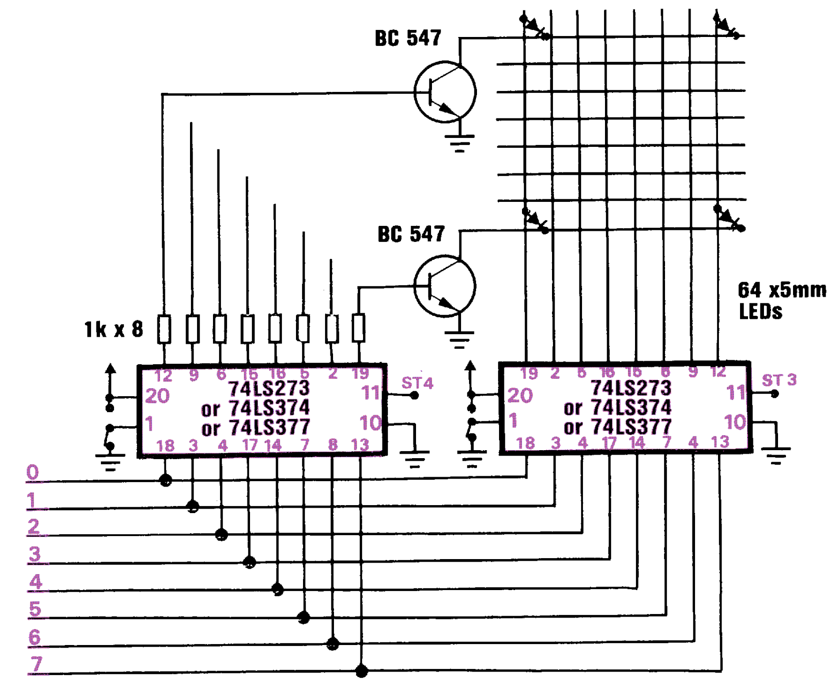
8x8 MATRIX



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.

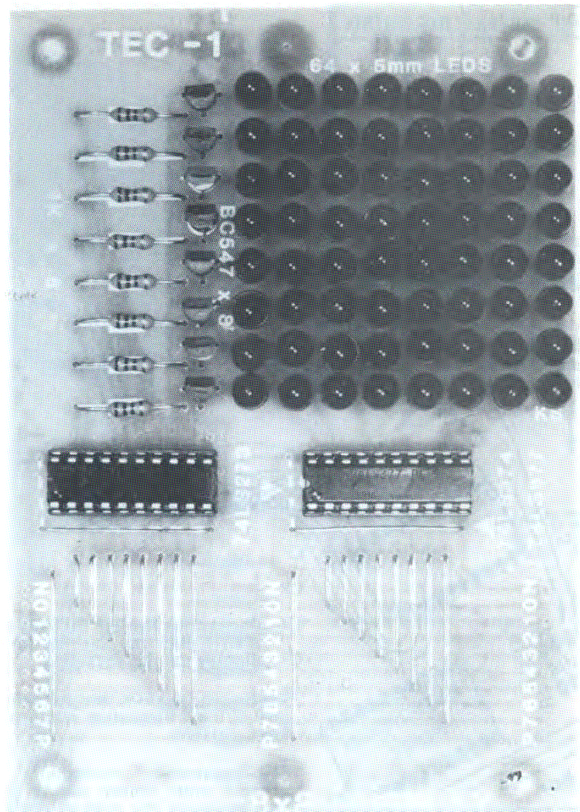
CONSTRUCTION

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.

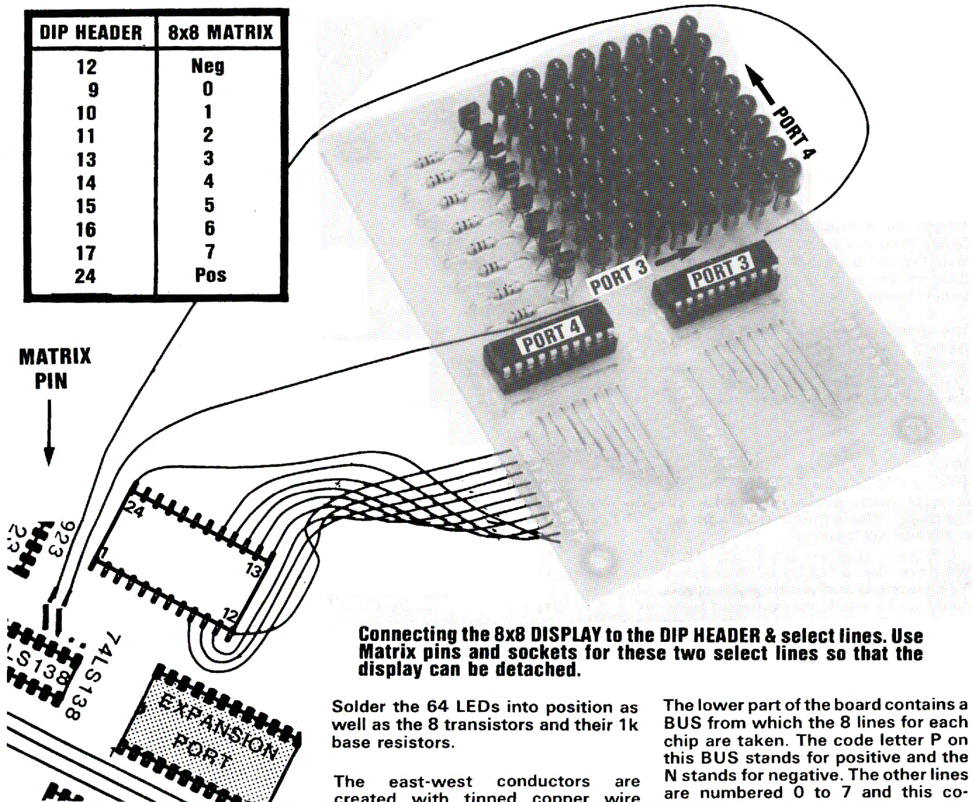
Next you must be certain which way they are to be inserted in the PC board. A mistake will take a very long time to rectify. The cathode lead is nearest to the row of transistors. When soldering the LEDs to the board, you must take special care to keep them all the same height and perpendicular to the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

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.

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.

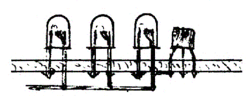


Diagram showing how the 'COMMON' line is created on the underside of the board. Both leads of each LED are soldered to the PC board. But only the ANODE lead is cut short. The CATHODE leads are either joined with a length of tinned copper wire which runs below the board, or each lead is bent over and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

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
```

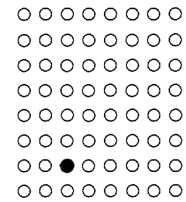
Replace any dull LEDs.

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
```

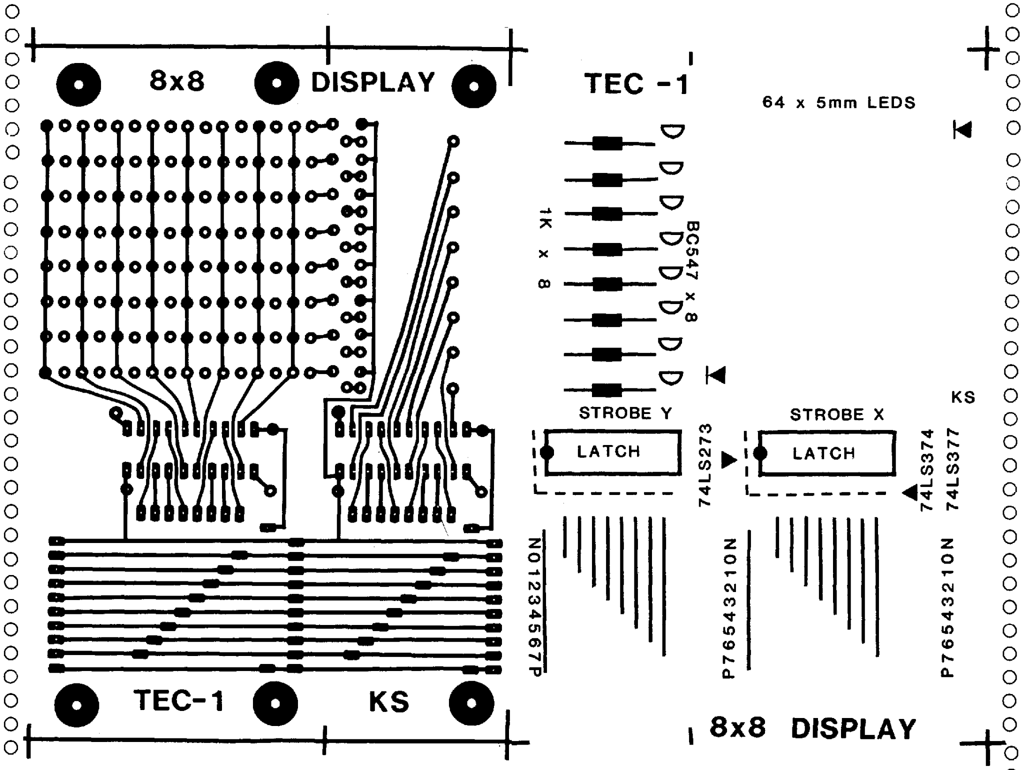
To check the data and chip select lines, insert this program:

```
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.

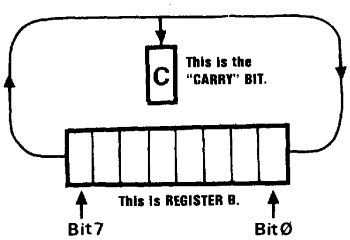
4. To run the number 7 across the screen, change location 801 to **29.**

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.

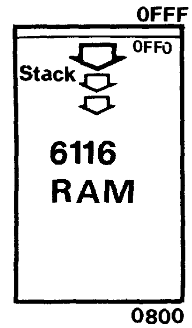
This area is located in the 6116 RAM at address 0FF0. (This is only 16 bytes from the end of this chip's memory and is usually considered to be the unused end of the RAM.)

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.

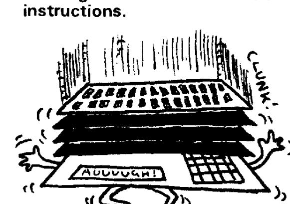
We PUSH new bytes onto the stack and POP bytes off the top.

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
```

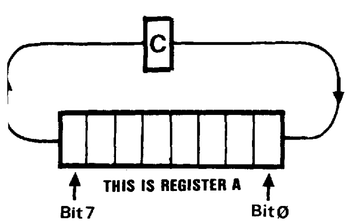
Using the program above, change the address location 808 to 17. This is the machine code instruction for rotating the accumulator left. (RLA), through the carry.

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 produced 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.

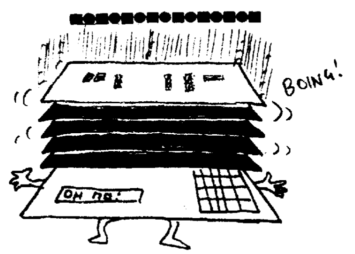
It should be noted that high memory is used by the Z80 to store its stack and thus memory above 0F00 should not be used.

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

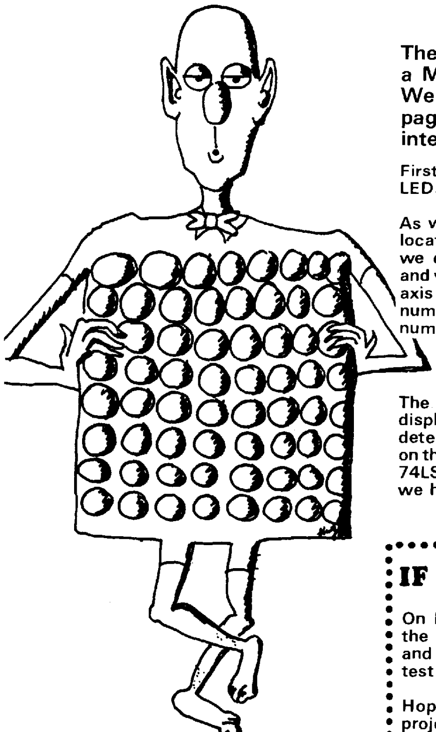
```
05B0   21   LD HL,0800
05B3   11   LD DE,0B00
05B6   7E   LD A,(HL)
05B7   FE   CP FF
05B9   C2   JPNZ,05C2
05BC   21   LD HL,0800
05BF   C3   JP 05B6
05C2   D3   OUT (03),A
05C4   1A   LD A,(DE)
05C5   FE   CP FF
05C7   C2   JPNZ,05D0
05CA   11   LD DE,0B00
05CD   C3   JP 05C4
05D0   D3   OUT (04),A
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.

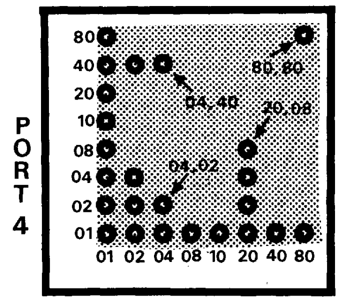
The only other fault we have seen is one row glowing brighter than the rest. This can be due to one of the transistors shorting between collector and emitter. A short to base may cause the row to be extinguished.

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.



PORT 3

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
```

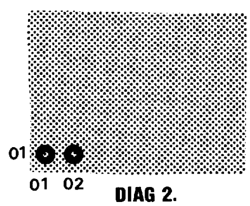
As you type the program, this is what you should be saying: Load the accumulator with 4, output it to port 3. Load the accumulator with 2 and output it to port 4. Halt.

Problems:

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

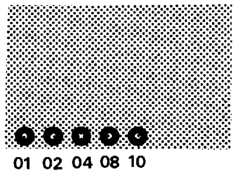
TWO OR MORE LEDs

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?

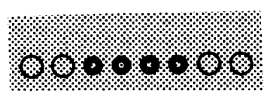
Diag 4.

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:

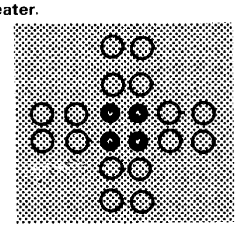


Diag 5.

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.

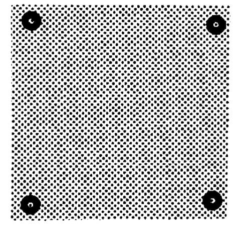


Diag 6.

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?

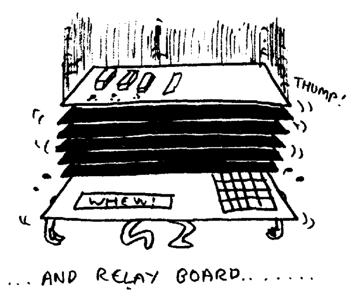


Diag 7.

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 Byte.
D3 03
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:

TOP: **80**, **40**, **20**, **10**
BOTTOM: **8**, **4**, **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

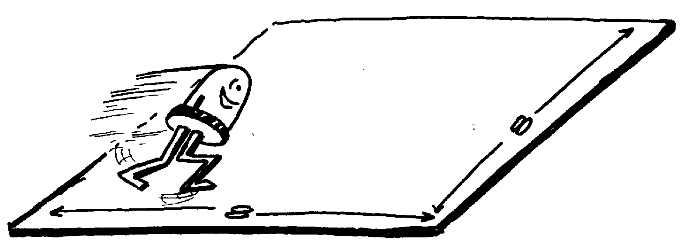
You can experiment with the length of the delay to produce a faster or slower flash rate.

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**	**0E 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**	**0D**
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.)

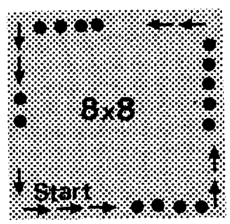
Delay routine at OCOO:

```
11 FF 06
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:

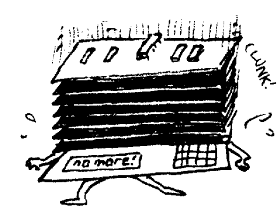
change **802** to **26 FF**
change **818** to **25**

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.

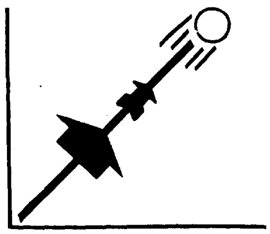
```
LD H,01        800   26 01
LD A,H         802   7C
OUT (3),A      803   D3 03
LD C,08        805   OE 08
LD A,01        807   3E 01
OUT (4),A      809   D3 04
LD B,A         80B   47
CALL DELAY     80C   CD 00 OC
LD A,B         80F   78
RLC A          810   CB 07
DEC C          812   OD
JP NZ LOOP     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   17
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.

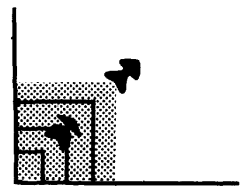
At location **801** insert the value 90. Try both directions of travel and watch the different effects.

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:

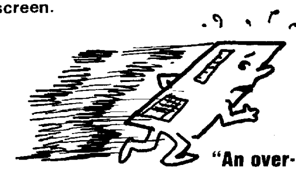
```
11 FF 06
1B
7A
B2
C2 03 09
C9
```

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.

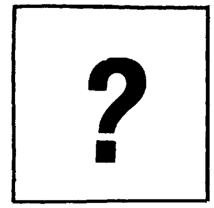
The final result is most impressive. The display fans out from the lower left-hand corner to fill the entire screen.



"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 0C 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.

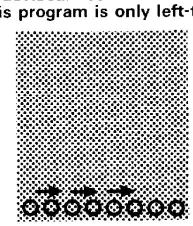
This is the basis of the next set of programs. The correct key must be pressed for the program to be executed. Otherwise the program will return to the HALT instruction and the outputs will not change.

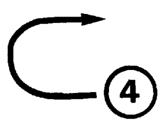
MOVING A LED VIA KEY '4'.

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.

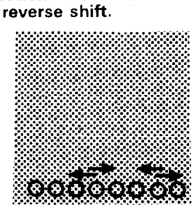
The program is HALTED at 808 and the Z80 waits for a keyboard instruction. When a key is pressed, the NMI line is activated and the data is sent to the Z80 and initializes the INTERRUPT VECTOR REGISTER 'I'. The keyboard data is placed in the accumulator register and compared with the value 04. If the 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.

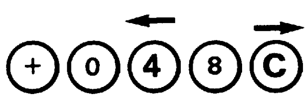
The new data-value in register B is loaded into register A at 805 and is passed to the display latch port 3 at 806 and 807.

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 0C	**815**	**FE 0C**
JP NZ 808	**817**	**C2 08 08**
RRC B	**81A**	**CB 08**
JP 805	**81C**	**C3 05 08**

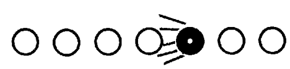
This, and many other features can be altered to suit your own requirements. It is a matter of experimenting and determining which instruction should be altered. If you discover these changes yourself, you will have a much greater understanding of how the program is put together.

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.

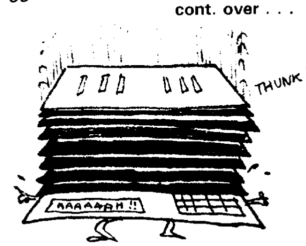
ADDING AUTO REPEAT

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.

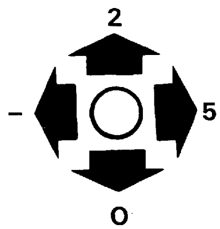
```
LD A,01        800   3E 01
OUT (4),A      802   D3 04
LD A,01        804   3E 01
OUT (3),A      806   D3 03
LD B,01        808   06 01
HALT           80A   76
LD A,I         80B   ED 57
CP 04          80D   FE 04
JP NZ 81A      80F   C2 1A 08
RLC B          812   CB 00
LD A,B         814   78
OUT (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

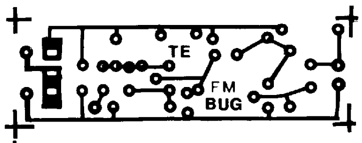
```
LD HL,0903     800   21 03 09
LD A,01        803   3E 01
LD HL,A        805   77
LD C,30        806   06 30
CALL 0903      808   CD 0E 09
INC (HL)       80B   34
DJNZ CALL      80C   10 FA
JP 0800        80E   C3 00 08

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
```

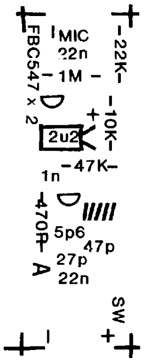
I have run out of room for this issue and still have lots more programs and ideas. Next issue will contain another 20 pages of programming and include 2 more programs from Mr Allison.

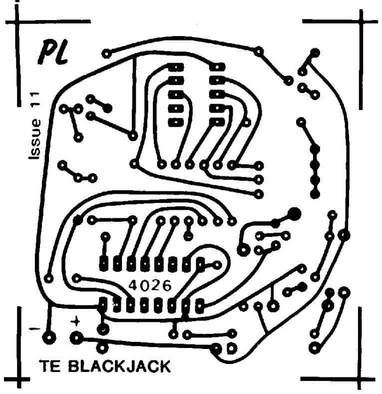
Turn to P.50 for 6 pages on the RELAY DRIVER BOARD project and type in the program for operating the relays.

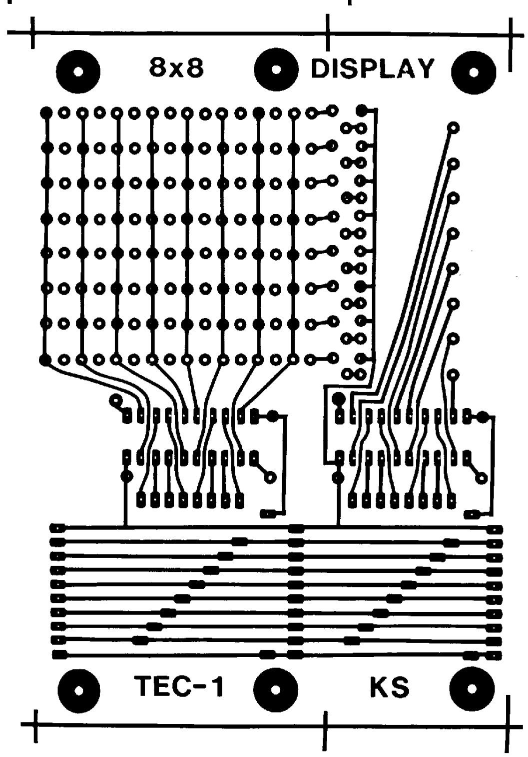
The projects for next issue . . . . .I'll keep them a secret, but you'll be very pleased; I assure you.

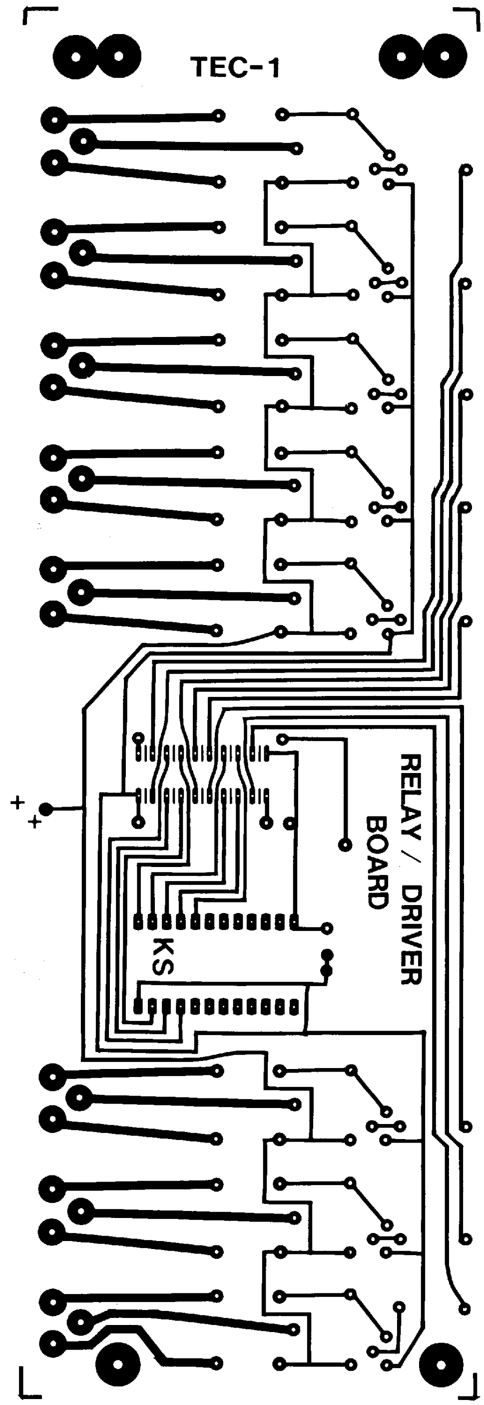
PC ARTWORK


TE
FM
BUG


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


PL
Issue 11
4026
TE BLACKJACK


8x8
DISPLAY
TEC-1
KS


TEC-1
RELAY / DRIVER
BOARD
KS

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:

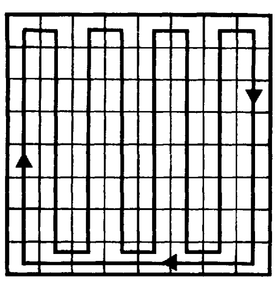
```
11 FF 06
1B 7B B2
C2 03 09
C9
```

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
```

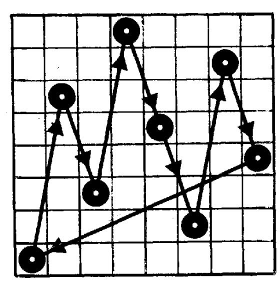
Type the first section into the TEC and **RUN**. This will check the code-values and prevent a major mistake. Type the second stage and **RUN**. Continue this way until the whole program has been inserted.

```
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
```

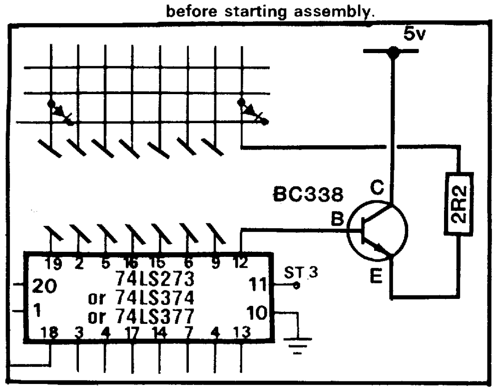
Delay at 0A00:

```
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
```



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 modification:

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.

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.

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.

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.

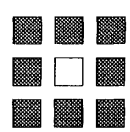
SCAN ROUTINE

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).

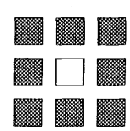
"Magic Square" contains a number of very valuable "building blocks" that can be used in your own programs. It can stand studying for many hours to see how the various operations have been achieved. The fully documented program is presented on the next two pages and you should add your own notes alongside Jim's to help you understand what is happening at each step.
Colin Mitchell.

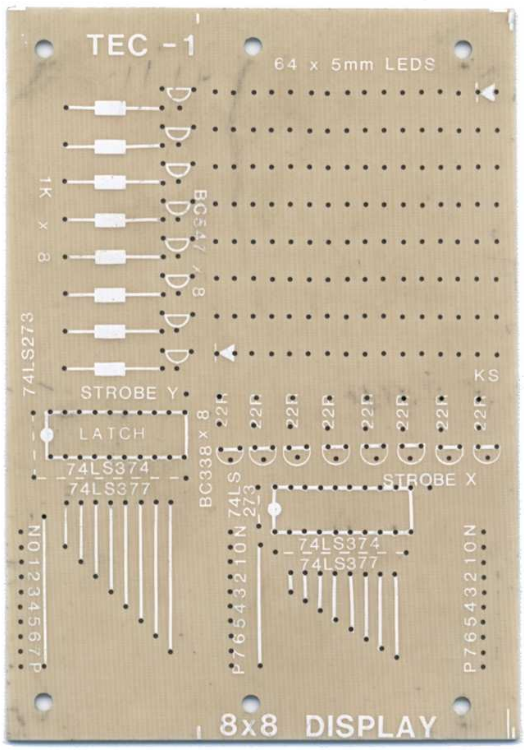
MAGIC SQUARE PROGRAM

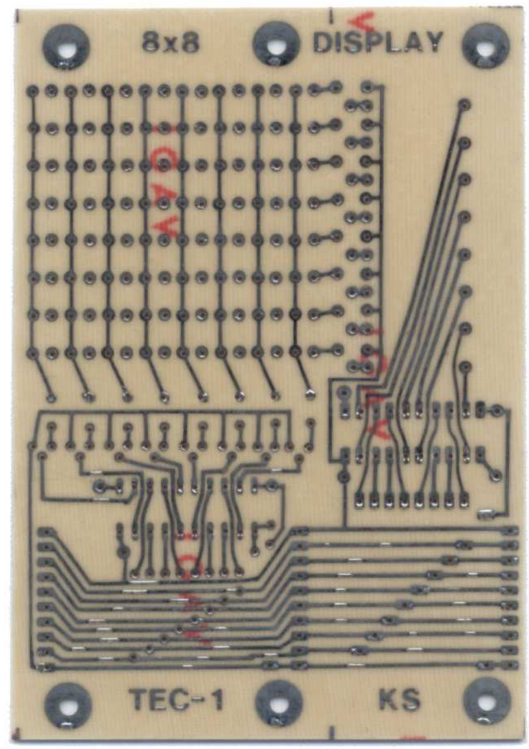
	0C00	11 00 00	LD DE,0000	Random number generated
	0C03	13	INC DE	by the duration it takes the player to release the key at the start of the
	0C04	DB 03	IN A,(03)	program.
	0C06	CB 77	BIT 6,A	
	0C08	28 F9	JR Z,0C03	
	0C0A	ED 5F	LD A,R	The value of the refresh register is loaded into the accumulator.
	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	16 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= period  E = loop counter
          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
```




TEC -1
64 x 5mm LEDS
1K x 8
BC547 x 8
74LS273
STROBE Y
LATCH
74LS374
74LS377
BC338 x 8
22P
KS
STROBE X
74LS 273
N01234567P
P76543210N
8x8 DISPLAY


8x8
DISPLAY
TEC-1
KS