want at 0800 before using this RST). After the scanning routine is done, the keyboard routine is called. The keyboard routine is actually called from RST 20. What happens is this. After the scan has been called from the RST 18, the program continues on at 0020, which is the start address for RST 20. So the RST 18 is the same as RST 20 EXCEPT THAT RST 18 CALLS FASTSCAN. Therefor the description below applies to BOTH RST 18 AND RST 20.

This keyboard routine is very intelligent and is able to detect several different conditions.

One important feature is that it "remembers" if it has already detected the one key press and it ignores it if it has. This provides us with a "ONE AND ONLY ONE" key recognition for each key press. Each key press is "recognized" on the first detection.

The key is checked for a "FIRST KEY PRESS" by the use of a flag byte. When the routine is entered AND NO KEY IS PRESSED, this flag byte is CLEARED. When a key is detected, the flag byte is checked. If zero, the key is accepted as a "FIRST KEY PRESS." The flag byte is then set to stop further "validating" of the same key press. The input value is then masked and returned in the Accumulator (only).

If the flag byte IS NOT CLEAR, then the key is not recognized as "valid."

Careful consideration was giving to the interaction of the MONitor and user routines so that the "GO" command from the MONitor WILL NOT BE TREATED AS THE FIRST KEY PRESS of a user routine. (This was achieved by using the same flag byte for both JMON and any user routine).

HOW TO INTERPRET THIS RST

If a key is recognized as a "FIRST KEY PRESS" then the ZERO FLAG will be set to its active state (a logic 1) and the MASKED KEY INPUT will be returned in the accumulator.

If the key is NOT valid then the ZERO FLAG will be clear AND the accumulator WILL HAVE ALL ITS BITS SET (FF).

(FOR ADVANCED PROGRAMMERS)

In addition to the zero flag being conditionally set, the RST 20 (E7) also sets the carry conditionally, according to the following conditions:

If there is a key pressed then the carry will be SET REGARDLESS of whether it is a "first key pressed" or NOT. If NO key is pressed then the carry is cleared.

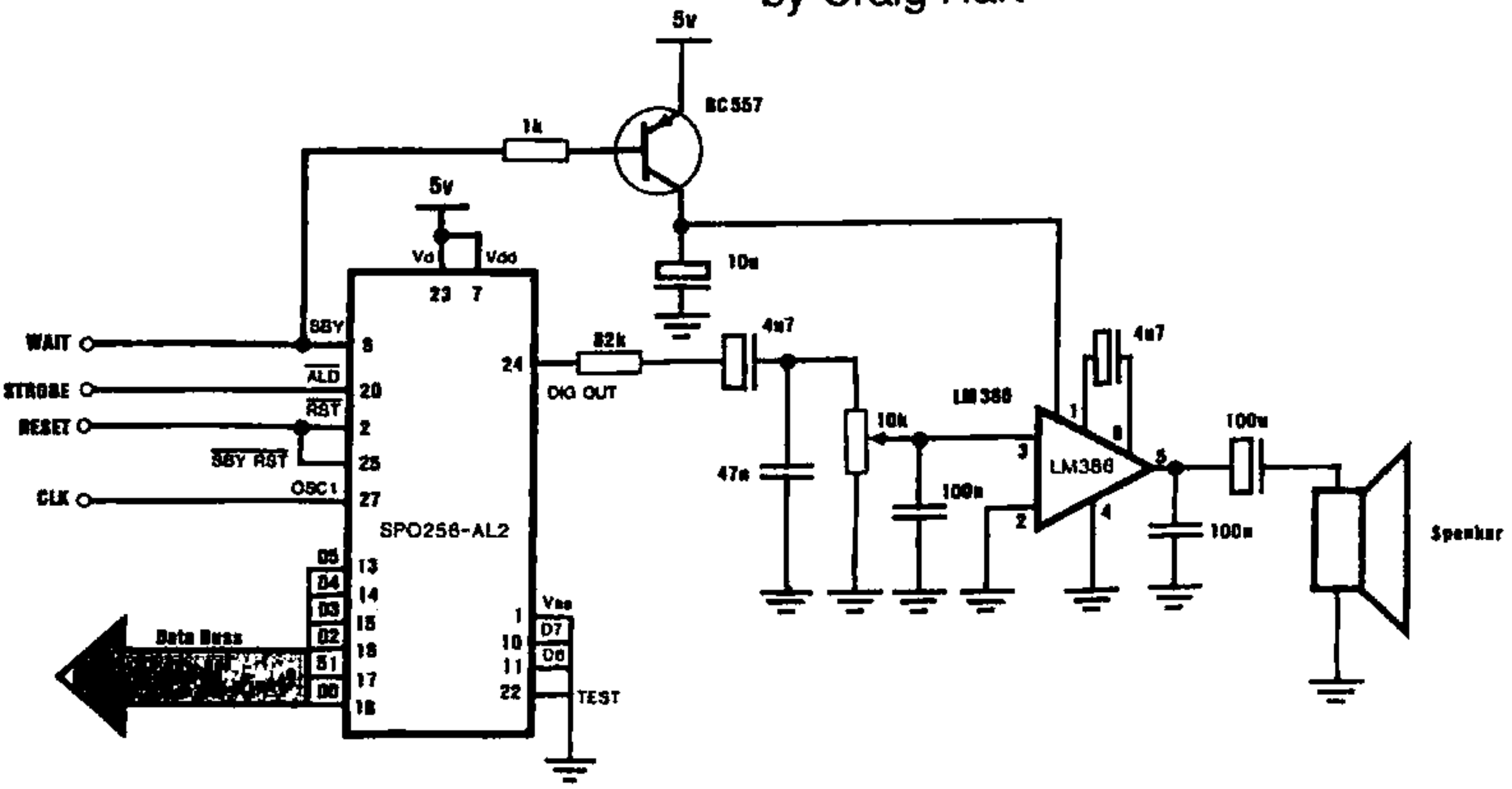
This allows you to interpret the keyboard the way you want, while still giving you the convenience of using the RST to do some of the work.

Jim's section cont P 47.

Speech Module

Add speech to your TEC!

by Craig Hart



SPEECH MODULE CIRCUIT

Since the dawn of time, Colin has been fascinated by electronic speech synthesis, so it was with immense joy that we discovered the SPO256A-AL2 speech chip. This chip is a universal speech unit that can be made to speak almost any English word. The price was cheap and the interface was minimal, it was just too good to pass up! So I took took up the project and this is the result.

The module is interfaced to the TEC, and the TEC controls what is said. The only requirement is that you have a crystal oscillator, as the module requires a 3.58MHz clock signal from the unit. Demonstration programs have been included for testing and simple word sequencing, and these programs will show how the unit is accessed.

This is the ideal companion project to go with the I/O board, and a robot created out of the two projects will cause a real stir if it speaks a comment in response to what it is sensing in its environment.

The module is connected via an 8 way ribbon cable and 4 flying leads. The ribbon cable picks up D0-D5, and the 5v supply. The other 4 leads connect to STROBE 05, WAIT, RESET, and CLK. Note that only the lower six bits of the data bus are used by the speech chip. The reasons for this will be explained later.

OPERATION

The operation of the unit is straight forward, but it is important to understand its operation so that you can use it once you have built it. The SPO256A-AL2 is made to speak by sending it a series of ALLOPHONES. An allophone is the smallest individual sound that the unit can speak. Words and sentences are formed by outputting a series of allophones, one after the other.

Each allophone is assigned a number and this number is loaded into the chip via the TEC data bus, then the ALD line is pulled low (by strobe line 05).

The SPO now commences to speak the allophone and indicates so by pulling the WAIT line low, halting the TEC until the module is ready for more data. The BC557 is turned on hard by this and the LM386 amplifier is switched on.

Sound is clocked out of the unit at a rate determined by the CLOCK line. For normal speech this is 3.58MHz. Sound is filtered by an R-C network, to make the sound more "human like" and amplified by the LM386.

PARTS LIST

All resistors 1/4W 5%
1 - 1k Brown Black Red
1 - 82k Grey Red Orange

1 - 10k trimpot.

1 - 47n greencap.
2 - 100n monoblock.
2 - 4u7 electrolytic.
1 - 10u electrolytic.
1 - 100u electrolytic.

1 - BC557 transistor.
1 - LM386 amplifier IC.
1 - SPO256A-AL2 Speech IC.
1 - 8 pin IC socket.
1 - 28 pin IC socket.

1 - 8 ohm speaker.
4 - PC pins.
4 - PC pin connectors.
1 - 20 cm length 14 way ribbon cable.
1 - 24 pin DIP header.
1 - 10 cm length 2mm heatshrink tubing.

1 - 'SPEECH MODULE' PC board.

When speech output ceases, the wait line goes HIGH, and the TEC is able to continue processing. In doing so, the BC557 is switched off and thus the LM386's power supply is switched off. The reason for doing this is due to the high input impedance of the chip; it is prone to picking up stray noise. The most common noise source is the scanning of the LED displays! This results in an uncomfortable buzz when the unit is not speaking and by switching the power to the amplifier this has been eliminated.

THE ALLOPHONE SET

The SPO has little intelligence about what you want it to speak. You cannot simply feed it a word, and have it say the correct pronunciation in every case. (Although other chips do have this capability) Instead you, the programmer, must translate each word into the appropriate allophone(s) for that word. There are 64 individual allophones, and each sounds different. In these 64 allophones, there are 5 pauses of various lengths, correspooding to word and sentence breaks.

By consulting the Allophone reference table you can look up what you think the right sequence is then play around with different pronunciations of the same basic letter, until you reach the best sounding word. It can be a tedious process, but many common words have been pre calculated and a list appears at the end of the article, along with the table of individual allophones.

Take a sample word: ALARM. Sound out the word slowly, letter by letter. Now look for a matching sound in the list. Write down your guess and progress through the word. Where you have two or more choices, pick the allophone of the appropriate length. For alarm, I chose AA LL AR MM, or 18 2D 3B 10. Add a pause to the end and the terminating byte 04 FF. Plug the data into the test program at 0910 and ruo it.

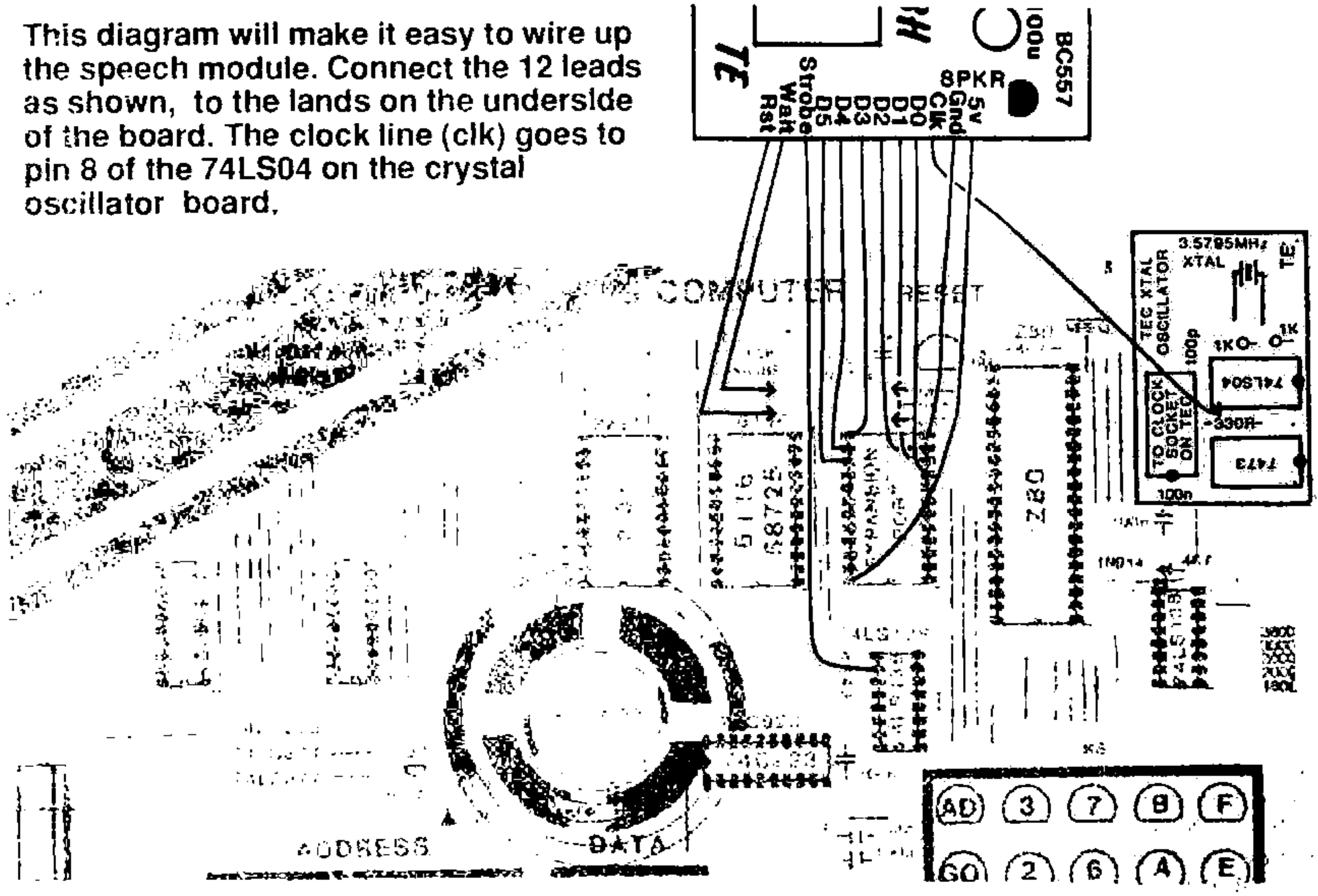
It sounds a little cut-off in the first 'a', so try a longer 'A' i.e. AX (0F) and try again. Enter 0F at 0910 and run the program again. Sounds better now doesn't it!

By following this method, you should be able to come up with any word within a short space of time. Remember, the secret is to sound each letter and syllable out and then search for the best allophone of the group. The sample word provided gives you a context in which the allophone is used. This is useful when deciding between TT1 and TT2 etc.

We also discovered that it was much easier to produce an understandable word if you used the slang way of saying it. The speech module always produces the same type of sound for any given allophone, so if you stick to spelling only, then the words always come out very strange. If you use slang then you will find that the resulting word is much easier to understand.

A perfect example of this came up when we first started work on the project. We bought our first sample chip from Tandy. It came with a list of words and full specification data. When the project was working, we started trying some given examples, and although the examples were recognizable, they were not very clear. Then Ross said to try the slang pronunciation. Voila! perfect. The words which were before just average became clearer and much more recognizable.

This diagram will make it easy to wire up the speech module. Connect the 12 leads as shown, to the lands on the underside of the board. The clock line (clk) goes to pin 8 of the 74LS04 on the crystal oscillator board.



PAUSES AND REPEATING ALLOPHONES

The five pauses are worthy of a separate mention. You must always pause after a word, to make the SPO stop talking. Use a PA1 or a PA2. Use PA3 or PA4 between sentances. Refer to the following table for when to use PA1, PA2, and PA3 DURING words.

PA1 Before BB, DD, GG and JH.

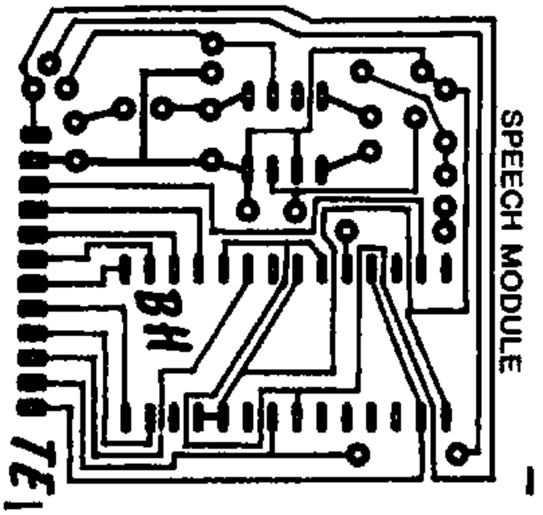
PA2 Before some BB, DD, GG and JH.

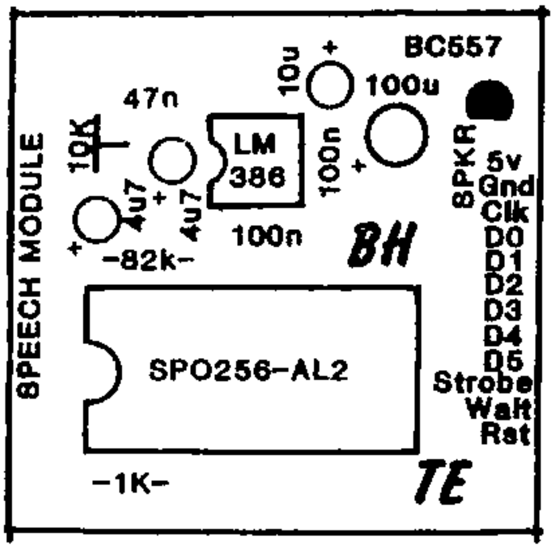
PA3 before PP, TT, KK, and CH.

Begin by inserting the resistors. Solder them in and cut their leads short. Next insert the Capacitors, observing polarity with the Electrolytics.

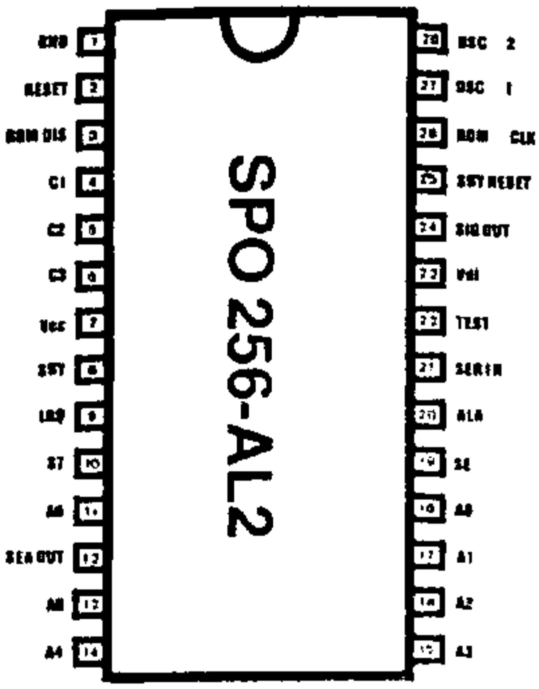
Insert and solder the trimpot, then finally the transistor. Turn the trimpot fully towards the SPO - this is full volume and should be set here until testing is complete.

Check to see that you have a BC557 and insert it according to the 'D' on the overlay. Lastly insert the two IC sockets and plug the chips in, being careful to orientate pin one with the mark on the PC and avoid touching the pins of the SPO256A.





The speech board is very simple. Don't forget you will need the crystal oscillator project to get the 3.58MHz clock line.



The pin out ol the SPO-256-Al2 allophone chip.

Strip 6 wires from the ribbon cable, then connect the remaining 8 between the data lines and the DIP header. Connect power with the last two strands. Follow the diagram and you can't go wrong. Separate 4 of the remaining wires into individual lengths and solder into the 4 remaining holes on the module.

Attach a matrix pin connector to the other end of each wire for connection to the TEC. Heatshrink each connector with the tubing supplied. A note on heatsbrinking: Don't skip this section because you think it's a waste of time or too hard to do. Heatshrinking the connectors strengthens them and the wire is

A repeating allophone is one which can be spoken twice and flow along. i.e. EY EY produces 'AY pause AY', while FF FF produces one long 'Ffff'. Only 10 of these 64 allophones are repeatable like this. They are: IH EH AE UH AO AX AA FF TH & SS. Use these allophones in preferance to long timed syllables like SH in SHirt, WE in tWEnty, or SH in leaSH.

CONSTRUCTION

Although a simple project, care should be taken to ensure that a good job is done, so do not rush. Lay all the parts out in front ol you on a piece of paper or cardboard (Not the High - Low shagpile of the living rooml) and check to see that you have been supplied with everything.

TEST PROGRAM

```
0900  21 10 09  LD HL,0910       HL = Points to start of table.
0903  7E        LD A,(HL)        Get next Allophone.
0904  FE FF     CP FF            End of table ?
0906  28 05     JR Z,090D        Yes, HALT.
0908  D3 05     OUT (05),A       Speak allophone.
090A  23        INC HL           Next allophone.
090B  18 F6     JR,0903          Say next ...
090D  76        HALT             EOT, stop until key pressed.
090E  18 F0     JR,0900          Key pressed, say again.

0910  0D 17 17 02 2A 0C 2C 04    Your allophones are entered
0918  04 2A 0F 10 00 31 16 0D    from 0910 onwards.
0920  33 04 04 FF                this says 'TALKING COMPUTER'

0910  1B 07 2D 35 00 36 07 2F    Here is another greeting
0918  04 06 00 1A 10 00 12 13    message.
0920  00 0D 13 03 13 03 37 13
0928  03 08 18 10 09 31 16 11    The TEC introduces itself
0930  33 04 04 04 38 20 00 30    here!
0938  0C 1D 37 09 13 32 04 FF
```

much less likely to break off. If you always melt the wire when shrinking over a candle, then try using the BARREL not the tip of your soldering iron. This gives you a better controlled heat source and a neat job can be done on those small connections.

The last two lengths of wire connect to the speaker. Wire these up and the board is complete. Now for connection to the TEC. You will need to have your crystal oscillator inserted. If you do not currently own a crystal oscillator, you must purchase one with a 3.58MHz crystal. If you have a different frequency crystal fitted, it must be around 3.2 - 4.0MHz otherwise the sound will be too high or low pitched. A 2MHz or 8MHz crystal will not suffice.

Insert a PC pin in port 5 pad, a second pin in the board for the WAIT line, and a third pin in the board for the RESET line. Most users will already have done so, but if not, see the wiring diagram for the three pin locations.

The other pin you will have to connect as best you can. To tap the 3.5MHz signal DO NOT connect to pin 6 of the Z80. This is because the crystal's frequency is divided by two before reaching the TEC board. Instead, solder a PC pin onto pin 8 of the 74LS04 on the crystal oscillator PC. This is the 3.5MHz clock output.

TESTING

Plug everything together and power up. If your TEC locks up or the unit makes strange sounds, remove power and go to the section on troubleshooting. Your TEC should start up as normal, with the unit deadly quiet. Enter the TEST PROGRAM and you should be greeted with a message. Listen carefully and let your hearing adjust to the metallic pitch. If all you can hear is junk, check your program, then if still no go, proceed to the troubleshooting section.

If the test program produces recognisable output, try the other examples and then try making up a few words of your own. You will soon find that you can say just about any word, once you get the right allophones.

There can be hours of fun even getting it to correctly pronounce your name. 'Paul' is easy enough, but what about 'Vouzopolous'?? or even common words like 'construction' and 'calculator'?? With such a versatile unit, the sky's the limit.

ALLOPHONE REFERENCE TABLE

NUMBER	ALLOPHONE	DURATION	SAMPLE
00	PA1	10 ms	PAUSE
01	PA2	30 ms	PAUSE
02	PA3	50 ms	PAUSE
03	PA4	100 ms	PAUSE
04	PA5	200 ms	PAUSE
05	OY	420 ms	Boy
06	AY	260 ms	Sky
07	EH*	70 ms	End
08	KK3	120 ms	Comb
09	PP	210 ms	Pow
0A	JH	140 ms	Dodge
0B	NN1	140 ms	Thin
0C	IH*	70 ms	Sit
0D	TT2	140 ms	To
0E	RR1	170 ms	Rural
0F	AX*	70 ms	Succeed
10	MM	180 ms	Milk
11	TT1	100 ms	Part
12	DH1	290 ms	They
13	IY	250 ms	See
14	EY	280 ms	Beige
15	DD1	70 ms	Could
16	UW1	100 ms	To
17	AO*	100 ms	Aught
18	AA*	100 ms	Hot
19	YY2	180 ms	Yes
1A	AE	120 ms	Hat
1B	HH1	130 ms	He
1C	BB1	80 ms	Business
1D	TH*	180 ms	Thin
1E	UH*	100 ms	Book
1F	UW2	260 ms	Food
20	AW	370 ms	Out
21	DD2	160 ms	Do
22	GG3	140 ms	Wig
23	VV	190 ms	Vest
24	GG1	80 ms	Got
25	SH	160 ms	Ship
26	ZH	190 ms	Azure
27	RR2	120 ms	Brain
28	FF*	150 ms	Food
29	KK2	190 ms	Sky
2A	KK1	160 ms	Can't
2B	ZZ	210 ms	Zoo
2C	NG	220 ms	Anchor
2D	LL	110 ms	Lake
2E	WW	180 ms	Wool
2F	XR	360 ms	Repair
30	WH	200 ms	Whig
31	YY1	130 ms	Yes
32	CH	190 ms	Church
33	ER1	160 ms	Fir
34	ER2	300 ms	Fir
35	OW	240 ms	Beau
36	DH2	240 ms	They
37	SS*	90 ms	Vest
38	NN2	190 ms	No
39	HH2	180 ms	Hoe
3A	OR	330 ms	Store
3B	AR	290 ms	Alarm
3C	YR	350 ms	Clear
3D	GG2	40 ms	Guest
3E	EL	190 ms	Saddle
3F	BB2	50 ms	Business

* = Repeating Allophone.

BASIC DICTIONARY

0	2B 3C 35
1	30 0F 0B
2	0D 1F
3	36 27 13
4	28 17 17 27
5	28 06 23
6	37 0C 29 37
7	37 37 07 07 23 0C 0B
8	14 11
9	38 06 0B
10	0D 07 07 0B
11	13 2D 07 23 34 0B
12	0D 2E 07 3E 01 23
13	1D 33 0D 13 0B
14	28 17 27 0D 13 0B
15	28 0C 28 0D 13 0B
16	37 0C 29 37 0D 13 0B
17	37 37 07 07 23 0C 0B 0D 13 0B
18	14 11 0D 13 0B
19	38 06 0B 0D 13 0B
A	14
Alarm	0F 2D 3B 10
Alex	1A 2D 07 29 37
Alexandra	1A 2D 07 29 37 1A 0B 15 27 0F
All	17 17 2D
Am	1A 10
Amateur	1A 10 1A 11 31 33
An	1A 0B
And	1A 0B 15
April	14 01 09 0E 0C 2D
Are	3B
At	1A 0D
August	17 1E 22 0F 37 11
B	3F 13
Baby	01 3F 14 01 3F 13
Balhe	3F 14 36
Bather	3F 14 36 33
Be	3F 13
Becky	3F 07 29 13
Bee	3F 13
Beer	3F 3C
Beth	01 3F 07 1D
Birthday	01 3F 33 1D 01 21 07 14
Bite	01 3F 06 03 11
Blank	01 3F 2D 1A 0B 02 29
Bob	01 3F 18 18 01 3F
Bread	1C 27 07 07 00 15

Brett	01 3F 27 07 03 11
Brother	01 3F 27 0F 1D 33
Buy	3F 18 06
By	3F 18 06
Byte	01 3F 06 03 11
Bytes	01 3F 06 03 11 2B
C	37 37 13
Calendar	2A 1A 1A 2D 07 0B 01 21 33
Calling	08 17 3E 2D 0C 2C
Cat	2A 1A 02 0D
Check	32 07 07 02 29
Checked	32 07 07 02 29 0D
Checker	32 07 07 02 2A 33
Checkers	32 07 07 02 2A 33 2B
Checking	32 07 07 02 2A 0C 2C
Checks	32 07 07 02 2A 37
Clock	2A 2D 18 18 02 29
Close	2A 2D 35 37 37
Clown	2A 2D 20 0B
Collide	08 0F 2D 06 36
Computer	2A 0F 10 09 31 16 11 33
Cookie	08 1E 2A 13
Correct	2A 34 07 07 01 29 01 11
Corrected	2A 34 07 07 01 29 01 0D 0C 01 15
Correcting	2A 34 07 07 01 29 01 0D 0C 2C
Correct	2A 34 07 07 01 29 01 11 37
Crane	08 27 14 0B
Crown	2A 27 20 0B
D	21 13
Data	21 18 18 01 11 33
Date	21 14 02 0D
Daughter	21 17 0D 33
Day	01 21 14
December	15 13 00 37 07 30 1C 33
Dennis	21 07 0B 0C 37
Disk	21 0C 37 37 29
Divided	21 0C 23 06 01 21 0C 01 15
Do	03 21 16 1F
Drive	21 27 06 36
Drives	21 27 06 36 2B
E	13
East	13 37 11
Eight	14 11
Eighteen	14 11 0D 13 0B
Elghty	14 0D 11 13
Eleven	13 2D 07 23 34 0B
Emergency	13 10 33 0A 07 0B 37 13
Engagement	07 07 00 0B 24 14 01 0A 10 07 07 0B 01 02 0D
Engages	07 07 00 0B 24 14 01 0A 0C 2B
Engaging	07 07 00 0B 24 14 01 0A 0C 2C
Enrage	07 0B 0E 14 01 0A
Enraged	07 0B 0E 14 01 0A 01 15
Enrages	07 0B 0E 14 01 0A 0C 2B
Enraging	07 0B 0E 14 01 0A 0C 2C
Error	07 07 27 00 33
Extent	07 2A 37 0D 07 07 0B 0D
Exterminate	07 29 37 0D 33 10 0C 00 14 0D
F	07 07 28 28
Father	28 3B 12 33
February	28 07 1C 00 19 1F 34 13
Fifteen	28 0C 28 0D 13 2B
Fifty	28 0C 28 0D 13
Fir	28 34
Five	28 06 23
Fool	28 1E 1E 2D
Force	28 3A 37 37
Four	28 17 17 27
Fourteen	28 17 27 0D 13 0B
Forty	28 17 27 0D 13
Freeze	28 28 0E 13 2B
Freezers	28 28 0E 13 2B 33 2B
Friday	28 27 06 01 21 14
From	28 27 18 10
Frozen	28 28 0E 35 2B 07 0B
G	0A 13
Glenn	01 22 2D 07 2C
H	14 01 02 32
Happy	39 1A 09 13
Has	1B 1B 1A 2B
Have	1B 1B 1A 23
Hello	1B 07 2D 35
Hertz	39 39 34 11 2B
How	39 20
Hundred	39 0F 0F 0B 01 21 27 0C 0C 00 15
I	06
Idiot	0C 01 21 0C 0C 0C 0F 11
In	0C 0B
Input	0C 0B 00 09 1E 11
Is	0C 2B
It	0C 03 11
J	0A 07 14
January	0A 1A 0B 1F 31 34 13
John	0A 18 0B
Julie	0A 31 3E 13
July	0A 1F 2D 06
June	2A 1F 0B
K	2A 07 14
Karen	2A 1A 27 00 07 0B
Kilo	2A 0C 2D 35
Know	38 35
Kristy	08 27 0C 37 11 13
L	07 07 3E
Live	2D 13 23
M	07 07 10
March	10 3B 32
Mark	10 3B 29
May	10 14
Memory	10 07 10 18 27 13
MHz	10 07 24 0F 39 39 34 11 2B
Minute	10 0C 0B 0C 02 0D
Minutes	10 0C 0B 0C 02 0D 2B
Modem	10 35 01 21 07 10
Monday	10 0F 0F 0B 01 21 14
Month	10 0F 0B 1D 1D
Mother	10 0F 36 33
My	10 06
N	07 07 0B
Name	38 14 10
Naughty	38 17 17 02 11 13
Nine	38 06 0B
Nineteen	38 06 0B 0D 13 0B
Ninety	38 06 0B 0D 13
No	38 35
November	38 35 00 23 07 10 1C 33
O	35
October	18 29 00 11 35 1C 33
Of	18 23
On	18 0B
One	30 0F 0B
Or	3A
Our	20 33
P	09 13
Past	09 3B 37 0D
Penelope	01 02 09 07 0B 07 2D 35 09 13
Penny	01 02 09 07 0B 13
Point	09 05 0B 11
Q	2A 31 1F
R	3B
RAM	27 01 1A 1A 10
Rebecca	0E 33 3F 07 02 08 3B
Ross	0E 18 37 37
S	07 07 37 37
Saturday	37 37 1A 02 0D 33 21 14
September	37 07 09 11 07 10 1C 33
Seven	37 37 07 07 23 0C 0B
Seventeen	37 37 07 07 23 0C 0B 0D 13 0B
Seventy	37 37 07 07 23 0C 0B

	0D 13
Sister	37 37 0C 37 0D 33
Six	37 0C 29 37
Sixteen	37 0C 29 37 0D 13 0B
Sixty	37 0C 29 37 0D 13
Son	37 0F 0B
Sound	37 20 0B 15
South	37 37 20 1D
Space	37 09 14 37
Speech	37 09 13 32
Statement	37 01 11 14 01 11 10 07 0B 11
Sunday	37 37 0F 0F 0B 02 21 14
T	0D 13
Talker	0D 17 17 01 29 33
Talking	0D 17 17 02 2A 0C 2C
Television	0D 07 2D 0C 23 0C 37 0C 18 0B
Ten	0D 07 07 0B
Test	0D 07 37 01 11
Testing	0D 07 37 01 11 0C 2C
The	12 13
There	36 07 2F
Thirteen	1D 33 0D 13 0B
Thirty	1D 33 0D 13
This	12 0C 37
Thousand	1D 20 2B 1A 0B 15
Three	36 27 13
Thursday	1D 34 2B 01 21 1A 14
Tim	0D 1C 10
Time	0D 06 10
To	0D 1F
Today	0D 1F 21 14
Tuesday	0D 31 2B 01 21 14
Twelve	0D 2E 07 3E 01 23
Twenty	0D 2E 07 0B 0D 13
Two	0D 1F
U	31 1F
V	23 13
Vision	23 0C 26 0C 0C 18 0B
W	21 0F 01 3F 3E 1F
Want	2E 18 0B 02 11
Wednesday	2E 07 07 0B 2B 01 21 14
What	30 18 02 11
Who	39 1E 1F
With	30 0C 1D
X	07 07 02 29 37 37
Y	2E 06
Year	19 3C
Yes	19 07 37 37
You	19 1F
Your	19 3A
Z	2B 07 02 15
Zero	26 13 27 35

IF IT DOESN'T WORK

If your speech unit does not work, DON'T PANIC. Firstly, check your wiring. Most errors are in wiring, causing the TEC to lock up. Look for obvious faults like shorts, dry joints, components of wrong value or orientation. Check that your chips are inserted correctly - pin one of each chip faces AWAY from the off-board wires.

If you bought your parts from all over the place, make sure you get a SPO256A-AL2 device. Other suffix numbers are not acceptable.

Check that the trimpot is turned all the way towards the SPO256A - full volume. You can temporarily short between the collector and the emitter of the BC557, to turn the amplifier on fully. This should produce a lot of hiss, and touching pin 3 of the LM386 should produce a buzzing sound.

Check that you have +5v on each chip, and that the SPO's reset pin (pins 2 and 25) are normally HIGH, and that they follow the reset pin of the Z80 (pin 26).

If all you get is garbage then you probably have the data lines wired around the wrong way. Check against the wiring diagram, and have a friend check it as well. Look for pins bent up under the SPO and not connecting with the IC socket. Check the program through and make sure that you are sending it the correct data.

If you are totally lost, give us a call. Sometimes we can solve a problem straight away, and most times within a few minutes. If all else fails, we offer a repair service. Costs are:

Basic repair $ 7.00
SPO256A replacement $15.00
Postage $ 3.00

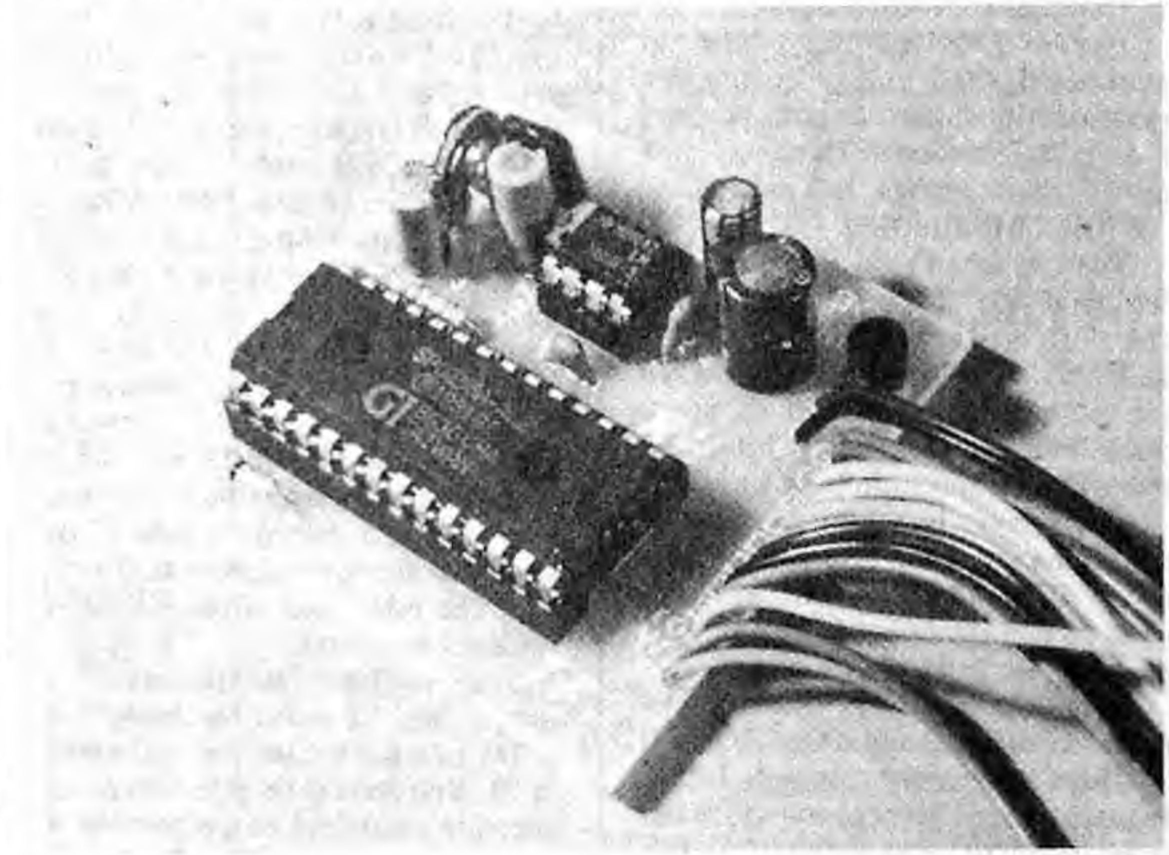
If your SPO256A-AL2 is damaged, you will be charged extra due to its high replacement cost.

MODIFICATIONS

If you don't intend to fit a crystal oscillator to your TEC, you can put a crystal on the speech board. Simply fit the crystal across pins 27 and 28 of the SPO256A. Then fit a 27p between pin 27 and ground, and a 27p between pin 28 and ground. This enables the SPO's internal oscillator. We did not include this on the basic board because we wanted to keep the price as low as possible, in order to counter balance the cost of the SPO256A. We reasoned that most people will change over to JMON, therefore purchasing a crystal oscillator anyway.

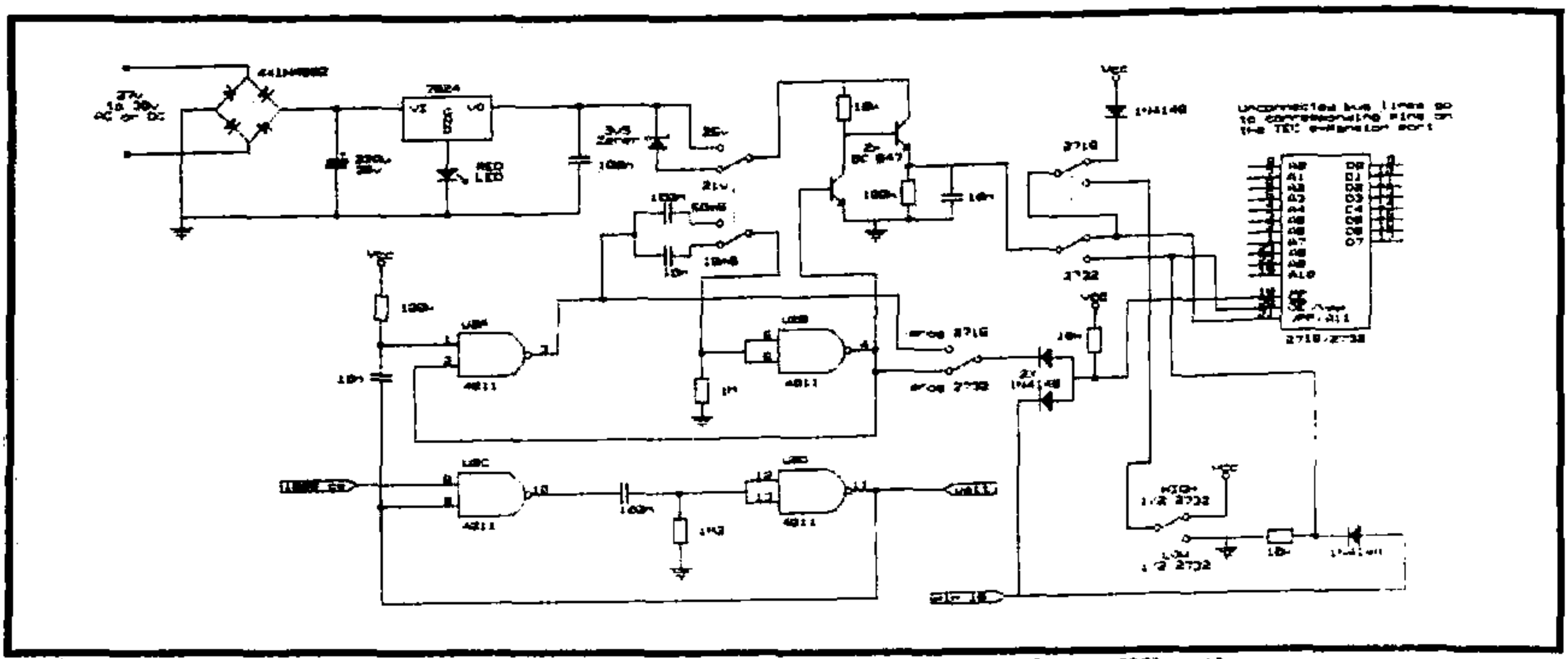
If you find that you are using long silent periods between words, you may find that you can hear an annoying click from the speaker as the LM386 gets switched. This is because the 10u capacitor is too low in value. Increase this capacitor to 22u or 47u and the problem should go away.

If you need to make the output louder, change the 4u7 between pins 1 and 8 of the LM386 to 10u. This increases the gain of the LM386 to 200. ●



EPROM PROGRAMMER REVISITED

Parts $2.30



Circuit diagram showing all corrections and modifications

CIRCUIT DIAGRAM CORRECTION

A mistake has been made with the circuit diagram on page 20 in issue 13.

The 100k resistor between pins 8 and 1 of the 4011 does not exist on the board and pin 8 is actually directly connected to the ROM SELECT LINE. It is not coupled through the 10n capacitor (via the 100k mentioned above) as shown.

CIRCUIT UP-GRADES

If your EPROM programmer is working ok and you're completely satisfied with its performance, perhaps it is best left alone. There are two modifications though, that are HIGHLY RECOMMENDED:

The first is the 100k resistor on the left-hand side of the EPROM socket (next to a diode) SHOULD BE REDUCED to 10k. This will allow for far more reliable readings (if yours doesn't read at all or very poorly, then this will almost certainly fix it).

The second is a 10n greencap is connected across the 100k resistor next to the EPROM socket on the right-hand side of the board (when looking at it from the top).

This 10n greencap is to prevent spikes from damaging the EPROM.

There are some other very handy mods to make. This next one will make it possible to read from 2732 (4k EPROMs) without baving to slide the switch across. The BIG advantage of this is that it is possible for the software to read from the 2732 just after you have programmed each location. The software can then diagnose a failure and re-try or abort quickly. The software routine is provided below which will do this for either a 2716 or 2732.

Three additional parts are required for this mod. They are two IN 4148 diodes and a 10k resistor.

The first diode is soldered between the DIP-HEADER and the EPROM socket. The cathode (the end with the band on it) is soldered to pin 18 of the DIP-HEADER and the anode is soldered to pin 18 of the EPROM socket. Next, the track running between pin 18 of the EPROM socket and the middle of (program 2716 read 2732)/program 2732 switch is cut. The anode of the second diode is soldered to the pin 18 side of the cut and the cathode is soldered onto the middle terminal of the switch. One end of the 10k resistor is soldered to the anode side of the second diode (the end connected to pin 18). The other end of the resistor is soldered to ground.

Once you have fitted this modification, it may be tested by fitting a 4k ROM into the socket and addressing 1000. You should be able to read the contents regardless of the position of the read/program 2732 switcb. The high/low switch is still used to select which half of the EPROM you wisb to read and the read/program switch is used to select the type of EPROM you wish to program.

PARTS LIST

(For all mods)

2 - 10k

1 - 10n greencap
1 - 100n greencap

2 - 1N4148 diodes
1 - 3v9 Zener diode

1 - DPDT switch
1 - 10cm tinned copper wire
1 - 10cm hook-up wire

The next mod is a little more involved but is an important one if you wish to re-program some of the EPROMs supplied by TE.

The programming requirements of some types of more modem (but now obsolete) EPROMs are not compatible with the current set-up of the EPROM programmer. This mod allows the EPROM programmer to be used with a wider variety of EPROMs. The mod does this by switching the programming voltage from 25v to 21v and reduces the programming pulse from 50mS to 10mS.

The parts required for this mod are: one DPDT switch, one 10n greencap, one 100n greencap, a 3v9 zener diode

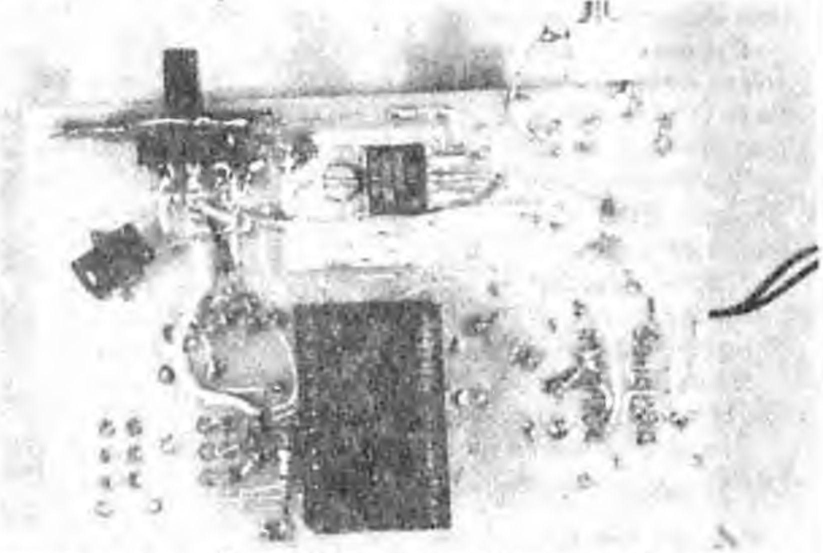
cont. P 45 . . . .

. . . . .from P. 32.

and some hook-up wire. To start, mount the switch on the bottom of the PCB by drilling two holes and wrapping tinned copper wire around the switch (see photo). Next cut the track between the output of the 24v regulator and the transistor switching block. The bottom middle terminal of the switch is connected to the transistor side of the cut. Connect the bottom right-hand side terminal to the regulator output and also solder the cathode end of the zener to this junction. The anode end of the zener is soldered to the bottom left-hand side of the switch. The zener, which is connected between the regulator and the high voltage switching section, drops the programming voltage by about 4v.

This completes the voltage switching section. Below is the programming pulse length mod.

The photograph on the right shows how the parts on our prototype are mounted. The description of the parts placement in the text, corresponds to this photo.



Remove the 100n greencap on the extreme left-hand side of the board (top view). Solder one end of the new 100n to the top right-hand side of the switch. Take the 10n cap and solder one side of this to the top left-hand side of the switch. The other ends of the caps are soldered together and a jumper is also soldered onto this junction. The jumper is then soldered to pin 3 of the 4011. Another jumper is soldered between pin 6 of the 4011 and the top middle terminal of the switch.

When the switch is in the right-hand position (top view), the EPROM programmer is set up for the modern 21v/10mS EPROMs.

One of these types of EPROM is being supplied by TE. It can be identified by the following markings:

TMS
2732A-25JL
LHE XXXX (DATE CODE)

To increase the reliability of the programmer, another mod is suggested. Follow the track from the ROM select line to where it joins the 10n cap. Cut both the tracks that join to the cap at this junction. Then run a link from the ROM select input pad to pin 8 of the 4011. Now run a jumper from the wait pin to the now isolated end of the greencap.

This mod slightly delays the programming pulse to the EPROM by triggering it from the wait line, not the input ROM select line.

The software for burning EPROMs provided in issue 13 is only very basic. There is one VERY IMPORTANT ADDITION to make to the issue 13 software. After you have loaded BC, DE and HL, as described in issue 13, add the following:

```
XOR A (AF)
LD I,A (ED 47)
JUMP 0700 (C3 00 07)
```

These instructions stop the noise on the expansion port which is a result of several TEC design oversights.

The following software is designed to be burnt into either a MON-1 or MON-2 EPROM at 0700.

The software is JUMP TO with the "from address" in HL, the "to address" in DE and the number of bytes in BC.

Before it attempts to burn into the EPROM, it checks that the area to be programmed contains only FF's. If not, the routine displays an "F", for FULL in the data display and halts. You may continue on and burn the EPROM by hitting "GO". Each location is checked after it is burnt and if not correct, it is reprogrammed several more times before being aborted. The routine then displays "E" for ERROR.

You must do the "read 2732" mod to program 2732 EPROMS.

An added feature to this software is that it flashes the address being programmed on the TEC display.

EPROM BURNING SOFTWARE

```
0700  AF          XOR A
0701  ED 47       LD I,A
0703  CD 90 07    CALL 0790
0706  7E          LD A,(HL)
0707  12          LD (DE),A
0708  D5          PUSH DE
0709  D9          EXX
070A  D1          POP DE
070B  CB 9A       RES 3,D
070D  D5          PUSH DE
070E  01 F0 0F    LD BC,0FF0
0711  C5          PUSH BC
0712  CD 5A 07    CALL 075A
0715  7B          LD A,E
0716  CD 5A 07    CALL 075A
0719  7A          LD A,D
071A  CD 5A 07    CALL 075A
071D  C1          POP BC
071E  01 10 00    LD BC,0010
0721  C5          PUSH BC
0722  CD 6E 07    CALL 076E
0725  C1          POP BC
0726  0B          DEC BC
0727  78          LD A,B
0728  B1          OR C
0729  20 F6       JR NZ,0721
072B  D1          POP DE
072C  1A          LD A,(DE)
072D  D9          EXX
072E  BE          CP (HL)
072F  20 08       JR NZ,0739
0731  23          INC HL
0732  13          INC DE
0733  0B          DEC BC
0734  78          LD A,B
0735  B1          OR C
0736  20 CE       JR NZ,0706
0738  C7          RST 00
0739  C5          PUSH BC
073A  01 05 00    LD BC,0005
073D  CB DA       SET 3,D
073F  7E          LD A,(HL)
0740  12          LD (DE),A
0741  10 FE       DJNZ,0741
0743  CB 9A       RES 3,D
0745  1A          LD A,(DE)
0746  BE          CP (HL)
0707  20 03       JR NZ,074C
0749  C1          POP BC
074A  18 E5       JR 0731
074C  0D          DEC C
074D  20 EE       JR NZ,073D
074F  C1          POP BC
0750  3E C7       LD A,C7
0752  D3 02       OUT (02),A
0754  3E 01       LD A,01
0756  D3 01       OUT (01),A
0758  76          HALT
0759  C7          RST 00
075A  F5          PUSH AF
075B  CD 63 07    CALL 0763
075E  F1          POP AF
075F  0F          RRCA
```

```
0760  0F         RRCA
0761  0F         RRCA
0762  0F         RRCA
0763  E6 0F      AND 0F
0765  21 B0 07   LD HL,07B0
0768  85         ADD A,L
0769  6F         LD L,A
076A  7E         LD A,(HL)
076B  02         LD (BC),A
076C  03         INC BC
076D  C9         RET
076E  21 F0 0F   LD HL,0FF0
0771  06 06      LD B,06
0773  0E 01      LD C,01
0775  7E         LD A,(HL)
0776  D3 02      OUT (02),A
0778  79         LD A,C
0779  D3 01      OUT (01),A
077B  0E 40      LD C,40
077D  0D         DEC C
077E  20 FD      JR NZ,077D
0780  07         RLCA
0781  4F         LD C,A
0782  AF         XOR A
0783  D3 01      OUT (01),A
0785  23         INC HL
0786  10 ED      DJNZ,0775
0788  C9         RET
0789  FF         RST 38
078A  FF         RST 38
078B  FF         RST 38
078C  FF         RST 38
078D  FF         RST 38
078E  FF         RST 38
078F  FF         RST 38
0790  D5         PUSH DE
0791  C5         PUSH BC
0792  CB 9A      RES 3,D
0794  1A         LD A,(DE)
0795  FE FF      CP FF
0797  20 09      JR NZ,07A2
0799  13         INC DE
079A  0B         DEC BC
079B  78         LD A,B
079C  B1         OR C
079D  20 F5      JR NZ,0794
079F  C1         POP BC
07A0  D1         POP DE
07A1  C9         RET
07A2  3E 47      LD A,47
07A4  D3 02      OUT (02),A
07A6  3E 01      LD A,01
07A8  D3 01      OUT (01),A
07AA  76         HALT
07AB  18 F2      JR 079F

 07B0 EB 28 CD AD 2E A7 E7 29
      EF 2F 6F E6 C3 EC C7 47
```

PRINT-2 AND PRINT-3 SOFTWARE

With the changes to the keyboard handler routines in both MON-2 and JMON, an up-dated printer ROM has been produced.

The new software is burnt into the same ROM at higher locations. When MON-2 was released, an up-dated ROM called print-2 was included in the printer interface kits. This gave you the same routines with an altered keyboard section. It was also a little more fancy as it showed the start address on the LED display as you typed it in. Unfortunately, Print-2 did not include a "dump string at 0900" routine to replace the dump from 0800 which is now unusable as MON-2 uses 0800 for its variable storage.

With the advent of JMON, the same arrangement has been used. The JMON printer routines are located higher again, so in the one ROM you have the printer software for all three MONitors. The list routine for JMON is an improvement on both earlier software packages, as JMON's routine uses the perimeter handler to allow you to enter both a START and END address. Print-3 includes a "dump from 0900" routine which can be used with MON-2.

The ROM with the JMON routines in it is called PRINT-3 and is supplied with the printer interface as standard.

JMON's hex dump routine is at 1A20, the typing routine at 1AA0 and the "dump string at 0900" routine is at 1AC0.

Below is a dump of PRINT-3. Burn the additional section(s) in PRINT-1/2 ROM.

The graphic demonstration routines in PRINT-1 will work with all MONitors.

```
1800  3E 0D D3 06 3E 0A D3 06
      76 ED 57 17 17 17 17 57
1810  CD 5D 18 76 ED 57 82 57
      CD 61 18 76 ED 57 17 17
182D  17 17 5F CD 5D 18 76 ED
      57 83 5F CD 61 18 C3 49
1830  18 3E 0D D3 06 3E DA D3
      06 7A CD 5D 18 7A CD 61
1840  18 7B CD 5D 18 78 CD 61
      18 06 08 3E 20 D3 06 1A
1850  CD 5D 18 1A CD 61 18 13
      10 F1 C3 31 18 1F 1F 1F
1860  1F 21 6C 18 E6 DF 85 6F
      7E D3 06 C9 30 31 32 33
1870  34 35 36 37 38 39 41 42
      43 44 45 46 FF FF FF FF
1880  21 00 08 7E FE FF 20 05
      3E 11 D3 06 C7 D3 06 23
1890  18 F1 FF FF FF FF FF FF
      FF FF FF FF FF FF FF FF
18A0  21 C3 18 7E FE FF 28 05
      D3 06 23 18 F6 06 0A 21
18B0  CF 18 7E FE FF 28 05 D3
      06 23 18 F6 10 F1 3E 11
18C0  D3 06 C7 0D 0A 0A DA 0A
      0A 0A 12 43 30 0D FF 49
18D0  2C 44 33 32 30 2C 30 0D
      4D 31 32 30 2C 30 0D 44
18E0  38 30 2C 2D 31 36 30 0D
      4D 32 32 30 2C 2D 38 30
18F0  0D 44 31 36 30 2C 2D 38
      30 2C 31 34 30 2C 2D 31
```

```
1900  36 30 2C 32 30 30 2C 2D
      31 36 30 0D 4D 31 35 30
1910  2C 2D 31 32 3D DD 44 32
      3D 30 2C 2D 31 32 30 0D
1920  4D 33 32 3D 2C 2D 38 30
      0D 44 32 36 3D 2C 2D 38
193D  30 2C 32 34 30 2C 2D 31
      36 30 2C 33 3D 30 2C 2D
194D  31 36 3D 0D 4D 33 36 30
      2C 2D 31 32 30 0D 44 34
1950  30 30 2C 2D 31 32 3D 0D
      4D 34 36 30 2C 2D 38 30
1960  0D 44 34 34 30 2C 2D 31
      36 30 0D 4D 32 2C 2D 32
1970  0D 43 33 0D FF FF FF FF
      FF FF FF FF FF FF FF FF
1980  76 ED 57 E6 0F 17 17 17
      17 57 76 ED 57 E6 0F 82
1990  D3 06 18 EC FF FF FF FF
      FF FF FF FF FF FF FF FF
```

The next block is the PRINT-2 additions:

```
19A0  76 3A E0 08 E6 0F 17 17
      17 17 57 76 3A E0 08 E6
1980  0F 82 D3 06 18 EA FF FF
      FF FF FF FF FF FF FF FF
19C0  3E 0D D3 06 3E 0A D3 06
      3E 29 21 D8 08 06 06 77
19D0  23 10 FC CD 00 1A 32 D8
      08 CD 00 1A 32 D9 08 CD
19E0  0D 1A 32 DA 08 CD 00 1A
      32 DB 08 CD D9 01 CD 89
19F0  02 5D 59 C3 31 19 FF FF
      FF FF FF FF FF FF FF FF
1A00  3E FF 32 E0 08 CD A0 02
      3A E0 08 FE FF 28 F6 E6
1A1D  0F C6 FF CD 70 01 06 01
      C9 FF FF FF FF FF FF FF
```

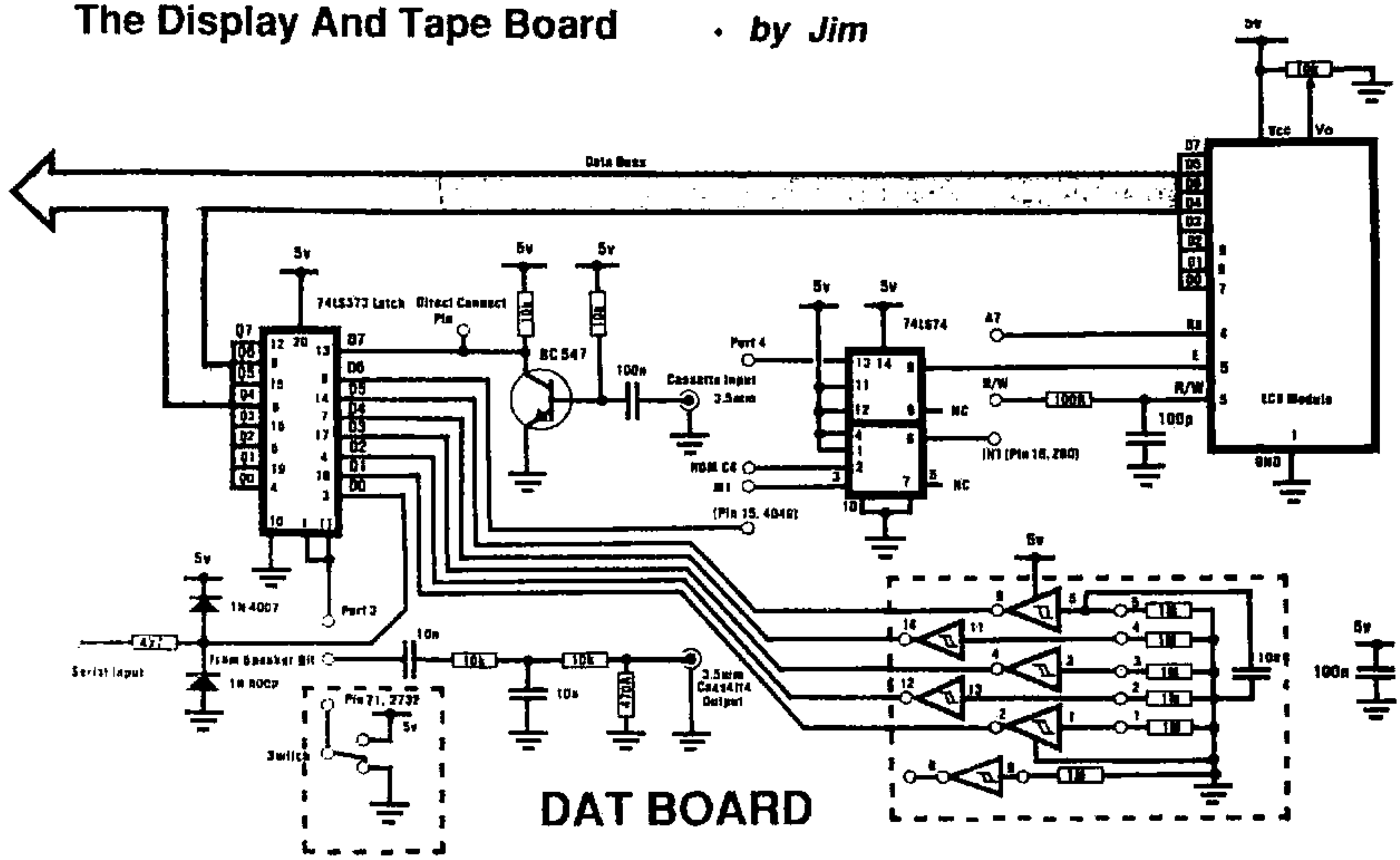
Below is PRINT-3 additions:

```
1A20  21 34 1A 11 80 09 01 0A
      00 ED B0 21 00 00 22 9C
1A3D  08 C3 44 00 00 00 3E 1A
      99 08 00 01 5D 1A 04 A7
1A40  04 C7 04 EB FF FF FF FF
      FF FF FF FF FF FF FF FF
1A5D  3E DD D3 06 2A 98 08 7C
      CD 82 1A 7D CD 82 1A 06
1A60  08 C5 3E 20 D3 06 7E CD
      82 1A 23 C1 10 F3 3E 0D
1A70  D3 06 3E 0A D3 06 ED 5B
      9A 08 E5 B7 ED 52 E1 38
1A80  D6 C9 F5 0F FF 0F 0F CD
      8B 1A F1 E6 0F C6 90 27
1A90  CE 40 27 D3 06 C9 FF FF
      FF FF FF FF FF FF FF FF
1AA0  CF E6 0F 07 07 07 07 57
      CF E6 0F 82 D3 06 18 FD
1AB0  D3 06 18 EC FF FF FF FF
      FF FF FF FF FF FF FF FF
1AC0  21 00 09 7E FE FF 20 05
      3E 11 D3 06 C7 D3 06 23
1AD0  18 F1 FF FF FF FF FF FF
      FF FF FF FF FF FF FF FF
```

TE

THE DAT BOARD

The Display And Tape Board · *by Jim*



DAT BOARD

This board will change the way you program for ever. The DAT BOARD is perhaps the most vital addition to the TEC ever. Not just a part time "add on," but rather a permanent addition to your TEC.

Once you start using it, we think you'll agree.

The name "DAT" is an acronym for Display And Tape. While others brawl over "their" DAT. (have you seen one?), we have quietly slipped in the back door with our version.

The DAT BOARD provides these functions:

* 16x2 LCD display.
* Cassette tape I/O interface.
* Single stepper module.
* 5 Buffered and latched input bits.
* 1 Inverter for general use.
* Diode clipped input line. (For RS232 input)
* MON select switch.

PORT 3

Port 3 addresses an input latch. Below is a break-down of the bits on port 3.

BIT#		
0	-	Serial in
1	-	input 1
2	-	input 4
3	-	input 2
4	-	input 5
5	-	input 3

The above are the inputs from the 74C14.

6	-	key pressed signal.
7	-	Tape input.

CONNECTION

Up until now, TEC add-on's have been connected via the expansion port. We wished to avoid this as there are too many devices cluttering up this area already. The search was on for a better place to put our new board. We decided upon the blank area left of the eprom, because it is common to all TEC's and has up until now not been used by anything else.

But there's nothing to connect to there! I hear you say. Well not quite, Simply solder a cut-up I.C. socket onto the links and you have an (almost) instant data buss socket. The DAT BOARD has a set of feed downs that push into the sockets and serve the dual purposes of connection and fixation.

PARTS LIST

1	-	100R
1	-	470R
4	-	10k
1	-	10k mini trimpot
1	-	100p ceramic
2	-	10n greencaps
3	-	100n greencaps
1	-	BC 547
1	-	74LS373
1	-	74LS74
1	-	5mm LED (for trimpot handle)
2	-	3.5mm sockets
1	-	20 pin IC socket
2	-	14 pin sockets (one to cut-up)
1	-	20cm 12 way ribbon cable
1	-	50cm figure-8 shielded cable
1	-	1.2 metres hook-up wire
4	-	3.5mm mono plugs
1	-	100cm tinned copper wire
1	-	Female matrix connector
3	-	32mm x 2.5mm bolts
9	-	2.5mm nuts
1	-	16 character x 2 line LCD*
1	-	DAT PC Board

*Don't Forget: The LCD display can be bought separately.

The feed downs are simply lengths of stiff wire soldered to the underside of the P.C. that extend about 1 to 1.5 cm down to push into the I.C. sockets.

The fixing of the DAT BOARD is also aided by three "stand offs," in the form of three bolts with nuts to tighten against the board. These may extend through the TEC board if you want as there is no track work underneath.

CONSTRUCTING THE DAT BOARD

Originally, the kit of parts for the DAT BOARD was going to be supplied in two sections. We have changed our minds since, but have decided to present these construetinn notes unchanged. The first thing to do, is to fit ALL the links, regardless of what section you are constructing.

If you have already built the TAPE and keyboard section and/or are now constructing the LCD/SINGLE STEPPER interfaces then skip ahead to the respective notes. Once you have built the LCD section skip back to the notes on inserting the feed downs, stand-offs and control buss leads.

THE TAPE AND LATCH SECTION

Most the components for the TAPE SECTION are fitted on the bottom left corner of the board. The exceptions being a 100n greencap, that goes on the middle left of the board, the latch chip and its socket. Fit these in the order you prefer and then solder a short piece of tinned copper wire in the hole marked "SP."

This is where the female matrix connector will slide on. If you are wondering why we recommend a piece of tinned wire instead of a male matrix pin, the reason is that the force needed to push a female over a male matrix pin is far to great to be healthy for the TEC or DAT PCBs. (The keyboard is destructive enough). The tinned wire can be tinned again to give just the right fitting diameter, if required.

After fitting all the components, cut the length of hook-up wire into 4 equal sections. Strip and tin each end of all the lengths. Solder two pieces to the ground strip next to the tape in and tape out pads on the DAT BOARD. The other ends of these wires solder to the top tags of the 3.5mm sockets. Solder the two remaining wires to the tape in and tape out pads. The other ends are soldered to the DIAGONALLY OPPOSITE tags on the 3.5mm sockets. Keep track of which socket the wires are joined to, and mark them accordingly.

Drill two holes large enough for the 3.5mm sockets in the back or side of the RETEX case and fit the sockets in place. Strip the ends of the shielded cable and twist the shield into one strand. Remove the covers of the 3.5mm plugs and slide them onto the figure 8 cables, so they are back to back. Solder the shields to the larger tags on the plugs. The middle conductor is soldered to the smaller tags. Do this for each of the four ends.

Solder a 5cm piece of hook-up wire on the 1K resistor which connects the output latch to the speaker transistor. The wire is soldered on the LATCH SIDE of the resistor. The other end of the wire is soldered to the female matrix connector. This matrix connector slides over the pin marked SP on the DAT BOARD.

Now you are ready to insert the feed downs.

INSERTING THE FEED DOWNS

The feed down are made of stiff tinned wire of about 2cm length. The easiest way to solder these is to solder a continuous length in each hole, and then trim it down afterwards. Do this for all the feed downs and try to get them straight as possible.

The feed downs plug in to a cut-up IC socket soldered across the links near the EPROM. The socket is soldered where the links form a straight line as they disappear into the TEC PCB. (See diagram). If you want, you may make the feed downs longer, remove the links, and permanently solder the DAT BOARD in place. Of course, you will need to put jumpers beneath the board to replace the missing links. This arrangement will provide a far more reliable circuit connection. Make sure you have finished the board COMPLETELY before you do this, as you will not be able to solder underneath the board afterwards.

CONNECTION OF THE CONTROL LINES

There are 10 control lines that are soldered to the bottom of the TEC board. A 20 cm 12 way ribbon cable is used to make all the connections. The ribbon cable is soldered to a row of pads on the DAT BOARD about 2.5cm below the top edge. The ribbon cable is soldered to the BOTTOM SIDE of the DAT BOARD and then drops down between the TEC board and the RETEX case (if you have one).

All the connections to the DAT BOARD are prinled on the solder side of the board while the connections to the TEC are made as per the wiring diagram.

The two 3.5 nım sockets for the tape in/out are mounted in either the back or side of the RETEX CASE. If you do not have a case, then the sockets can be connected with short pieces of wire and left "floating." We do not recommend that you drill holes in either the TEC or DAT boards for the sockets. This is to save the expensive TEC board from the excessive force involved in plugging and unplugging the leads. The best idea is to hold the sockets when inserting the leads.

THE STAND-OFFs

In addition to the feed downs, three bolts act as stand-offs. The head of these bolts sits on the TEC board or, if you wish, you may drill into the board and feed the bolts up through the board. If you have the original TEC-1 board with the 8212 latch chips, the top bolt will not be able to be feed through the board as there is track work associated with the (now aborted) on-board tape interface and battery backed RAM.

If you have drilled the holes, then feed the bolts up from the bottom of the TEC and lock each in place with a nut. A second nut is screwed down to about 1cm off the TEC board on each bolt. This sets the height of the DAT BOARD. The DAT BOARD is then placed over the two bolts and a third nut is tightened onto the DAT BOARD.

If you to not wish to drill into your TEC, which is quite understandable, then place a nut on each bolt and wind it down to about 1cm from the head. Poke the bolts through the the hole in the DAT BOARD and tighten down the second nut.

Next, insert the board and note how high it is off the TEC. Ideally it should be 1.5 to 2cm off the board. Trim the feed downs until you are happy with the height. Adjust the stand-offs until they all sit neatly on the TEC board. Finally, a blob of blu-tack can be used to secure the top stand-off on to the board. This will help keep the DAT BOARD square on the TEC.

TESTING THE LATCH/TAPE INTERFACE

The latch is easily tested by running up JMON. If the keyboard works then the latch is obviously working. You can test each bit of the latch by taking the remaining inputs to ground. These pins are connected to pins 2,4,6,8 and 12 on the 74Ct4 socket and also pin 3 of the latch chip itself. Make sure that you don't have the 74C14 fitted as this may damage the chip.

The following program will echo the latch on the LED display:

```
0900 3E 3F D3 02 DB 03 E6 3F
0908 D3 01 C3 00 09
```

To test the tape, refer to the pages on using the tape system that show how to use and trouble shoot the tape interface.

THE SINGLE STEPPER/LCD INTERFACES

If you are constructing this section before the tape/latch section, you will need to make a modification to the TEC. The mod is to add a 4k7 resistor between pin 15 of the 4049 and pin 10 of the Z80. The purpose of this mod is to route the DATA AVAILABLE SIGNAL to the DATA BUSS. Without this, JMON is unable to read the keyboard. (This mod is described numerous times throughout this issue). The LCD interface consists of just four components. They are a D flip flop, a 100p cap, a 100R resistor and a 10k trimpot. The D flip flop, (that was spare) is configured to act as an INVERTER!! This design saved us from having to use another chip.

The single stepper interface simply uses one half of a dual D flip flop!

CONSTRUCTION NOTES

These 2 interfaces are simple to construct. Just take care with the orientation of the 74LS74 chip. If you have a spare LED on hand then you can solder it onto the trimpot to use as a knob (one is provided in the kit).

FITTING THE LCD

Place the LCD FACE DOWN on the work bench and feed a 5cm length of tinned copper wire into each hole on the LCD. Solder the wires in place and then, starting at one end, trim the wires to form a ramp. This helps you to insert the 14 wires one-at-a-time into the DAT BOARD. The DAT BOARD edge connector is placed at the top of the DAT BOARD and the LCD overhangs the board like a verandah.

Insert the LCD into the DAT BOARD as best you can. A second person with a pair of tweezers could help tremendously in getting each wire down its hole.

After you have fitted the wires into their holes, position the LCD to the height you want. This should be about 1cm to 1.5cm, and carefully solder it in place.

TESTING THE LCD

After you have finished construction and wired the DAT BOARD to the TEC as shown in the wiring diagram, you're ready to go. Fit the board in place and turn the 10k trimpot clockwise when looking at it from the left. Turn it as far as it goes, then turn it back just slightly. This sets the contrast level and if it is not approximately at the position described above, nothing will appear on the LCD.

If you have JMON then fit it into the EPROM socket and power up the TEC. All things being equal, the display will show the following:

```
     0900>xx  xx  xx  xx
     Data   xx  xx  xx  xx
```

If not, the most likely cause is that one of the data lines is not getting to the display. The easiest way to check this is to type in the following:

```
0900 3E 55 D3 04 C7
```

AFTER you have entered this, connect a jumper between port 4 and the wait line of the Z80. When you have done this, hit go.

The TEC should go "dead." Now, with a logic probe, test the edge connector of the LCD. Starting from the right, the logic levels should be: H, L, H, L, H, L, H AND L.

If not, then check all the connections and retry until right. If the connections are right, but there is nothing on the display, check the voltage on pin three of the LCD. This voltage should be in the range of 0.5v to 1v. Adjust the trimpot until you measure this voltage.

Still no luck? Turn off the TEC, hold reset down and turn the TEC back on while still holding down the reset. The top row of the LCD should be dark and the bottom line should be light. If not then there maybe no power getting to the LCD, the contrast voltage may be incorrect (but you have already checked this), or the display has been damaged, they are all tested before they leave TE).

If the top line is dark when power is applied but the display does not respond when reset is released, then put your logic probe on pin 6 of the LCD. Hold down the "+" key and watch the logic probe. Pin three should pulse HIGH each time the TEC beeps. If not then check that you have the wire going to port 4 in the correct place. Check the track work around the 74LS74 chip and the chip itself.

If pin 6 seems ok, then check that the 100p cap is fitted as this is VERY IMPORTANT. Pin 5, the r/w line, should always be pulsing. Check this with the logic probe.

The only other line left to test is the register select (RS). This line is address 7, and the easiest way to check this is with a continuity tester. If the LCD clears when power is applied, but nothing appears on the LCD, then it is odds-on that the cause is address 7 not being wired correctly.

TE REPAIR SERVICE

Still can't get it going? Check it all through again, keep in mind that the most likely cause is a mistake in your wiring. As a VERY last resort (after ringing us) send it in and we'll see what we can do.

Our repair fee is $9.00, plus $2.50 for post and handling. This includes replacement of all parts except the LCD (that was tested before leaving us). Before you send it in, remove the control buss wires (the ribbon cable) from the DAT BOARD. Pack it up securely and send it down. If you want the tape section tested leave the 3.5mm sockets connected.

TESTING THE SINGLE STEPPER

This is easy. With JMON fitted, enter this at 0900:

```
0900: 00 00 00 00 00 C3 00 09
```

Now, press shift 2. The single stepper will show 0900 PC. Press any data key and the single stepper will cycle automatically. The occasional clicking you (may) hear is a result of the interaction of the interrupt response cycle and the decoding of the 74LS138 decoder chip.

If the single stepper doesn't work, then check your wiring as it is doubtful that the 74LS74 chip is faulty.

WHAT THE LCD INTERFACE DOES

The LCD is designed to directly interface tn microprocessors. Unfortunately there are two main types of microprocessor buss timing and the LCD is designed for the wrong type (as far as we are concerned). In order to get the LCD to interface to the Z80, a little bit of juggling with the *timing* is *needed*. The first problem is the the LCD requires an active HIGH Enable signal. This has been achieved by inverting the PORT 4 I/O select line. This inverting is done by the spare D flip flop on the DAT BOARD. By looking at the TRUTH TABLE for the 74LS74, I found that it was possible to configure it as an inverter if I used the CLR pin as the DATA input!

To cut a long story short, the idea worked. Eureka!

The next problem is the LCD requires R/W to be stable on the falling edge of the E signal. If you look at the Z80 timing, you will see that the R/W line and the IORQ change state simultaneously. By the time that IORQ has gated port 4 and the port 4 signal has been inverted, the R/W line will actually change (slightly) before the E line on the LCD!

To overcome this problem, a simple RC network has been placed on the R/W line. This RC delay holds the R/W line stable while the E line goes low. The time we are talking about here is just a fraction of a microsecond, but that is all it takes for the chips in the LCD to accept or reject the in-coming signals.

Another problem is that the LCD requires 2 ports to communicate with the Z80. It also wants to decode the second port itself. This is a common requirement of many peripheral devices, and the solution provided here is also useful for all these.

To give the LCD its second port, and let it decode it for itself, address line 7 has been presented to the LCD. This means that the second port is decoded (by the LCD) on port 84.

DISPLAY CONTRAST

The LCD requires an external voltage to set the contrast level.

The contrast of LCDs varies with temperature and viewing angle. To allow for this, the LCD has an external contrast control. The contrast is controlled by adjusting the voltage on this pin. This is the function of the 10k trimpot, that is wired as a voltage divider.

OPTIONS

Several optional extras can be added to the DAT BOARD. Below is a description of each:

MON SELECT SWITCH

When you add the DAT BOARD, there may not be enough room between the board and the EPROM to fit your MON select switch. If this is the case, provision has been made to fit the switch to the DAT BOARD. Simply install the dotted link and move your switch to the dotted switch position on the DAT BOARD. Run a wire between the pin marked 'ROM P21' and pin 21 of the EPROM.

SERIAL INPUT

The SERIAL INPUT (SI)

This input is for a serial signal, or a RS232 level signal from a printer or RS232 device. This input clips the signal, which can be +/-15V to +/-25V, to safe logic levels.

This signal winds up as D0 on the 74LS373.

THE 74C14

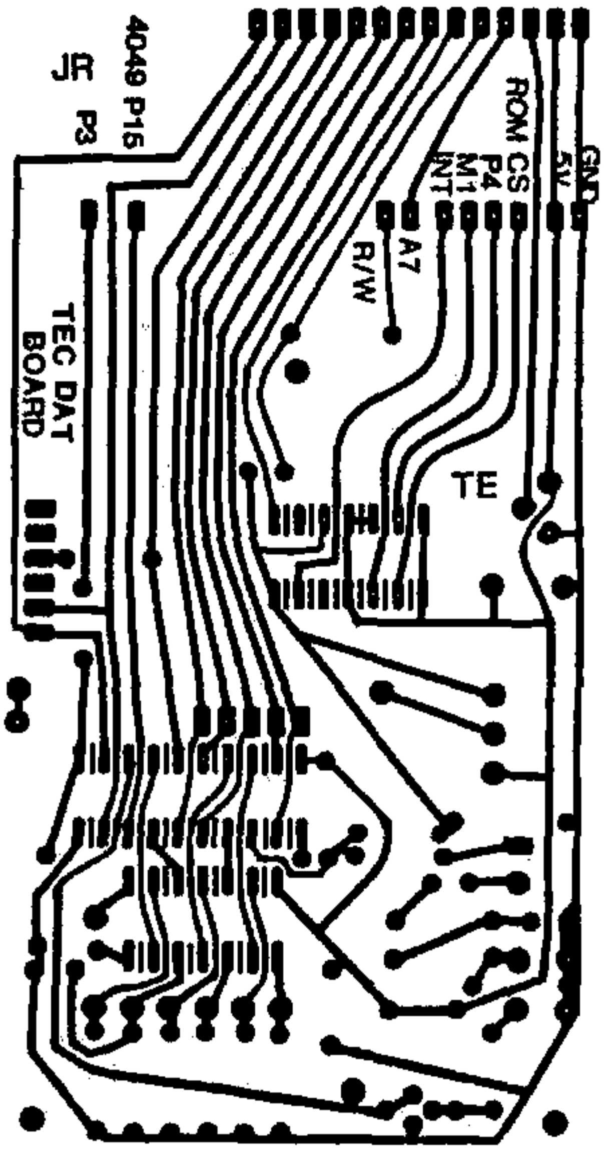
This has been added to increase the versatility of the DAT BOARD. Some possibilities for it include a touch sensitive qwerty key pad, an external time reference, a thermistor controlled oscillator for temperature measurement or just buffered inputs. Nothing permanent has been planned for it, it is mainly for experimentation. We are open to your ideas!

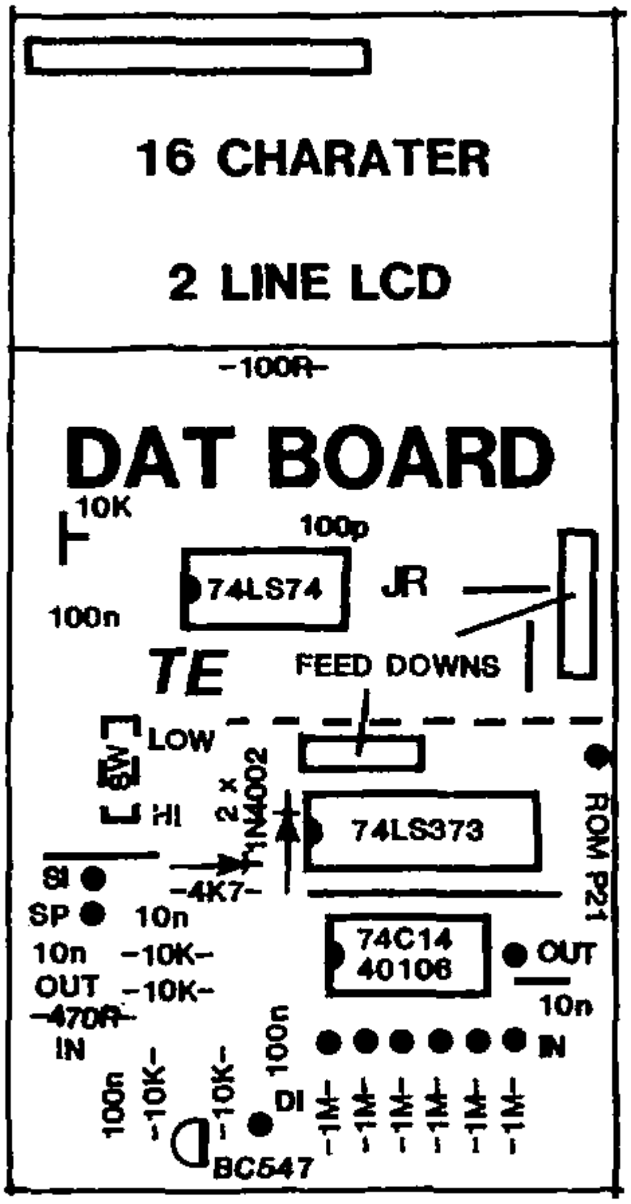
THE DIRECT CONNECT PIN

This is located between the transistor and the 6 x 1M resistors. The purpose of this pin is to allow *direct* connection between two TECs. One TEC can down load to another through the tape software or a serial communication program. (I have a 9600 Baud routine that also talks to IBM's and compatibles).

THE UNUSED INVERTER

The input for the unused inverter is the right most matrix pin on the bottom right-hand side of the DAT BOARD. The output is the matrix pin directly above it.





HOW THE TAPE CIRCUIT WORKS

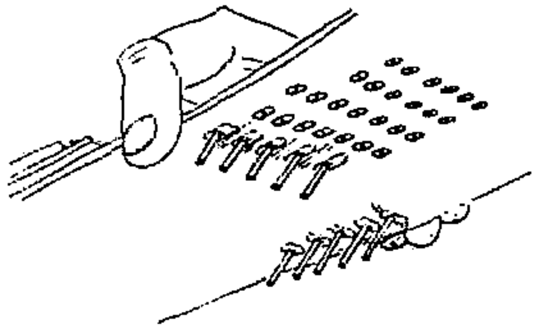
There's not much to describe about the tape circuit as all the hard work is done by software. The output section consists basically of an AC coupled LOW PASS filter with some attenuation on the end to prevent the digital level voltage from over driving the cassette players input.

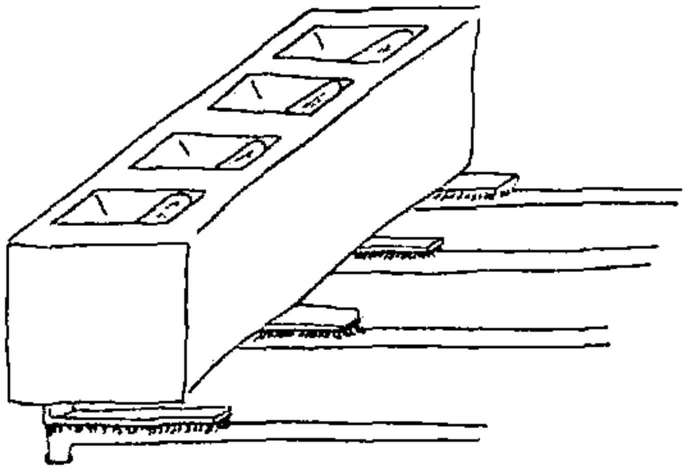
The input section is just a simple AC coupled common emitter transistor amplifier with the base heavily biased on. The bias on the transistor is important as this ensures that the software is able to read a steady logic 0 when no (AC) input is present.

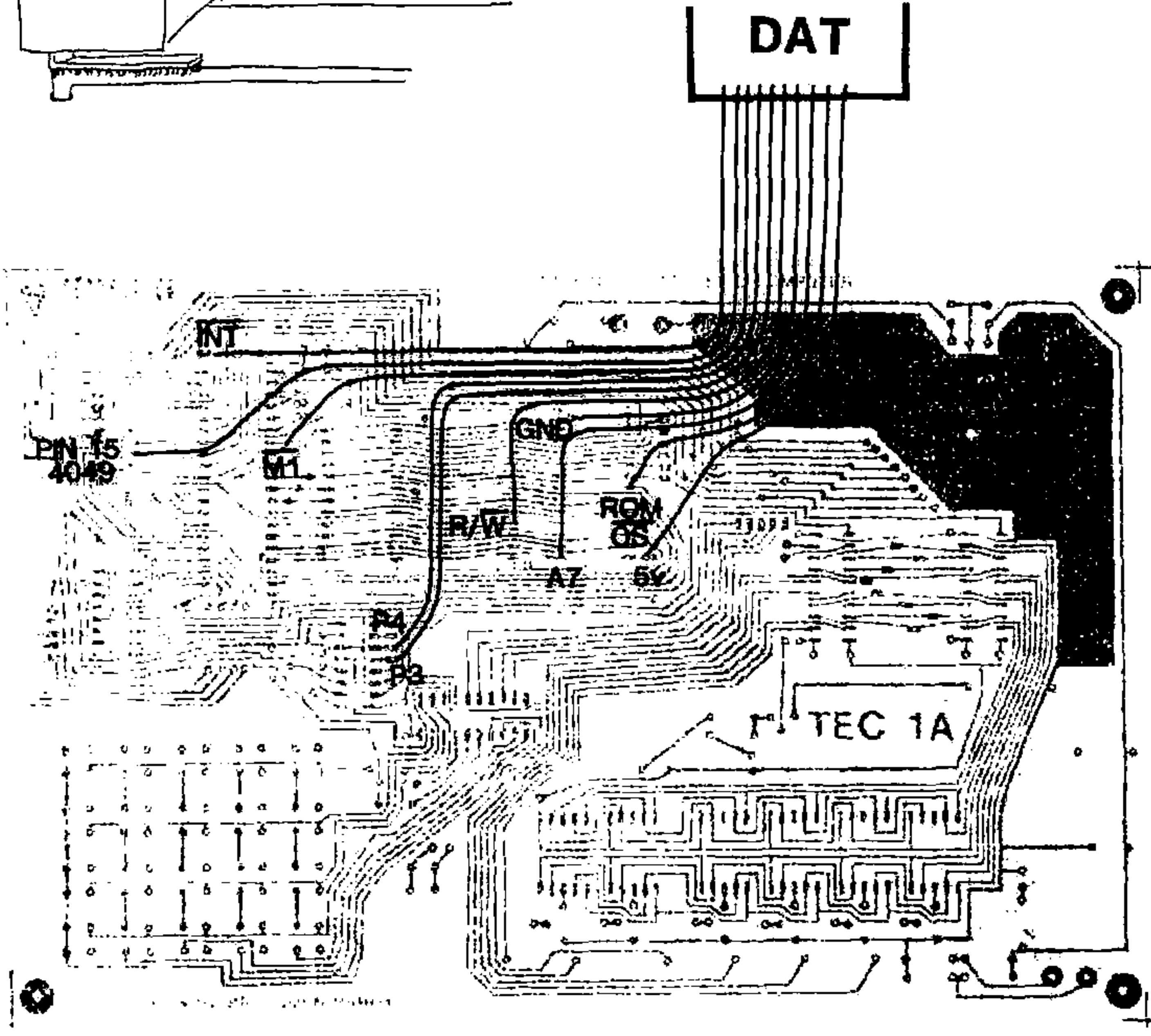
HOW THE SINGLE STEPPER INTERFACE WORKS

The single stepper INTERFACE works by interrupting the Z80 after each instruction. The interrupts are generated from a D flip flop on the DAT BOARD. Each time the Z80 fetches the first byte of an instruction a special signal called M1 is generated. This M1 is used to clock the ROM CS line into the D flip flop. The Q-bar output of the flip flop is connected to the INTerrupt pin. This means that an interrupt will be requested on every instruction fetch unless the instruction was fetched from the MONitor ROM.

It is important to prevent interrupts while executing in the MONitor ROM. If we don't, then an interrupt will occur just after it is re-enabled, at the end of the stepper routine. Immedialely following the EI (enable interrupt), is a RETurn. If an interrupt occurs on this RETurn, then the stepper routine is reinvoked and each time this RETurn is reached, the program loops back to the stepper routine forever!! (If it wasn't for this problem we would not require any external hardware at all).







TOP RIGHT
Bottom side of the DAT board with the feed-downs fitted.

TOP LEFT
Diagram showing how the cut-up IC socket is mounted on the links.

LEFT
Wiring diagram showing where the "flying leads" from the underside of the DAT board are connected to the TEC.
Note that the diagram DOES NOT show the wires leaving the DAT board in the correct order, only the correct places on the TEC board. Use the labels on the underside of the DAT board for the correct DAT wiring positions.

THE LIQUID CRYSTAL DISPLAY

by Jim.

INSIDE THE DISPLAY

The display has three internal registers though which all communication is done. These are the registers:

THE DATA REGISTER

The data register is a read or write register to which all DISPLAY DATA (in ASCII format) and BIT MAPPED PROGRAMMABLE CHARACTERS are sent. This DATA register acts as a TEMPORARY BUFFER between the internal DISPLAY RAM or CHARACTER GENERATOR RAM (both described below) and the host computer (our TEC).

Characters may also be read from this register.

Internal operations transfer data between this register and the internal RAM (or between RAM and this register). This register is located on port 84H.

THE INSTRUCTION (or CONTROL) REGISTER

The instruction register receives all instruction bytes. ALL bytes sent to this register will be interpreted as CONTROL by the LCD. This is a WRITE ONLY register and is decoded on port 04.

THE ADDRESS COUNTER/BUSY FLAG

Bit 7 of this register is used as the busy flag. After EVERY operation it goes HIGH to indicate that the display is not ready to perform ANY type of additional operation yet. As soon as the display becomes "on line" again, it will go LOW.

The lower 7 bits are the current address of the internal cursor. All read or write operations occur between the data register and the address held in this register.

This register is READ ONLY. (The ADDRESS COUNTER is set or altered by instructions sent to the INSTRUCTION REGISTER and then transferred into THE ADDRESS REGISTER by an INTERNAL operation). This register is located on port 04 with the control register. Internal decoding gates the R/W line to select between each register.

As well as the registers, the display contains both RAM and ROM. Below is a description of the internal memory inside the LCD.

THE DISPLAY RAM

All the display information sent to the DATA REGISTER is transferred into the DISPLAY RAM by an internal operation. This RAM can hold 80 bytes of display information. While the LCD may only display 32 characters at a time, the extra bytes allow for the display to be shifted or can serve as general purpose storage RAM. An unusual feature of the display RAM is that the address from the last location on the top line (27H) to the address on the bottom line (40H) IS NOT CONSECUTIVE.

THE CHARACTER GENERATOR ROM

This ROM contains 192 different 5x7 dot matrix characters. These include full upper and lower case Alphabet characters, numbers, maths symbols, Greek and Japanese characters.

All of the most used characters are here. Any type of character we need that is not there, can be made up on the CHARACTER GENERATOR RAM.

THE CHARACTER GENERATOR RAM

The CHARACTER GENERATOR RAM allows us to define up to 8 different characters of our choice. The format of each is a 5x8 dot matrix with the cursor making up the 8th row. Any or all can be displayed together on different parts of the LCD and also may appear in several places at once. We can use this to make games characters.

GETTING SOMETHING ON THE LCD

Using the LCD is easy because it contains its own "intelligent" chips which do all the hard work for us. From JMON, putting anything on the LCD is VERY easy because the LCD has been set-up by JMON.

JMON sets the LCD to shift the cursor right after each entry. You cannot see the cursor as it has been turned off by the software in JMON.

To aid with the experiments below, put FF at 0821 (the LCD will stop changing after the first F) and AA at 08FF. These disable the LCD from the MONitor (the FF at 0821) and stop the MONitor rebooting its variables on a reset (the AA at 08FF). The MONitor will reset to 0A00 to remind you that the variables have not be re- booted on reset. (Unless a key was held down while reset was pushed, in which case you must again put FF at 0821 and AA at 08FF).

Ok, lets start by putting the letter L on the screen.

Firstly we must clear the screen and send the cursor home. This may be done by one instruction - 01. We output this to the control register on port 04. Before we can output to the LCD we must wait until it is ready. Because this is required to be done frequently, the RST 30 instruction has been used to do this for us. The RST 30 reads the LCD busy flag and loops until it goes LOW.

Ok lets type this in:

```
0A00  F7      RST 30
0A01  3E 01   LD A,01
0A03  D3 04   OUT (04),A
0A05  76      HALT
```

Reset, Go

The display will go blank and the (invisible) cursor will return to home (top left-hand corner). The 01 instruction sets all the display RAM locations to 20H (space). The 01 instruction doesn't affect any previous mode setting or display options (discussed below).

Now enter this with the RST over the HALT at 0A05:

```
0A05  F7      RST 30
0A06  3E 4C   LD A,4C "(L)"
0A08  D3 84   OUT (84),A
0A0A  76      HALT
```

Reset, Go

The letter L appears in the top left corner.

Ok, now as before, put this in with the RST over the HALT:

```
0A0A  F7      RST 30
0A0B  3E 43   LD A,43 "(C)"
0A0D  D3 84   OUT (84),A
0A0F  F7      RST 30
0A10  3E 44   LD A,44 "(D)"
0A12  D3 84   OUT (84),A
0A14  76      HALT
```

Reset, Go

The *above* section outputs two more bytes to the DATA REGISTER.

Until now we have just been using a simple method to output data. This has shown us the basic way to talk to the LCD. Now that we have come this far and learned the basics, we'll advance to something more useful.

The code below will output a word onto the bottom line of the LCD. The display DATA will be held in a table at 0B00.

```
0A14  F7        RST 30
0A15  3E C0     LD A,C0
0A17  D3 04     OUT (04),A
0A19  01 84 06  LD BC,0684
0A1C  21 00 0B  LD HL,0B00
0A1F  F7        RST 30
0A20  ED A3     OUTI
0A22  20 FB     JRNZ 0A1F
0A24  76        HALT

0B00 4D 41 53 54 45 52
```

To set the cursor to the bottom line we output 80 to the instruction register (bit 7 sets the cursor address entry) + 40 (which is the actual address of bottom left display) = C0.

The OUTI instruction is new to our repertoire. It's operation is to output the byte addressed by HL to the port addressed by C. HL is then incremented and B is decremented. If B becomes ZERO the ZERO FLAG is set and the operation is complete. This instruction can output up to 256 bytes at a time.

Because we need to check the busy flag we loop back to the RST 30 until all the bytes have been done. If we didn't need to check the busy flag we could have used the OTIR instruction which automatically repeats itself until B=0.

All the above is done with the cursor switched off. For the next section we want to have the cursor on. To switch on the cursor output 0E to the instruction register on port 04.

```
0A00  F7     RST 30
0A01  3E 0E  LD A,0E
0A03  D3 04  OUT (04),A
0A05  76     HALT
0A06  C7     RST 00
```

Now let's see what does what on the display.

Using the above routine, output the bytes below one at a time, to port 04 and HALT between each. (leave what's on the display there).

Check the function of each on the table of controls.

```
18 1C 1C 1C 02 14
14 10 0C 0F 08 0C
```

Good luck!!

SETTING THE ENTRY MODE

The display may be configured to perform several different functions UPON EACH DATA BYTE ENTRY. They are:

1 INCREMENT CURSOR ADDRESS after storing inputted data byte (06H). This is our normal mode.

2 DECREMENT CURSOR ADDRESS after storing input (04).

3 SHIFT THE DISPLAY RIGHT after entry (05).

4 SHIFT THE DISPLAY LEFT after entry (07).

Each mode is selected by outputting the byte shown to port 04.

Once the entry mode is set it IS ONLY CHANGED BY ANOTHER ENTRY MODE SET COMMAND. None of the other control bytes will alter the entry mode.

The shift on entry feature (05,07) has been found to be difficult to use and even appears to contain design bugs.

You may experiment with it but we won't be using it in these notes.

The CURSOR DECREMENT may come in handy sometimes but it's more likely to be useful to processors which move blocks of data around in a more limited way to the Z80.

RUNNING WORDS ON THE LCD

Running words along the LCD is also simple because the LCD'S intelligent chips do most the work for us again. Our job is to enter the words we want to scroll (up to 16 characters per line for this routine) and send shift commands each time we want a shift.

The routine below is entered in 3 sections. Each section is a logical progression and increases the programs abilities. You can look at the instructions in each section and compare it to what the section does. This way you can learn how to put blocks together to use the display any way you want. Before entering the code below put FF at 0821 and AA at 08FF as described before.

Enter this and INCLUDE the NOPS and the table at 0B00 then run it:

```
0A00  3E 01     LD A,01
0A02  D3 04     OUT (04),A
0A04  F7        RST 30
0A05  3E 06     LD A,06
0A07  D3 04     OUT (04),A
0A09  F7        RST 30
0A0A  3E 0C     LD A,0C
0A0C  D3 04     OUT (04),A
0A0E  F7        RST 30
0A0F  00        NOP
0A10  00        NOP
0A11  00        NOP
0A12  00        NOP
0A13  00        NOP
0A14  01 84 10  LD BC,1084
0A17  21 00 0B  LD HL,0B00
0A1A  F7        RST 30
0A1C  ED A3     OUTI
0A1D  20 FB     JRNZ 0A1A
0A1F  F7        RST 30
0A20  3E C0     LD A,C0
0A22  D3 04     OUT (04),A
0A24  F7        RST 30
0A25  21 30 0B  LD HL,0B30
0A28  06 10     LD B,10
0A2A  F7        RST 30
0A2B  ED A3     OUTI
0A2D  20 FB     JRNZ 0A2A
0A2F  76        HALT

0B00: 54 41 4C 4B 49 4E 47 20
0B08: 20 20 20 20 20 20 20 20
      (TALKING)
0B30: 45 4C 45 43 54 52 4F 4E
0B38: 49 43 53 20 20 20 20 20
      (ELECTRONICS)
```

This will put "TALKING" on the top line and "ELECTRONICS" on the bottom line of the LCD and stop. Study the above section and see if you can work out the role of each instruction.

Now we'll add the shift section. Enter this with the first "NOP" over the last "HALT" and run it:

```
0A2F  00        NOP
0A30  00        NOP
0A31  3E 18     LD A,18
0A33  D3 04     OUT (04),A
0A35  01 00 80  LD BC,6000
0A38  0B        DEC BC
0A39  78        LD A,B
0A3A  B1        OR C
0A3B  20 FB     JRNZ 0A38
0A3D  18 F2     JR 0A31
```

The above code loads the shift instruction (18H) into the accumulator and outputs it to the control register on port 04.

As you can see it shifts the display, but this method is not very good if we want to shift only a few characters as we must wait for them to be shifted through the entire display RAM before they re-appear. To overcome this we can count the number of shifts and reset the display with a 02 command, as soon as all the letters have been shifted outside the display. The 02 instruction resets the display from shift WITHOUT CHANGING the contents of the DISPLAY RAM, CHARACTER GENERATOR RAM, or the CONTROL MODE. Because we would like the words to shift across the entire display and re-appear as soon as they have all gone, we must load the words just outside the screen to the right. The following additions make the words start shifting into the display from right-to-left.

Ok, Now enter the following, AT THE ADDRESSES SHOWN:

```
0A0F  3E 90  LD A,90
0A11  D3 04  OUT (04),A
0A13  F7     RST 30
-------------------
0A22  3E D0  LD A,D0
```

The above instructions set the DISPLAY RAM ADDRESSES to the RAM locations just right of the screen. The address of the top line is 90 and the address of the bottom line is D0. (Actually these are the addresses + 80H, the SET ADDRESS instruction).

```
0A2F  16 1B       LD D, 1B
```

(The D register is our shift counter).

```
0A3D  00      NOP
0A3E  00      NOP
0A3F  15      DEC D
0A40  20 EF   JRNZ 0A31
0A42  3E 02   LD A,02
0A44  D3 04   OUT (04),A
0A46  F7      RST 30
0A47  18 E6   JR 0A2F
```

The last group makes up the shift counter and reaets the display when the counter reaches Zero. When the 02 command is received by the LCD the display is returned to its NORMAL position. This means that the inputted data is returned to WHERE IT WAS ENTERED (just right of the screen). Now, when the next shift command is received, the lettera start to shift left back on to the screen.

QUESTION:
Why don't we need to wait for the BUSY flag to go low after the ahift instruction?

If you wish to change the number of characters to be shifted, you may do so by putting your new characters at 0B00 for the top line +and at 0B30 for the bottom line. Unused locations should have 20 (space) inserted until 16 locations are filled. (From 0B00 to 0B10 and from 0B30 to 0B40). The value of the loop counter loaded into D at 0A2F should also be changed. The valua of the loop counter is best set to 10H + the number of letters occurring in the longest line.
e.g. For the the example above:
ELECTRONICS = 11 (0BH) Letters.

So add 0BH + 10H = 1BH.
So 1BH is loaded into D at 0A2F.
To understand the above formula better, try 1C and 1A and see the result.

FINAL NOTES
The slow response of tha LCD detracts from the effectiveness of the shifting a little but by experimanting with the dalay at 0A35 you should be able to gat a good compromise betwean apeed and display clarity.

The abova ehifting method is just one of dozens of ways we could have used. A more complex program could shift information across and out ona end and load new information in the other to create a running information display.

Use the blocks in this program and the others to make up your own display routines. If you come up with something interesting, write in. We would love to aee what you've come up with.

DESIGNING YOUR OWN CHARACTERS

You can have up to aight different characters stored in a character-generator RAM. Each character Is displayed on the screen when it is addrassed in the display RAM. The addresses are between 0-7. The user-defined characters are made up of an 8x8 matrix (only 5 columns are displayed, the cursor makes up the 8th row.)

To set up a character, 8 bytas are outputted to the character- generator RAM. The first byte makes up the top row (only the 5 lower bits are displayed). The second byte makes up tha second row etc.

Before sending the 8x8 charactar (actually a 5x8 character), the entry moda must ba sat (if not alraady) to address-increment with no display shift (06) and a set character RAM address operstion must be done.

The control byte for this is 40 + the address of the first byte of each character-matrix. E.g: 40, 48, 50 for charactars 1, 2, 3 etc.

Once a character is set up, it is displayed by placing its address in the DISPLAY DATA RAM. Before doing this the DISPLAY RAM must be selected via 80 + address.

OK, let's put our own charactar on the LCD.

```
0A00  F7        RST 30
0A01  3E 01     LD A,01
0A03  D3 04     OUT (04),A
0A05  F7        RST 30
0A06  3E 40     LD A,40
0A08  D3 04     OUT (04),A
0A0A  01 84 08  LD BC,0884
0A0D  21 00 0B  LD HL,0B00
0A10  F7        RST 30
0A11  ED A3     OUTI
0A13  20 FB     JRNZ 0A10
0A15  F7        RST 30
0A16  3E 80     LD A,80
0A18  D3 04     OUT (04),A
0A1A  F7        RST 30
0A1B  3E 00     LD A,00
0A1D  D3 84     OUT (84),A
0A1F  76        HALT

0B00:

11, 0A, 04, 11, 0A, 04, 11, 0A
```

Experimant with the values in the table and see how it all goes together. By increasing the value loaded into B, to 10(hex) (at 0A0C) a second character may be programmed at the same tima. Tha table for the second character will start at 0B08. This will be displayed when a 01 is written into the DATA DISPLAY REGISTER. Experimant and see if you can get 8 characters appearing in several places at once on the display.

MYSTERY EFFECT

The routine below produces a very interesting effect. It uses the PROGRAMMABLE CHARACTER GENERATOR to produce 8 different characters soma ol which are displayed severat times. We won't tell you tha effact, we'll let you type it in and see for yourself. You won't ba disappointed!

Tha program consolidates much of what we have learned about "driving" the LCD. If you axperimant further and add a shift to it than it will be a complete ravision of what we have covered in thesa pages.

Now that you know how to use the LCD, start writing some programs that use it. If you come up with something interesting don't hesitate to send it in to TE. We would be very interested in some simple animation or an adventure game or anything that others would be interested in seeing. Go to it!

```
0A00  F7        RST 30
0A01  3E 01     LD A.01
0A03  D3 04     OUT (04),A
0A05  F7        RST 30
0A07  3E 06     LD A.06
0A08  D3 04     OUT 04
0A0A  21 00 0B  LD HL,0E00
0A0D  01 84 10  LD BC,1084
0A10  F7        RST 30
0A11  ED A3     OUTI
0A13  20 FB     JRNZ 0A10
0A15  F7        RST 30
0A16  3E 40     LD A.40
0A18  D3 04     OUT (04).A
0A1A  21 20 0B  LD HL,0B20
0A1D  06 40     LD B,40
0A1F  F7        RST 30
0A20  ED A3     OUTI
0A22  20 FB     JRNZ 0A1F
0A24  F7        RST 30
0A25  3E C0     LD A,C0
0A27  D3 04     OUT (04),A
0A29  21 10 0B  LD HL,0B10
0A2C  06 10     LD B,10
0A3E  F7        RST 30
0A3F  ED A3     OUTI
0A31  20 FB     JRNZ 0A3E
0A33  76        HALT

0B00: 20 4D 49 52 52 4F 52 20
0B08: 49 4D 41 47 45 21 20 20
0B10: 20 00 01 02 02 03 02 20
0B18: 01 00 04 05 06 07 20 20
0B20: 00 11 11 11 15 15 1B 11
0B28: 00 0E 04 04 04 04 04 0E
0B30: 00 11 12 14 1E 11 11 1E
0B38: 00 0E 11 11 11 11 11 0E
0B40: 00 11 11 1F 11 11 11 0E
0B48: 00 0F 11 11 17 10 11 0E
0B50: 00 1F 10 10 1E 10 10 1F
0B58: 00 04 00 00 04 04 04 04
```

CONCLUSION

This concludes this issues instalment on the LCD.

Study the previous notes carefully and get to know the LCD fully. There is enough information here for you to write routines using the LCD and we would like to see some ideas sent to us for issue 16.

The LCD will be supported further in issue 16 and if all goes well, we will have a cheap, full alpha-numeric keyboard with supporting software. I am working towards the stage were you can anotate your routines and send the text and the routine in on tape. We can then load them into our desk top publisher.

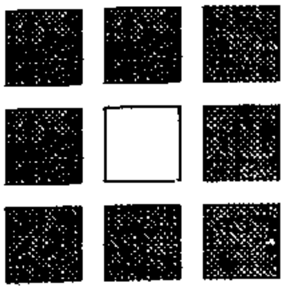
Don't forget if you have any good ideas or questions about the TEC, send them in to "TEC TALK." ●

Below is the table of LCD control bytes. Use these in conjunction with the previous notes

Instruction	Code										Function	Execu-tion time
	RS	R/$\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0		
(1) Display clear	0	0	0	0	0	0	0	0	0	1	Clears all display and returns cursor to home position (address 0)	1.64 ms
(2) Cursor Home	B	0	B	0	0	0	0	B	1	•	Returns cursor to home position. Shifted display returns to home position and DD RAM contents do not change.	1.64 ms
(3) Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets direction of cursor movement and whether display will be shifted when data is written or read	40 µs
(4) Display ON / OFF control	0	0	D	D	0	0	1	D	C	B	Turns ON/OFF total display (D) and cursor (C), and makes cursor position column start blinking (B)	40 µs
(5) Cursor/Display Shift	0	D	0	0	B	1	S/C	R/L	•	•	Moves cursor and shifts display without changing DD RAM contents	40 µs
(6) Function Set	B	0	0	B	1	DL	1	•	•	•	Sets interface data length (DL)	40 µs
(7) CG RAM Address Set	0	B	B	1	A_{CG}						Sets CG RAM address to start transmitting or receiving CG RAM data	40 µs
(8) DD RAM Address Set	0	0	1	A_{DD}							Sets DD RAM address to start transmitting or receiving DD RAM data	40 µs
(9) BF/Address Read	B	1	BF	AC							Reads BF indicating module in internal operation and AC contents (used for both CG RAM and DD RAM)	0 µs
(10) Data Write to CG RAM or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM	40 µs
(11) Data Read from CG RAM or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM	40 µs

• : Invalid bit	I/D = 1 : Increment	C = 1 : Cursor ON	R/L = 1 : Right shift
A_{CG} : CG RAM address	I/D = 0 : Decrement	C = 0 : Cursor OFF	R/L = 0 : Left shift
A_{DD} : DD RAM address			
	S = 1 : Display shift	B = 1 : Blink ON	DL = 1 : 8 bits
	S = 0 : No display shift	B = 0 : Blink OFF	DL = 0 : 4 bits
	D = 1 : Display ON	S/C = 1 : Display shift	BF = 1 : Internal operation in progress
	D = 0 : Display OFF	S/C = 0 : Cursor movement	BF = 0 : Instruction can be accepted

MAGIC SQUARE



by Jim Robertson

This is a fun game for the 8x8 that will have you amused and frustrated for hours.

The object is to light up the outside square of the 8x8. The game is made up of three 2x2 boxes of LEDs with a space between each. This makes full use of the 8x8 to display a playing field that is actually 3x3.

Nine keys are used to play the game and each key corresponds to a group of LEDs on the display.

TO SET UP

This game, like JMON, requires EITHER a 4k7 resistor between the NM1 (pin 17 of the Z-80) and D6 (Pin 10 of the Z-80) OR the LCD expansion board with the input chip fitted on port 3.

The 8x8 is fitted to ports 5 and 6 with the port select strobe of the left-hand latch going to port 6.

This is very important! (once you master the game, try swapping them over, this will invert the playing field and gives you a mirror image to work with).

The 8x8 is placed with the LEDs above the latch chips.

It is important to fit the 8x8 before typing in the code or at least hold down the reset if you have already entered the code, by using your third hand.

MAGIC SQUARE has been written to run with the TEC crystal oscillator however it will work with the 4049 oscillator but the tones will be lower pitched.

TO PLAY

Type in the code and save it if you have a tape system. Now address 0C00 and press GO. The code is placed at 0C00 to allow Simon and Magic Square to be saved, loaded and played together (however they do not require each other). (Unfortunately Simon has been held over to issue 16 because of the shortage of space in this issue).

After starting the game, a random pattern appears. By pressing the game keys, the playing field will change. Each key has a particular effect that remains constant throughout the game. The effects of each key is for you to work out!

The keys used for the game are: 4, 5, 6, 8, 9, A, C, D and E.

As you can see, these make up a 3x3 box pattern on the keyboard.

Go to it! The object of the game is to light up the outside border with the centre OFF.

A fair point to add is that it is always possible to do this regardless of the starting pattern - believe it or not!

When (if!) you finally succeed, your effort will be greeted enthusiastically on the 8x8. The game may be re-started by hitting the GO key.

HOW THE SOFTWARE WORKS

Three random numbers are generated from the time it takes to release the GO key and also from the refresh register.

The three lowest bits of these three bytes are used to form a 3x3 matrix. The top 5 bits are ignored.

All processing, pattern changing and testing is done on this 3x3 matrix. After processing, this matrix is converted to its equivalent 8x8 display and then scanned. A loop is used to scan the 8x8 and read the keyboard until a key is detected.

When any key is detected for the first time, a flag byte remembers this and the program will ignore any subsequent pushes.

This allows each key to be processed just once. When no key is pressed, the flag is cleared to allow the next key to be processed.

When a key is pressed and allowed as a "FIRST KEY" press, it is checked for a corresponding table entry. If no corresponding value is found, the key is ignored. This is how the unwanted keys are masked.

After a key has been validated a table entry 9 bytes higher is accessed. This entry is a byte that will be exclusive-ORed with the first byte of the 3x3 matrix. A second byte 9 bytes higher again contains the low order byte of the address of the 3x3 matrix entry. The first byte is now EX-ORed with the matrix byte and the result stored as the new updated matrix byte. This is how the patterns are changed.

The above process is repeated for the second and third matrix bytes. The exact same process described above is used. The entry for the second byte is 9 bytes higher than the first and the address 9 higher again.

The same convention is used for the third entry. This convention allows a loop to be used for all three matrix bytes. This loop is located at 0C49.

After the above process, the 3x3 is checked for the required box pattern. If correct, the pattern is converted to its 8x8 format and flashed with accompanying tones.

If the pattern is not complete, the program loops back to the main playing loop.

A routine at 0CAB converts the 3x3 to 8x8 display format. This routine is called after all the required processing has been performed on the 3x3 matrix. This routine is a loop that gets each 3x3 matrix byte, calls another routine to convert each matrix bit to two 8x8 bits and spacing, then stores the result twice and adds a blank line.

The last blank line is ignored by the scan routine and the result is an 8x8 format.

At 0CC4 a loop converts one bit to two and adds spacing. This is done by shifting the matrix bit into the carry and if the carry is clear, the two 8x8 bits are left clear and shifted twice for the 2x2 box bits and once for the space between.

If the carry is set, the 2x2 box bits are set by rotating the SET CARRY into the 8x8 byte and also setting bit 7 before rotating. This will then set the carry after the first rotation, ready for the second rotation. The third rotation clears the space bit. After this is done three times, the 8x8 byte is rotated back to remove the last unwanted space before returning.

THE TONE ROUTINE

The tone routine is located at 0CD8. The duration of the tone period is in D while the cycle count is in E. The "KEY PRESS" beep uses this value loaded into DE while other tones such as the restart tone load DE before calling the tone routine.

SCAN ROUTINE

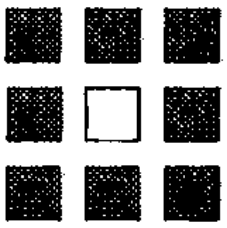
The scan 0CE7 is a straight-forward multiplex routine except that it scans backwards. This allows the 8x8 to be right-way-around while keeping the rest of the program straight forward (otherwise the 8x8 buffer would need to be loaded backwards).

"Magic Square" contains a number of very valuable "building blocks" that can be used in your own programs. It can stand studying for many hours to see how the various operations have been achieved. The fully documented program is presented on the next two pages and you should add your own notes alongside Jim's to help you understand what is happening at each step.

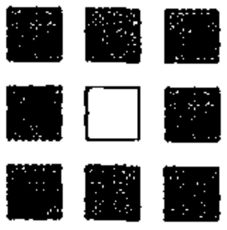
Colin Mitchell.

MAGIC SQUARE PROGRAM

```
         0C00  11 00 00  LD DE,0000     Random number generated
         0C03  13        INC DE         by the duration it takes the player to release the key at the start of the
         0C04  DB 03     IN A,(03)      program.
         0C06  CB 77     BIT 6,A
         0C08  28 F9     JR Z,0C03
         0C0A  ED 5F     LD A,R         The value of the refresh register is loaded into the accumulator.
         0C0C  82        ADD A,D        D register is added to the accumulator and stored as the first value.
         0C0D  32 40 0D  LD (0D40),A    E register is added (with carry) and stored as the second value.
         0C10  8B        ADC A,E        Registers are added to the accumulator and shifted to produce the
         0C11  32 41 0D  LD (0D41),A    third random number. This is also stored.
         0C14  82        ADD A,D
         0C15  83        ADD A,E
         0C16  07        RLCA
         0C17  32 42 0D  LD (0D42),A
MAIN     0C1A  CD AB 0C  CALL 0CAB      Call 3x3 to 8x8 conversion routine.
PLAYING  0C1D  CD E7 0C  CALL 0CE7      Call scan.
LOOP     0C20  DB 03     IN A,(03)      Test for key press.
         0C22  CB 77     BIT 6,A        If bit 6 on port 7 HIGH then no key is pressed.
KEY      0C24  28 06     JR Z,0C2C      Jump if key pressed otherwise clear "key pressed" flag and loop until
PRESSED  0C26  AF        XOR A          key pressed. Otherwise clear.
         0C27  32 43 0D  LD (0D43),A    "key pressed" flag.
         0C2A  18 F1     JR 0C1D        Loop until key pressed.
         0C2C  3A 43 0D  LD A,(0D43)    Test "first key press" flag.
         0C2F  B7        OR A
         0C30  20 EB     JR NZ,0C1D     Jump if key already pressed, otherwise set key pressed flag
         0C32  3E FF     LD A,FF
         0C34  32 43 0D  LD (0D43),A
         0C37  21 00 0D  LD HL,0D00     HL = base of valid key table.
         0C3A  01 09 00  LD BC,0009     BC = number of valid key entries
         0C3D  DB 00     IN A,(00)      Get input value from encoder chip
         0C3F  E6 1F     AND 1F         mask unwanted bits
         0C41  ED B1     CPIR           block compare with increment.
         0C43  20 D8     JR NZ,0C1D     NZ means no right entry. After all values tested, ignore key.
KEY      0C45  CD D8 0C  CALL 0CD8      Key valid. Call key pressed beep.
VALID    0C48  2B        DEC HL         Decrement HL as CPIR increments it before testing the zero flag.
         0C49  11 09 00  LD DE,0009     DE = table index.
         0C4C  06 03     LD B,(03)      Set B for 3 loops. One for each matrix byte.
         0C4E  19        ADD HL,DE      Get value to EX-OR with matrix.
         0C4F  7E        LD A,(HL)      Save in A.
         0C50  19        ADD HL,DE      Calculate address of low byte of matrix byte and put in HL.
         0C51  E5        PUSH HL        Save for later.
         0C52  6E        LD L,(HL)      Set HL to matrix byte address.
         0C53  AE        XOR (HL)       Toggle bits and store
         0C54  77        LD (HL),A      as updated matrix byte
         0C55  E1        POP HL         Recover HL
         0C56  10 F6     DJNZ,0C4E      Loop for 3 bytes.
         0C58  21 40 0D  LD HL,0D40     Check for box pattern. (HL) = first matrix byte.
         0C5B  7E        LD A,(HL)
         0C5C  E6 07     AND 07         Remove unwanted bits
         0C5E  FE 07     CP 07          and test for 7 (111)
         0C60  20 B8     JR NZ,0C1A     Jump to main playing loop if not 7, otherwise
         0C62  23        INC HL         Test second matrix byte.
         0C63  7E        LD A,(HL)
         0C64  E6 07     AND 07
         0C66  FE 05     CP 05          Test for 5, (101)
         0C68  20 B0     JR NZ,0C1A     Jump if not, otherwise
         0C6A  23        INC HL         do third matrix
         0C6B  7E        LD A,(HL)      byte which should
         0C6C  E6 07     AND 07         be equal
         0C6E  FE 07     CP 07          to 7 (111)
         0C70  20 A8     JR NZ,0C1A     Jump if not box pattern.
PATTERN  0C72  CD AB 0C  CALL 0CAB      Pattern right so call 3x3 to
DONE!    0C75  11 30 00  LD DE,0030     8x8. Load DE with win tone
         0C78  CD DB 0C  CALL 0CDB      and call tone routine.
         0C7B  06 03     LD B,03        Set B for 3  flashes.
         0C7D  C5        PUSH BC        and save count
         0C7E  18 10     LD D,10        D = scan counter
         0C80  CD E7 0C  CALL 0CE7      Call scan.
         0C83  15        DEC D          Loop until D = 0
         0C84  20 FA     JR NZ,0C80
         0C86  AF        XOR A          Clear display.
         0C87  D3 06     OUT (06),A
         0C89  CD D8 0C  CALL 0CD8      Call beep.
         0C8C  01 00 15  LD BC,1500     Load BC with off time
         0C8F  0B        DEC BC         and delay.
```



```
          0C90   78         LD A,B
          0C91   B1         OR C
          0C92   20 FB      JR NZ,0C8F
          0C94   C1         POP BC        Recover flash loop counter
          0C95   10 E6      DJNZ 0C7D     and loop for 3 flashes.
          0C97   CD E7 0C   CALL 0CE7     Call scan.
          0C9A   DB 00      IN A,(00)     and loop continuously
          0C9C   E6 1F      AND 1F        looking for the GO key
          0C9E   FE 12      CP 12         to be pressed.
          0CA0   20 F5      JR NZ,0C97    Jump if GO not pushed.
          0CA2   11 80 00   LD DE,0080    Load DE with restart tone
          0CA5   CD DB 0C   CALL 0CD8     Call tone.
          0CA8   C3 00 0C   JP 0C00       Restart game.
3x3       0CAB   06 03      LD B,03       B = loop counter set for 3 conversions.
to        0CAD   21 40 0D   LD HL,0D40    HL = address of 3x3 matrix.
8x8       0CB0   11 50 0D   LD DE,0D50    DE = 8x8 buffer.
MATRIX    0CB3   C5         PUSH BC       Save loop counter.
TO        0CB4   7E         LD A,(HL)     Get matrix byte.
DISPLAY   0CB5   CD C4 0C   CALL 0CC4     Call 1 to 3 bit conversion.
FORMAT    0CB8   12         LD (DE),A     Save first display
          0CB9   13         INC DE
          0CBA   12         LD (DE),A     byte twice
          0CBB   13         INC DE        and then
          0CBC   AF         XOR A         add
          0CBD   12         LD (DE),A     a blank line
          0CBE   13         INC DE        increment to next display buffer.
          0CBF   23         INC HL        Increment HL to next matrix byte.
          0CC0   C1         POP BC        Recover loop counter.
          0CC1   10 F0      DJNZ 0CB3     Repeat for 3 bytes.
          0CC3   C9         RET           done.
1 TO 3 BIT 0CC4  01 00 03   LD BC,0300    B = 3 loops. C is cleared ready to receiver display byte.
CONVER-   0CC7   0F         RRCA          Rotate matrix byte to set or clear carry.
SION      0CC8   30 02      JR NC,0CCC    Jump NC to shift C 3 places
          0CCA   CB F9      SET 7,C       else set bits 1 and 2 of C with SET CARRY and
          0CCC   CB 11      RL C          bit 7
          0CCE   CB 11      RL C          rotate C left
          0CD0   CB 11      RL C          Last rotation inserts space
          0CD2   10 F3      DJNZ 0CC7     do for 3 loops
          0CD4   CB 19      RR C          remove last space
          0CD6   79         LD A,C        place result in A.
          0CD7   C9         RET           done.
BEEP      0CD8   11 50 50   LD DE,5050    D= perfod  E = loop countar
          0CDB   AF         XOR A         Clear A.
TONE      0CDC   D3 01      OUT (01),A    Sound out to speaker.
          0CDE   42         LD B,D        Delay for tone
          0CDF   10 FE      DJNZ 0CDF     period.
          0CE1   EE 80      XOR 80        Toggle bit 7,A (speaker bit)
          0CE3   1D         DEC E         Decrement loop counter.
          0CE4   20 F6      JR NZ,0CDC    Loop until zero.
          0CE6   C9         RET           Done.
SCAN      0CE7   21 57 0D   LD HL,0D57    HL = end of 8x8 buffer.
          0CEA   06 80      LD B,80       B = scan bit output byte.
          0CEC   7E         LD A,(HL)     Output first display
          0CED   D3 05      OUT (05),A    byte to port 5
          0CEF   78         LD A,B        then output scan bit
          0CF0   D3 06      OUT (06),A    to port 6.
          0CF2   06 40      LD B,40       short multiplex
          0CF4   10 FE      DJNZ 0CF4     display delay
          0CF6   2B         DEC HL        Decrement HL to next display byte
          0CF7   47         LD B,A        replace scan bit in B.
          0CF8   AF         XOR A         clear accumulator and
          0CF9   D3 06      OUT (06),A    output to port 6.
          0CFB   CB 08      RRC B         Shift scan bit loop until scan bit
          0CFD   30 ED      JR NC,0CEC    falls into carry
          0CFF   C9         RET           then return.
TABLES:

0D00: 04 05 06 08 09 0A 0C 0D 0E 06 04 00 07 02 00 03

0D10: 01 00 40 40 40 40 40 40 40 40 40 40 06 04 02 07

0D20: 02 01 03 01 41 41 41 41 41 41 41 41 41 00 04 06

0D30: 00 02 07 00 01 03 42 42 42 42 42 42 42 42 42
```



DRIGS

How many times have you flipped through a magazine in the vain hope of finding an interesting article? As you thumb through the pages, the hopes are dashed and finally you come to the last page.

There you find an article of untold wit and interest to make the purchase of the magazine worthwhile.

I hope this isn't the present case but now that you have reached the end I want to summarise all that's happened in the TE world and bring together the projects in this issue.

Firstly we have the CAR ALARM. It has a range of features to rival anything on the market and can be built for under $100. If you have a car worth saving, this alarm will keep your mind at rest.

The TEC article is quite large in this issue as a result of Jim's tireless effort. Some readers, not interested in the TEC, will think it goes on for too long. But one of the biggest criticisms of worthwhile projects in other magazines is the lack of back-up and support.

Generally the project extends over an issue or so and is never heard of again. All those who build it up are left high and dry with little understanding of its full potential.

Not so with the TEC. We have gone e full circle looking for other microprocessors, to rival the Z-80. But after spending thousands of dollars and hundreds of hours we have come to the conclusion that the Z-80 is the best (overall) and cheapest on the market.

With this wa have no hesitation in bringing you pages of assistance in designing and developing programs in Z-80 code and it is our firm belief the the Z-80 will be around for years to come.

The two new projects for the TEC are the DAT board and Speech board.

The DAT board is a boon for programmers as it allows programs to be written and analysed one step at a time via a single stepper program.

The DAT board interfaces the TEC to a tape recorder to allow the storage of programs in a very convenient form.

The software to drive the DAT board and the tape interface is contained in Jim's new MONitor ROM called JMON.

JMON is the result of 9 months continuous refinement and hundreds of different versions have been created over that time. The end result is certainly a very good MONitor package.

Because JMON is a considerable advancement over MON 2, if you are building the computer from scratch you should start with MON 1B/2 and go through the experiments contained in issues 10, 11, 12, 13, and 14.

One essential add-on for the TEC is the NON VOLATILE RAM (issue 13) and if you would like to create a matrix of 64 pixels, the 8x8 is a must.

After these you can build the DAT BOARD and appreciate its wide range of capabilities.

In the TEC article we have supplied a game program called MAGIC SQUARE.

It can be typed in and played on the 8x8 display. MAGIC SQUARE is fully documented and it is hoped that you will appreciate how the routine works as much as you enjoy tha game.

MAGIC SQUARE will have you baffled for hours. Once you work out how to get the square out, see it you can work out how the program works!

Once you have typed it in, it can then be saved on tape and recalled later.

If the TEC computer has grabbed you, a documentary package is evailable from Jim for $15.00 plus $2.50 postage. In this you get a full line-by-line explenation of how all the JMON routines work. As Jim put it, he hopes that you can understand the purpose of every instruction. Also if you purchase this package, you help Jim offset his costs on developing JMON and the DAT BOARD etc.

Drigs is the dregs!

There ere also some other notes on programming in the package and will prove to be more beneficial than buying a $25 book on the Z-80 by a "bandwagon" publishing Co.

If you have really been bitten by the bug, you can buy a program tape with two TEC games, written by Cameron Sheppard. The first program is called MAZE and is played on the 8x8. It consists of a 27x30 playing field and the aim is to get out. This will keep you occupied for weeks!

The other game is "TEC invaders" and it's a bit like Space Invaders on a smaller scale.

These programs came as a result of co-operation between Jim and Cameron. It all started when Cameron came into TE some years ago with his TEC INVADERS. At the time the program was far to long to be published and the best efforts of Cameron and myself were unable to shorten it. One day Jim found the program and thought it would be a good program to put onto tape. Jim rang Cameron and they formed an agreement togather.

Jim provided Cameron with a tape system and Cameron did some work required to add the finishing touch.

The rest is history!

To go over the TEC projects once again: DAT BOARD and PCB is $55.35 or $16.35 if you want just the tape and single stepper facilities without the LCD. Speech is $27.25, JMON is $16.00. Jim's package is $15.00. Cameron's tape is $6.50. Jim's EPROM programmer up-date is $2.30. These are all essential if you want to get into programming.

Next we have a beginners project (although the soldering requires a fair degree of skill). It's an Organ along the lines of a stylus organ and is great fun to pley. It looks most impressive when built up and is ideal as e gift for the budding Beethoven.

A miniature FM radio has been a constent request from readers who have constructed one of our FM bugs. It's small enough to be hidden and allows you to create your own FM link.

And finally we have the FM bug that everyone's been waiting for. Our 1km bug, the ULTIMA. It's a sneak preview of out next bugging book "Security Devices." Once the word got out thet it had been developed, we started selling kits! Now it's available for everyone and provided you are careful, you can experiment and achieve ranges of 1km and more.

The articles for Security Devices are nearly ready for final page-making and they should be going to the printer very soon.

Apart from the 1km bug and FM radio, we have included 6 other security-related projects to add to our range.

Many of these are not available on the market while others cost hundreds, if not thousands of dollars. (Take for instance the Pen bug. It sells for over $3,000 on the commercial market!)

You can save a fortune by building things yourself and at the same time, learn how its all done.

Look out for this book, as well as future issues of the magazine at your local newsagents or send for a subscription and be assured you don't miss a copy.

ISSUE 16

We hope to see both issue 16 and 17 out in '89. Issue 16 should not be too far off as we have have numerous articles left over from this one.

Jim is designing an expansion board for the TEC. The board was to be presented in this issue but it became clear that there just wasn't the room. Jim's board increases the memory by 20k. 8k of this is a battery backed RAM. There is an on-board "intelligent" EPROM programmer thet when not being used to read and program EPROMS, can be used as 20 general input/output lines.

So we come to the end of another packed issue. So full that we didn't have any room for the adverts. Ah! Such is life. A magazine without advertising.