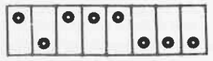
DICE

The DICE Program at **0470** introduces a few more programming skills.



The first of these is a RANDOM NUMBER GENERATOR. Random numbers are almost impossible to generate via a computer due to it being a very predictable machine. The only reliable way to get a random number is to introduce the human element.

This is what we have done in this program.

At the start of the program a running LED routine moves a single LED around the 4x4 matrix. The ON time for each LED is created by a delay routine that uses the B abd C registers. The C register is loaded with 6 and decrements to zero. Each time this is done, the B register is decremented and when it reaches zero, the LED jumps to the next location.

The random number is generated in the C register and we can exit from the program with a value remaining in C. Since C is the inside loop of the delay it is decrementing very fast and it is not possible to predict what value C will contain.

If it were the outside loop it would be a different matter. Players would gradually get to understand that pressing at the beginning of cycle would generate a low number and at the end of a cycle, a high number.

Owing to the unpredictability of the human reaction, an even spread of numbers from 1 to 6 is created with our routine.

The second feature of the program is the COMPARE and BRANCH.

After the random number has been obtained, a number of flashes are created on the screen and then the accumulator is compared with the random number before jumping to the display routine.

This routine is a very simple multiplexing routine in which three bytes are outputted for a period of 80 cycles.

The program then detects that the input button has been released and jumps to the start of the program.

If a button-check was not made, the same number would appear on the displays due to a constant number of cycles occuring in the program for each game.

At 04D3:

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

```
LD D,0C          0470  16 0C     C is the byte table counter for the 4x4
LD HL,04D3       0472  21 D3 04  HL will point to the byte table address
LD A,(HL)        0475  7E        A is loaded with the value of the first byte in the table.
OUT (02),A       0476  D3 02     The accumulator is outputted to port 02.
INC HL           0478  23        The byte table pointer is incremented.
LD B,15          0479  06 15     Load B with 15, for a delay value of 21 loops.
LD C,06          047B  0E 06     Load C with 6, for the dice values: 0-6.
IN A,(01)        047D  DB 01     Input from the input port to the accumulator.
BIT 7,A          047F  CB 7F     Check to see if button A has been pressed.
JR NZ,048D       0481  20 0A     If pressed, jump out of the delay routine.
DEC C            0483  0D        Decrement register C.
JR NZ,047D       0484  20 F7     If C is not zero, jump up, If C zero, advance.
DJNZ 047B        0486  10 F3     Decrement B and if not zero, jump up.
DEC D            0488  15        Decrement the byte table register D.
JR Z,0470        0489  28 E5     When D is zero, jump to start of program.
JR 0475          048B  18 E8     If not zero, continue DELAY ROUTINE.
LD D,06          048D  16 06     Load D with 6 for six flashes of the display.
LD A,0F          048F  3E 0F     Load A to turn on the whole 4x4 display.
OUT (02),A       0491  D3 02     Output to port 02.
DJNZ 0493        0493  10 FE     Register B is decremented to create a delay.
LD A,FF          0495  3E FF     Load A with a value to turn 4x4 OFF.
OUT (02),A       0497  D3 02     Output to port 02.
DJNZ 0499        0499  10 FE     Create a short delay with register B.
DEC D            049B  15        Decrement the flash-count register.
JR NZ,048F       049C  20 F1     Loop for 6 flashes.
LD D,80          049E  16 80     Load D for 80 loops for multiplexing routine.
LD A,C           04A0  79        Load our random number into the accumulator.
LD HL,04E0       04A1  21 E0 04  Load HL with address of table for multiplex routine.
CP 01            04A4  FE 01     Compare the accumulator with 1.
JP Z,045F        04A6  CA F5 04  If the accumulator is 1, jump to multiplex routine.
LD HL,04E3       04A9  21 E3 04  Load HL with start address for displaying '2'.
CP 02            04AC  FE 02     Compare the accumulator with 2.
JP Z,045F        04AE  CA F5 04  If accumulator is 2, jump to multiplex routine.
LD HL,04E6       04B1  21 E6 04  Load HL with start-address for displaying '3'.
CP 03            04B4  FE 03     Compare accumulator with 3.
JP Z, 045F       04B6  CA F5 04  If accumulator is 3, jump to multiplex routine.
LD HL,04E9       04B9  21 E9 04  Load HL with start-address for displaying '4'.
CP 04            04BC  FE 04     Compare the accumulator with 4.
JP Z,045F        04BE  CA F5 04  If accumulator is 4, jump to multiplex routine.
LD HL,04EC       04C1  21 EC 04  Load HL with start-address for displaying '5'.
CP 05            04C4  FE 05     Compare accumulator with 5.
JP Z,04F5        04C6  CA F5 04  If accumulator is 5, jump to multiplex routine.
LD HL,04EF       04C9  21 EF 04  Load HL with start-address for displaying 6.
CP 06            04CC  FE 06     Compare accumulator with 6.
JP Z,04F5        04CE  CA F5 04  Jump to multiplex routine is accum is 6.
```

At 04E0: A jump value must be found and the micro jumps to the multiplexing routine below and produces a display on the 4x4 that is similar to the spots on the face of a dice. The routine runs for 80 loops , makes sure button A is not pressed, then jumps to the start of the DICE program.

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

```
LD A,(HL)        04F5  7E        Load A with the value pointed to by HL.
OUT (02),A       04F6  D3 02     Output the value to port 02.
LD B,0A          04F8  06 0A     Load B with a short delay value.
DJNZ 04FA        04FA  10 FE     Create a short delay with register B.
INC HL           04FC  23        Point to next display address
LD A,(HL)        04FD  7E        Load the value pointed to by HL into A.
OUT (02),A       04FE  D3 02     Output to port 02.
LD B,0A          0500  06 0A     Load B with a short delay value.
DJNZ 0502        0502  10 FE     Create a short delay with register B.
INC HL           0504  23        Inc HL to look at next address.
LD A,(HL)        0505  7E        Load value pointed to by HL in the accumulator.
OUT (02),A       0506  D3 02     Output to port 02.
LD B,0A          0508  06 0A     Load register B with a short delay value.
DJNZ 050A        050A  10 FF     Decrement register B to zero.
DEC HL           050C  2B        Dec HL to look at start of display table.
DEC HL           050D  2B
DEC D            050E  15        Decrement multiplex routine loop counter.
JR NZ, 04F5      050F  20 E4     Loop again if D is not zero.
XOR A            0511  AF        Zero the accumulator and output to port 02 to
OUT (02),A       0512  D3 02     blank the display.
IN A,(01)        0514  DB 01     Look at the output port to see if button A is NOT
BIT 7,A          0516  CB 7F     pressed before re-starting the DICE program.
JR NZ,0514       0518  20 FA     Loop if A is pressed.
JP 0470          051A  C3 70 04  Jump to start of DICE program.
```


MICRO-COMP PART III

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

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

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

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

MODS

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

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

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

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

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

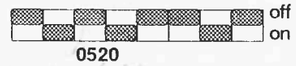
Now, to the programs:

0520 EPROM IN BINARY

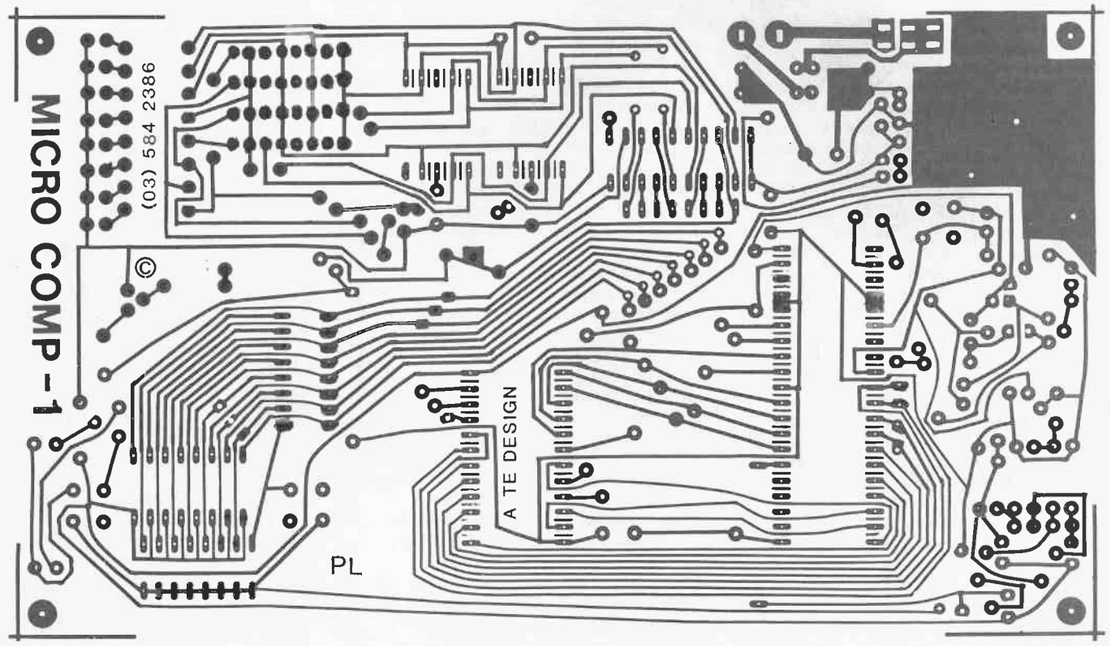
This is a program that displays the contents of the EPROM on the set of 8 LEDs.

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

The input switches are positioned thus:



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



Artwork for Micro-Comp

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

EPROM DISPLAYED IN BINARY:

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

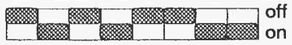
POKER PROGRAM:

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

0530 POKER

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

The input switches are positioned as follows:



0530

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

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

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

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

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

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

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

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

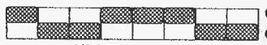
The aim of the program is to produce multiplexing on the 4x4 so that each column can be incremented separately. The program is long because we haven't taken any short cuts.

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

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

0630 BINARY CLOCK

Here's the switch positions:



0630

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

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

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

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

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

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

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

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

See over for Binary Clock tables.

BINARY CLOCK PROGRAM

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

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

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

ADJUSTING FOR 1 MINUTE

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

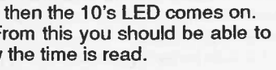
Make sure the brightness of the 4x4 is maximum by shorting between base and emitter of both cathode driver transistors.

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

at 06BA:
F1
E1

06C0 ONE MINUTE TIMER

These are the switch positions:



06C0

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

1 MINUTE TIMER PROGRAM

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

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

06D0 3 MINUTE TIMER

The switch settings:

06D0

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

3 MINUTE TIMER PROGRAM

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

06E0 1 HOUR TIMER

The switch settings:

06E0

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

1 HOUR TIMER PROGRAM

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

06F0 ADJUSTABLE TIMER

The switch settings:

06F0

ADJUSTABLE TIMER PROGRAM

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

MAIN DELAY PROGRAM:

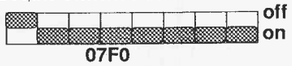
LD I,A	0740	ED 47
LD DE,8BFF	0742	11 FF 8B
DEC DE	0745	1B
LD A,E	0746	7B
OR D	0747	B2
JR NZ,0745	0748	20 FB
LD A,I	074A	ED 57
DEC A	074C	3D
JR NZ,0740	074D	20 F1
DJNZ,074F	074F	10 FE
LD A,81	0751	3E 81
OUT (02),A	0753	D3 02
LD A,00	0755	3E 00
OUT (02),A	0757	D3 02
IN A,(01)	0759	DB 01
BIT 7,A	075B	CB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

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

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

07F0 FINAL MESSAGE

Set the switches thus:



07F0

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

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

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

MICROCOMP TUTORIAL

Some more notes on the Microcomp programs

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

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

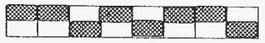
Although this arrangement may never be used in a full-size design, it shows how 8 lines can be arranged to drive a matrix of 16 LEDs and produce the beginnings of a large display.

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

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

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

Switch the input switches to 0290:



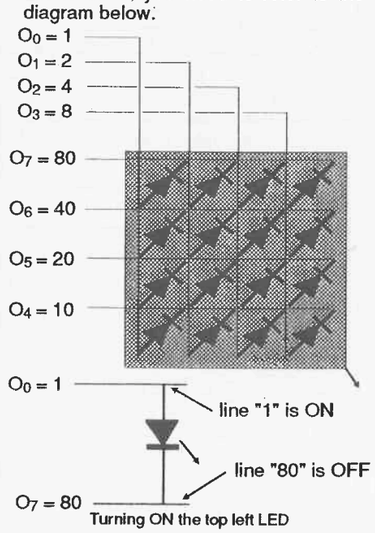
and push reset. Turn all the switches off and the LEDs on the 4x4 will go out.

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

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

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

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



Turning ON the top left LED

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

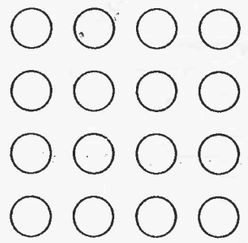
The simplified diagram shows this more clearly. If "80" is off, it means switches 40, 20, 10 are ON and thus the value needed to turn on the LED is 40 + 20 + 10 + 1 = 71.

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

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

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

Place your results here:



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

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

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

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

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

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

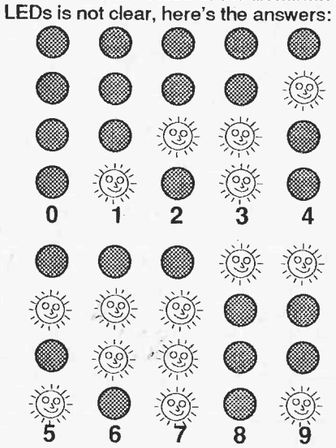
One of the last jobs in finalising the program is to adjust the delay lengths to get the speed right.

CLOCK PROGRAM

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

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

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



CREATING MESSAGES

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

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

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

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

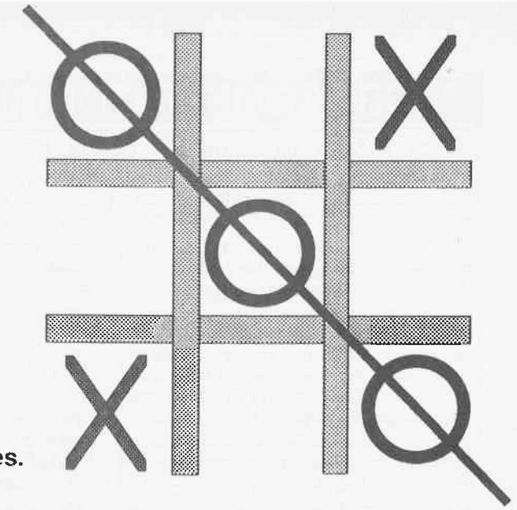
When more displays are used, the possibilities for displaying all kinds of effects becomes quite challenging. You can get running letters, flashing words and complete sentences appearing at any speed you wish.

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

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

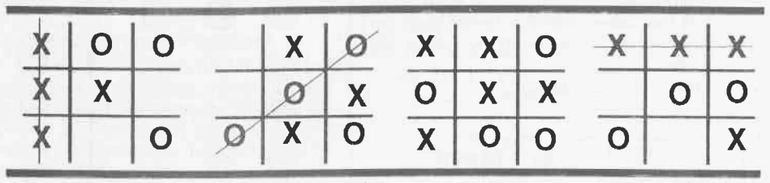
Next time we will have a great add-on for the Micro- Comp. Look out for it.

NOUGHTS & CROSSES



AN "ADD-ON" FOR THE MICROCOMP

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



Parts & PC: $29.40
PC board only: $4.80

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

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

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

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

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

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

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

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

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

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

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

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

THE RULES

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

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

ABOUT THE MICRO-COMP

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

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

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

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

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

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

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

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

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

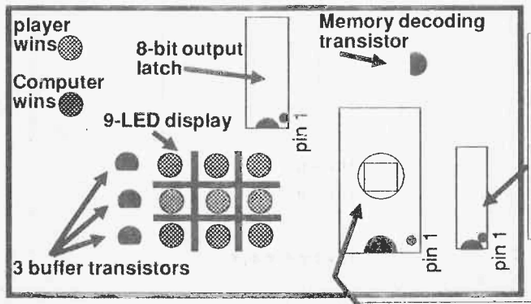
0's & X's IS AN 'ADD-ON'

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

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

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

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



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

EPROM containing the Noughts & Crosses program

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

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

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

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

THE PROGRAM

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

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

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

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

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

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

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

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

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

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

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

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

The computer must recognise this

0801	0802	0803
0804	0805	0806
0807	0808	0809

The positions on the board corresponding to the memory locations

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

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

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

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

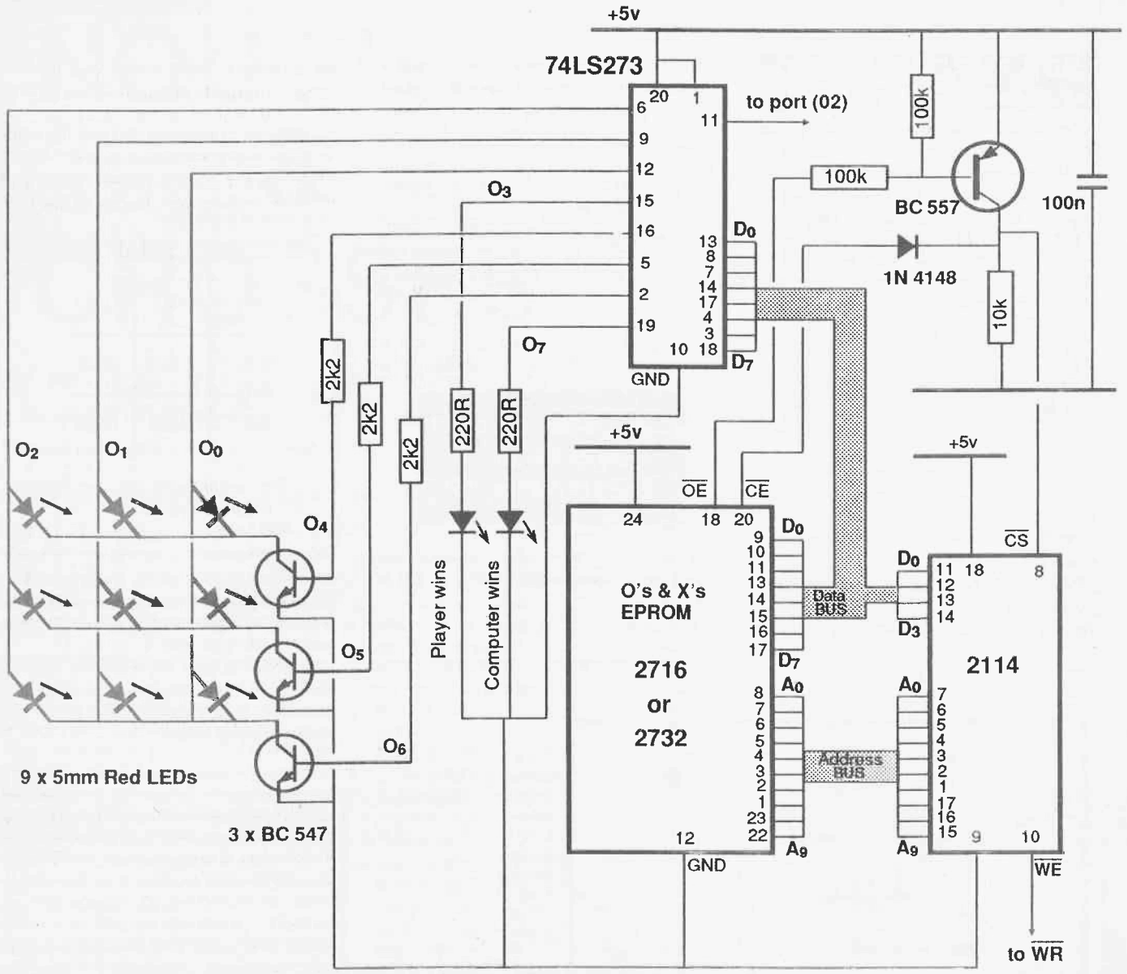
As you have seen, the basic Microcomp project runs without any RAM and if RAM is not available, the program cannot use commands such as PUSH and POP since a stack is not available.

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

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

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

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



NOUGHTS AND CROSSES CIRCUIT

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

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

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

The latch is accessed at port (02).

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

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

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

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

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

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

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

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

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

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

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

HOW THE CURSOR LED WORKS

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

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

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

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

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

THE ALGORITHM

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

PARTS LIST

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

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

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

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

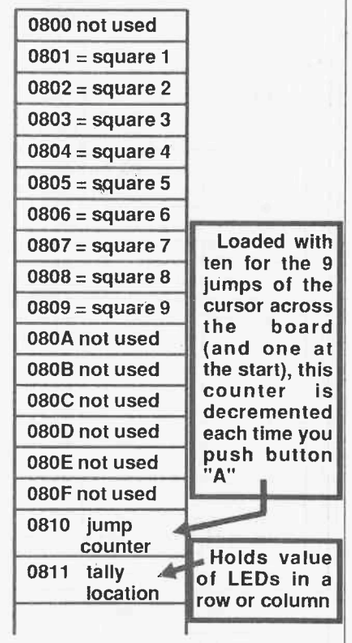
1 - Noughts & Crosses PC board

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

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

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

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



Inside the 2114

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

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

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

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

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

1	2	3
4	5	6
7	8	9

The numbering of the squares on the board

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

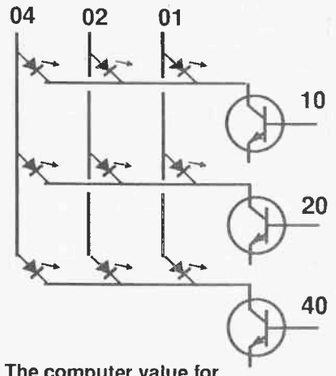
Next the program looks for a 'stopper' This is detected by looking for 02 in a row, column or diagonal and represents 2 player pieces. It responds by inserting a computer move into the empty location and returning to SCAN.

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

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

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

The program must detect if the player



The computer value for each column and row.

NOUGHTS AND CROSSES PROGRAM

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

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

C

A B X

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

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

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

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

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

Button B is read

Button A is read

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

Bit 0,C prevents switch bounce (see text)

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

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

SCAN ROUTINE FOR COMPUTER AND PLAYER PIECES at 0017

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

SCAN ROUTINE FOR ONLY COMPUTER PIECES at 0146

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

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

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

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


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

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

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

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

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

A B X C C Y Z

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

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

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

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

The flashing cursor is replaced with a player piece.

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

Return to SCAN

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

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

Sub-routine for row values containing computer pieces.

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

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

BUTTON 'A' ROUTINE at 00F1

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

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

BUTTON 'B' DETECTED at 011A

The cursor LED is replaced with a player piece.

LOOKS for a PLAYER WIN at 011F

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

LOOKS for a COMPUTER WIN at 0137

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

Looks for a STOPPER at 0148

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

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

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

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

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

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

Y Z not first move empty W

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

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

Jump to 017C and try again.

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

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

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

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

TABLES:

at 0220:

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

at 0238:

01	01	07
07	03	09
03	09	

at 0240:

02	04	08
08	06	04
06	02	

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

Random corner if first move at 0168

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

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

Looks for a centre square at 0196.

Computer looks for a centre square to

Looks for a '6' at 01A4

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

Looks for a '4' at 01E2

W found not found

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

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

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

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

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

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

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

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

Looks for a 'STALEMATE' at 0248

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

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

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

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

```
at 01E2:
LD B,04     01E2  06 04
LD IY,026C  01E4  DD 21 6C 02
LD IY,0178  01E6  FD 21 78 01
at 026C:
LD A,(0809) 026C  3A 09 08
CP00        026F  FE 00
JP NZ 0155  0271  C2 55 01
LD A, (0806) 0274 3a 06 08
CP 01       0277  FE 01
JR Z,0283   0279  28 08
LD A,(0808) 027B  3A 08 08
CP 01       027E  FE 01
JP NZ,0155  0280  C2 55 01
LD A,04     0283  3E 04
LD (0809),A 0285  32 09 08
JP 012E     0288  C3 2E 01
```

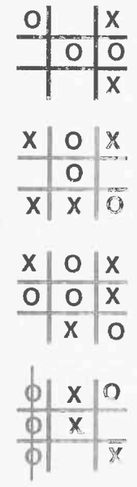
Look for a computer piece.
IX contains the "Found" return address
IY contains the NOT FOUND return address.

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

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

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

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



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

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

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

HOW THE PROGRAM RUNS

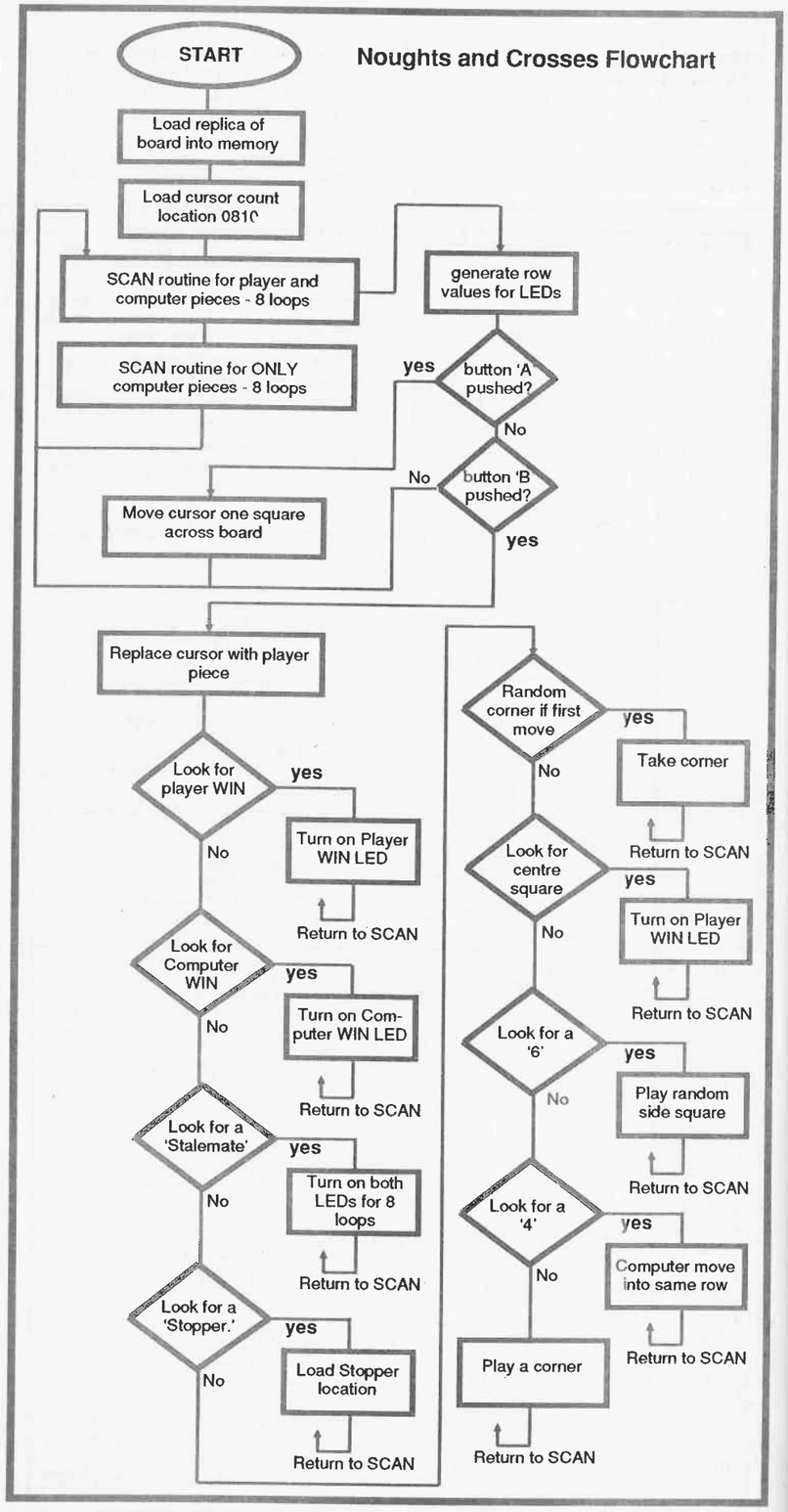
The program is running all the time. Most of the time it is executing the SCAN routine which lights up the LEDs in the display. The top row of LEDs are the first to be illuminated. The routine determines if the first LED is to be lit by looking at memory location 0801. If this location is not zero, 04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

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

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

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

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



of the output latch.

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

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

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

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

SCANNING THE DISPLAY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Let's look at the reason why.

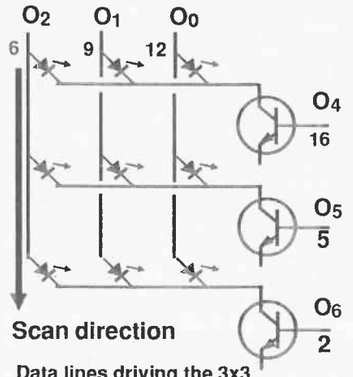
To turn on the top left-hand LED, the values are 04 and 10. To turn on the bottom right-hand LED the values are 01 and 40. Putting these values together we get

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

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

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

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



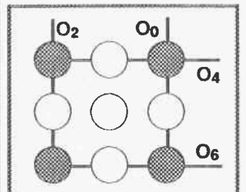
Data lines driving the 3x3 display

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

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

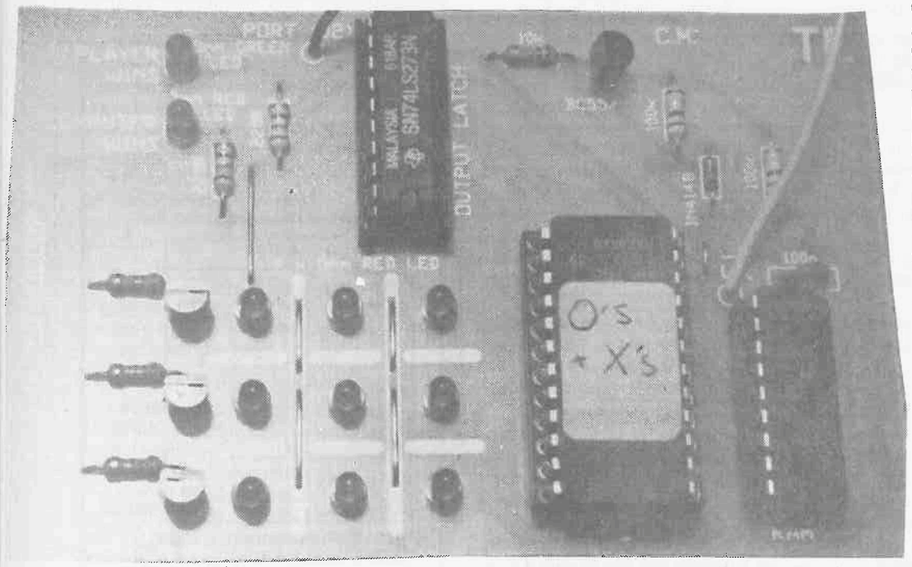
Hex value		Binary value		Effect on display
Bit 0	01	0000	0001	drives col 3
Bit 1	02	0000	0010	drives col 2
Bit 2	04	0000	0100	drives col 1
Bit 3	08	0000	1000	"Player wins" LED
Bit 4	10	0001	0000	drives row 1 trans.
Bit 5	20	0010	0000	" row 2 transistor
Bit 6	40	0100	0000	" row 3 transistor
Bit 7	80	1000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

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



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

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



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

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

SWITCH DEBOUNCE

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

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

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

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

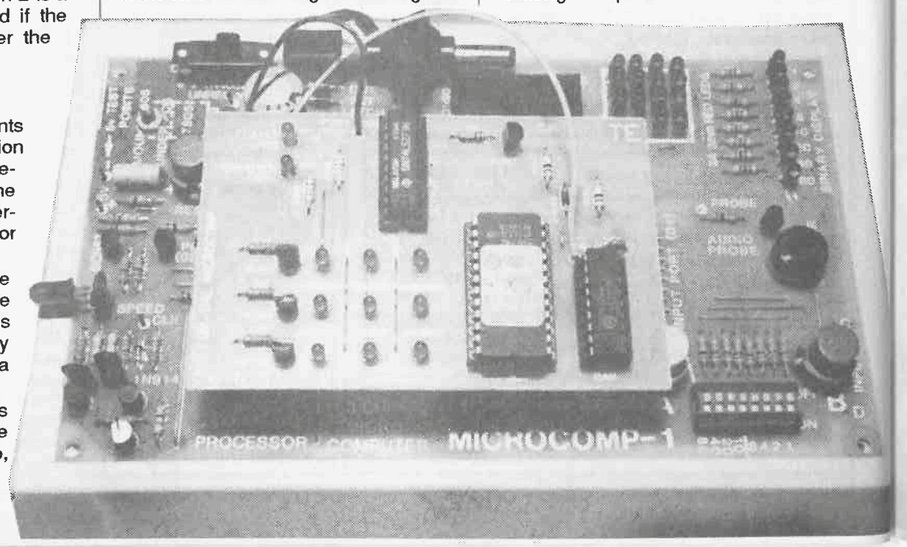
This is obviously not acceptable as it will produce a false count and the only way to overcome it is to produce a key scan program that will create a delay between opening and closing of the switch.

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

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

WIRING THE CHIPS

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



CONSTRUCTION

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

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

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

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

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

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

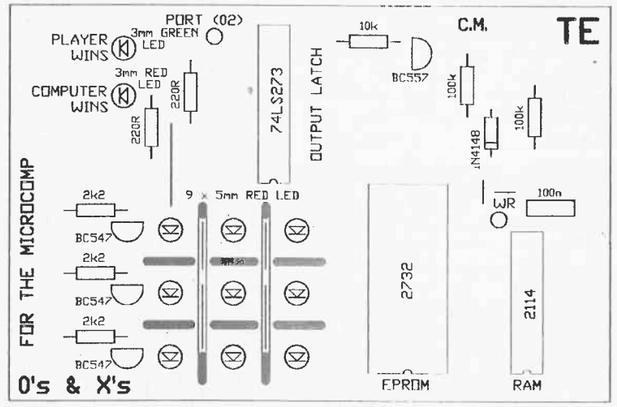
Using an IC socket like this is not as good as a component header as it is not as strong, But it is much cheaper and an acceptable alternative when headers are out of stock.

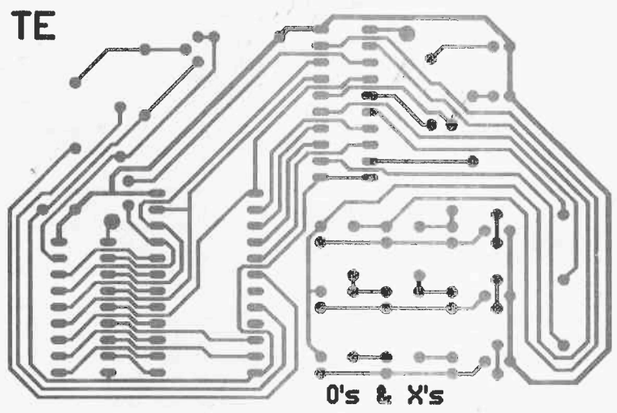
Next fit the jumpers. Four of these are on the display, one is above it and one is on the other side of the EPROM.

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

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

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





Full-size artwork for Noughts and Crosses

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

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

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

IF IT DOESN'T WORK

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

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

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

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

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

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

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

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

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

You can check the 0's and X's ROM to see if it has a program by placing it in the Microcomp socket. The right hand display will come on to show that the program is scanning the display and if you turn the speed down, the segments will flicker.

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

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

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

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

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

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

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

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

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

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

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

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

HOW TO PLAY

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

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

ACTIVATING THE MEMORY

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

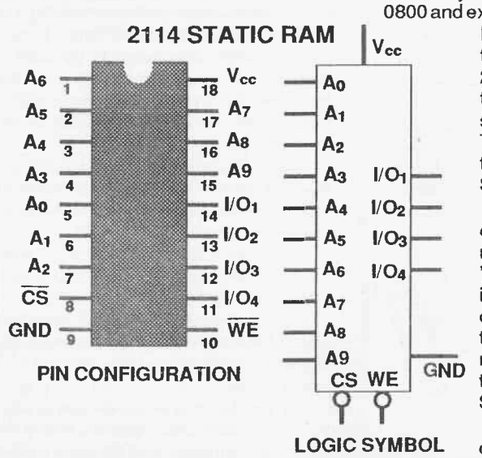
A 2114 memory chip has been used for this and is capable of holding a lot more information than we need. They are very old technology devices and are no longer manufactured but can be obtained from old computer boards.

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

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

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

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



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

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

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

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

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

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

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

MORE PROGRAMMING IDEAS

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

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

I know the choice I prefer.

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

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

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

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

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

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

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

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

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

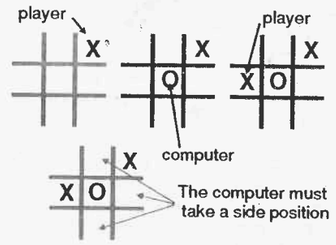
Programs are kept as short as possible by creating sub-routines that can be called by the main program. Once the sub-routine is executed, any results that are generated are stored in a register or in RAM.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

THE FUTURE OF PROGRAMMING

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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