THE INS AND OUTS OF THE

TIMEX TS1000 & ZX81

DON THOMASSON

A HARDWARE MANUAL FOR TS1000 & ZX81

THE INS AND OUTS OF THE

TIMEX TS1000 & ZX 81

DON THOMASSON

MELBOURNE HOUSE

Published in the United Kingdom by:
Melbourne House (Publishers) Ltd.,
Glebe Cottage, Glebe House,
Station Road, Cheddington,
Leighton Buzzard, Bedforshire, LU7 7NA,
ISBN 0 86161 118 7

Published in Australia by:
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria, 3205,
National Library of Australia Card Number and
ISBN 0 86759 119 6

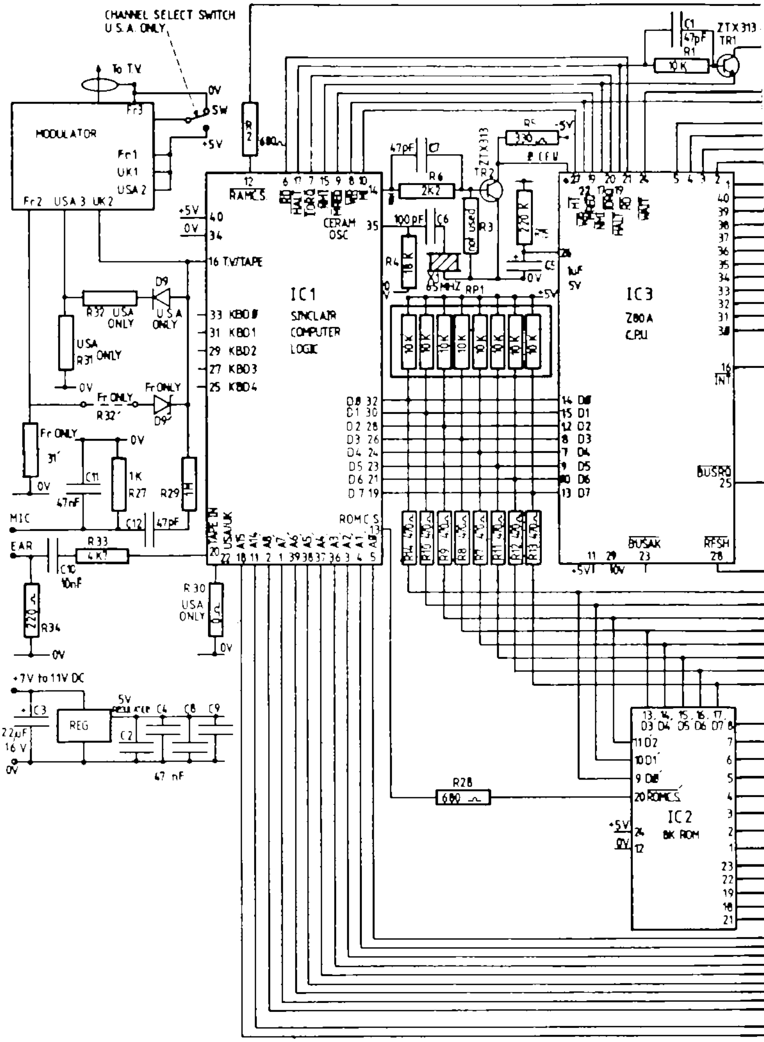
Published in the United States of America by:
Melbourne House Software Inc.,
347 Reedwood Drive,
Nashville NT 37217.

Printed in Hong Kong by Colorcraft Ltd.

Contents

Illustrations

Tables



CHANNEL SELECT SWITCH
U.S.A. ONLY
To T.V.
MODULATOR
0V
SW
+5V
IC1
SINCLAIR
COMPUTER
LOGIC
T.V/TAPE
KBD1
KBD2
KBD3
KBD4
CERAM
OSC
RAMCS
ROMCS
IC3
Z80A
CPU
INT
BUSRQ
BUSAK
RFSH
IC2
8K ROM
ZTX313
TR1
TR2
RP1
REG
MIC
EAR
+7V to 11V DC
USA ONLY
Fr ONLY
47 nF

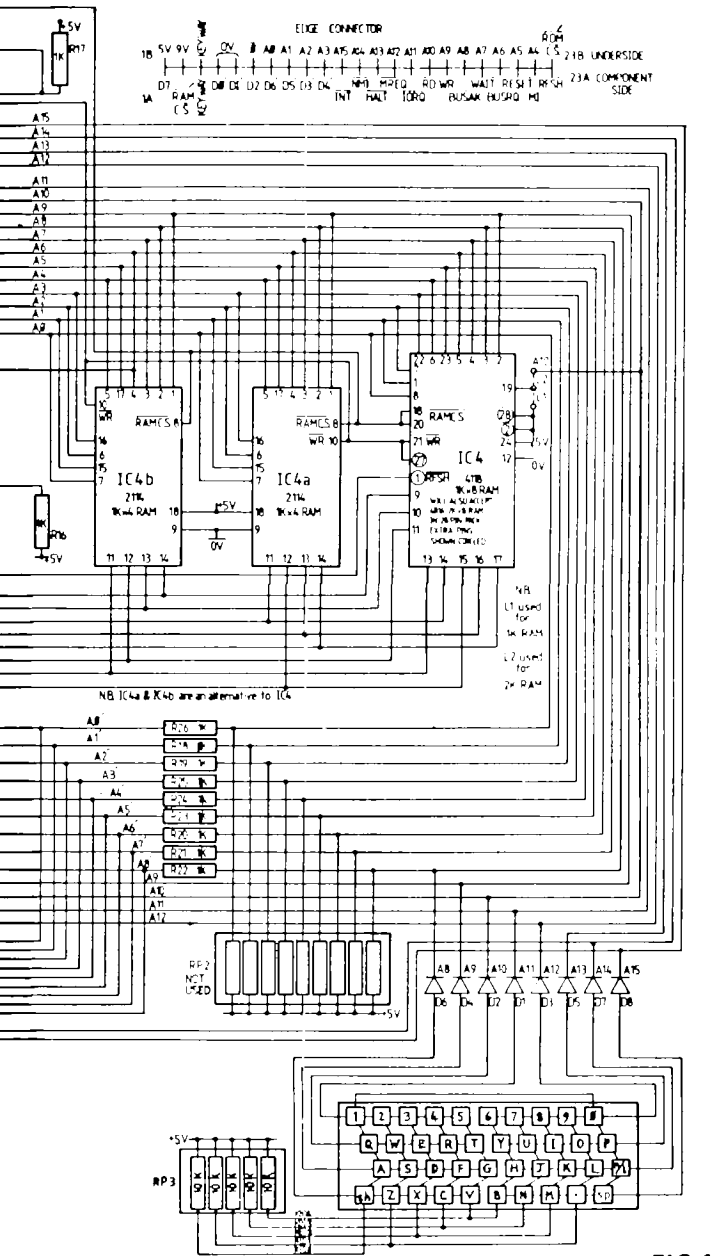

EDGE CONNECTOR
UNDERSIDE
COMPONENT SIDE
IC4d
IC4a
IC4
NB IC4a & IC4b are an alternative to IC4
RP2 NOT USED
RP3

FIG 1

INTRODUCTION

A computer, even one as small and inexpensive as the TS1000 or the ZX81, can provide seemingly endless interest. There is so much to be discovered, so much pleasure in the realisation that writing programs is not impossibly difficult, so much satisfaction in the tracking down of elusive 'bugs'.

In time, however, even this brave new world may fade a little, the shine of novelty wearing away, and that is the signal to explore further, to find out what is inside the box, to connect external bits and pieces, to make the computer do something new. This book is your guide in that exploration, your map of safe paths, your warning of dangers.

The first warning is perhaps the most important. Computer components are not partial to being handled too casually. Some electrical work, such as the addition of extra accessories to a car, is unlikely to land anyone in serious trouble. The battery may go flat, or some wires may overheat, but unless the car itself catches fire the damage can usually be mended fairly easily. Putting a damaged computer right can be a lot more difficult.

The most common source of trouble is due to hamfisted use of the soldering iron, which can overheat components or wreck printed circuit boards. Those who doubt their skill in this direction should practice on something less vulnerable before they tackle changes to the ZX81. If the soldering iron is removed as soon as it has done its work, there should be no difficulty. The trick lies in judging when the work is done.

Another cause of damage is static electricity. Most computer components are designed to work at relatively low voltages, and the charges that make your clothes crackle when you take them off in dry weather would be more than enough to punch holes in the insulation of many chips. Walking across a nylon carpet can charge you to similar levels. It does you no harm, because the power involved is minute, but it can make you fatal to any integrated circuit which you touch.

It is therefore wise to guard against static build up, by simple precautions that should become a habit. If you always touch an earth point on a component or circuit before you touch anything else, the voltage levels will be equalised, removing the danger. Those who handle computer systems professionally may be required to observe rather more stringent rules, but equalisation of voltage levels remains the key point.

Remember that the soldering iron is connected to earth, so before you use it you should earth both yourself and the equipment.

You should also switch power off before making any changes to the system. Clearance between pins and circuit tracks is so small that accidental shorts are only too easy to create. Before restoring power, make a careful check for stray whiskers of wire and splashes of solder, which can cause expensive damage, and are likely to produce obscure faults that are difficult to trace.

You should not allow yourself to be too discouraged by these warnings. Keeping to the rules should see you through.

You may be aware that the TS1000 and ZX81 computers are essentially the same machine: The ZX81 was developed by Sinclair Research in 1981 in the UK and is still marketed by Sinclair in the UK and distributed by them through much of the world. The TS1000 is manufactured by the Timex Corporation under licence from Sinclair Research, and the major difference is that TS1000s come with 2K of RAM, while the ZX81s come with 1K. In this book, any information about the ZX81 also refers to the TS1000, unless specifically noted. For ease of reading, the two computers are simply referred to as the ZX81.

The book is divided into three parts: The first examines the inside of the ZX81, the second deals with the external interface, and the third describes some systems which can be added to the basic ZX81 via that interface. The division is not rigid, because most of the changes that can be made affect more than one area, but in broad terms each section prepares the ground for what comes next.

INTERNALS

Before looking inside the ZX81, let us consider what we expect to find. There will be a microprocessor, some RAM (Random Access Memory) for it to talk to, and some fixed data in ROM (Read Only Memory) to tell the processor what to do. There must be some provision for communication with the outside world, or we would never find out what was happening inside the system.

That description of a minimum computer system also constitutes a fair definition of the ZX81, but two factors give the machine a unique personality, the programs which are held in the ROM and the highly specialised ULA (Uncommitted Logic Array) which the manual calls the 'dogsbody'. The program and the ULA complement each other, and changing either would be both difficult and expensive, which places a limit on the alterations which can be made to the system as a whole. Within that limit, however, quite a lot can be done.

The key feature of the ZX81 is its apparent ability to maintain a video display with minimal resources. The task is demanding. Line synchronising pulses must be generated every 64 uS, with a modification of their pattern to produce frame synchronising pulses. During 40 uS periods between line sync pulses a string of up to 256 data bits must be output to form the display, at a rate of 6½ bits per microsecond. Some computers need 30 integrated circuits to achieve this.

In SLOW mode, the ZX81 appears to handle the display output tasks while it is busy executing a program, but that is an illusion. The program runs for only about a quarter of the time, during the periods when the display scan is not demanding detailed output data. While the pattern data is being output, the whole system is occupied with that task, and normal processing is suspended.

In FAST mode, the display is abandoned to its own devices, so processing can be continuous, and therefore four times as fast.

To refresh the display, it is necessary to read codes from the display file in RAM, use them to form pointers to character pattern data in ROM, then transmit the pattern bits serially as video output. All parts of the system are involved, because the processor supplies the base address used in accessing the pattern data. The 'dogsbody' chip controls the whole operation.

Quite apart from this, the ULA chip has other duties:

- Generation of a 6.5 MHz clock, using a ceramic filter as a frequency standard.
- Derivation of a 3.25 MHz clock for the processor.
- Driving the tape recorder in SAVE
- Responding to tape output in LOAD
- Sensing keyboard action
- Deciding when ROM and RAM should be active, on the basis of the address provided by the processor.
- Controlling general system timing.

It is scarcely surprising that some of these tasks come into conflict with each other, imposing system limitations. The display cannot be maintained during SAVE and LOAD, which must act continuously to maintain the correct baud rate for the data passing to and from the tape. In any case, the same pin on the ULA provides both the video signal and the output to the tape recorder, hence the odd patterns on the screen when tape is in use.

Another point is that software-controlled timing of inputs or outputs cannot be completely precise. The stop-go processing must introduce some degree of jitter. For longer intervals, the PAUSE command provides a solution, but such tasks as output waveform generation must be performed in FAST mode.

The other principal factor limiting the system performance is the ROM-borne program. This is entirely conditioned by the assumption that BASIC will normally be used, and in that role

it performs very well, but the use of machine code is not as simple as might be wished. It can be run, but only via BASIC.

A rather important deficiency is the lack of IN and OUT instructions in the BASIC repertoire, which means that they have to be implemented in machine code if external devices are to be brought into play.

However, most of the limitations mentioned can be overcome by a little ingenuity, once their nature and origin are understood.

Getting Inside

Opening up the ZX81 is no problem once you know that three screws are hidden under the stick-on rubber feet. Remove these and the two visible screws, and the underside of the case, which is really the cover, will come away, revealing the underside of the circuit board, which is held in place by two more screws. As the screws are of different sizes, it is useful to make a note of their positions.

When the last screws have been removed, the circuit board can be turned over, but it will still be attached to the case by the keyboard connections, so handle it carefully. Touch the case of the video modulator first, to equalise voltages and avoid static damage. The modulator is easy to identify by its metal case and the connector marked 'TV'.

The circuit board carries four — sometimes five — integrated circuits, the modulator, and a number of smaller components. The metal case is a heat sink to help cool the voltage regulator, to which it is bolted. If that is on your right, furthest away from you, the 'dogsbody' chip is in front of it, next to the modulator, then comes the ROM chip, and then the processor. On the left is the RAM, which may be implemented as one 24-pin chip or two 18-pin chips, the board layout allowing for either configuration. The ZX81 has 1K of RAM, the TS1000 has 2K.

Of the smaller components, the keyboard connector is obvious because the keyboard film wire goes to it, the clock

circuits are on the near side of the board, close to the 'dogs-body' chip, together with some odd circuit elements, the area near the three jack sockets contains filter circuits for the tape input and output and for some video output standards, and most of the remaining components are resistors, diodes, or decoupling capacitors. If the details of the assembly fail to match the picture in the manual, there is no need to worry, as the layout permits a number of variations.

One variation is the omission of sockets for the RAM, which is a pity, as it makes modification in this area more difficult.

The printed circuit can be separated from the keyboard by pulling the film wires out of their sockets, and this makes handling easier, but it should be done sparingly, as the wires are easily damaged.

The internal circuits consume about 300 mA, and the external power unit is capable of delivering up to 700 mA, but a second limit on available current is set by the internal regulator, which gets rather hot under the collar if it is asked to do too much. It can be loaded by about another 50-100 mA, but if more power is required it should be obtained from a separate regulator driven from the +9v supply rail. The outputs of the two regulators should not be joined together, as has been done in some proposals, as one will then tend to take all the load.

The Circuit

The circuit diagram (Fig. 1) is not too easy to follow, so it has been supported by a signal list showing the main interconnections. (Table 1).

The system is broadly what one would expect, but there are a few surprises. For example, address lines A0-A8 go directly from the processor to the RAM, but are connected to the ULA and ROM by series resistors, allowing the ULA to control the ROM address directly when picking up pattern data, as will be explained when the display system is examined. Similarly, the data lines go directly from the processor to the ULA, but are passed to the ROM, the RAM and the outside world

ZX81 SIGNAL LIST (1)

SIGNAL NAME	IC3 CPU	IC4a RAM	IC4b RAM	IC4 24-pin RAM	IC4 28-pin RAM	IC2 ROM	IC1 ULA	External	Other	Notes
Address Lines										
A0	30	7	7	8	10	—	—	7B	R26	R18-R26: 1K. See A0'-A8'
A1	31	15	15	1	3	—	—	8B	R18	
A2	32	6	6	7	9	—	—	9B	R19	
A3	33	16	16	22	24	—	—	10B	R25	
A4	34	5	5	6	8	—	—	22B	R24	
A5 ___	35	17	17	23	25	—	—	21B	R23	
A6/INT	36/16	4	4	5	7	—	—	20B/11A	R20	
A7	37	3	3	4	6	—	—	19B	R21	
A8	38	2	2	3	5	—	—	18B	R22,	KD6 KD = Keyboard diode.
A9	39	1	1	2	4	22	—	17B	KD4	
A10	40	—	—	See Note		19	—	16B	KD2	Link L2 connects A10 to
A11	1	—	—	—	—	18	—	15B	KD1	pin 19 of a 24-pin RAM,
A12	2	—	—	—	—	21	—	14B	KD3	pin 21 of a 28-pin RAM.
A13	3	—	—	—	—	—	—	13B	KD5	Link L1 replaces A10 by
A14	4	—	—	—	—	—	11	12B	KD7	+5v
A15	5	—	—	—	—	—	18	11B	KD8	

ZX81 Signal List (2)

A0'	—	—	—	—	—	8	5	—	R26	Secondary Address Lines,
A1'	—	—	—	—	—	7	4	—	R18	linked to the main lines
A2'	—	—	—	—	—	6	3	—	R19	by resistors

A3′		—	—	—	—	—	5	36	—	R25	
A4′		—	—	—	—	—	4	37	—	R24	
A5′		—	—	—	—	—	3	38	—	R23	
A6′		—	—	—	—	—	2	39	—	R20	
A7′		—	—	—	—	—	1	1	—	R22	
A8′		—	—	—	—	—	23	2	—	R21	
Data Lines											
DO	14	—	—	—	—	—	—	32	—	R14,RP1	R7–14: 470 ohms
D1	15	—	—	—	—	—	—	30	—	R10,RP1	RP1: 10K to +5v
D2	12	—	—	—	—	—	—	28	—	R9,RP1	
D3	8	—	—	—	—	—	—	26	—	R8,RP1	
D4	7	—	—	—	—	—	—	24	—	R7,RP1	
D5	9	—	—	—	—	—	—	23	—	R11,RP1	
D6	10	—	—	—	—	—	—	21	—	R12,RP1	
D7	13	—	—	—	—	—	—	19	—	R13,RP1	

ZX81 Signal List (3)

DO′	—	—	14	9	11	9	—	4A	R14	Secondary Data Lines,
D1′	—	—	13	10	12	10	—	5A	R10	linked to the main data
D2′	—	11	—	14	16	11	—	6A	R9	lines by resistors.
D3′	—	13	—	16	18	13	—	9A	R8	
D4′	—	14	—	17	19	14	—	10A	R7	
D5′	—	—	12	11	13	15	—	8A	R11	
D6′	—	—	11	13	15	16	—	7A	R12	
D7′	—	12	—	15	17	17	—	1A	R13	

Other Lines

$\overline{\text{NMI}}$	17	—	—	—	—	—	15	12A	Emitter of TR1
$\overline{\text{HALT}}$	18	—	—	—	—	—	17	13A	Base of TR1 via R1/C1.

$\overline{MREQ}$	19	—	—	—	—	—	9	14A	—
$\overline{IORQ}$	20	—	—	—	—	—	7	15A	—
$\overline{RD}$	21	—	—	—	—	—	6	16A	—
$\overline{WR}$	22	10	10	21	23/27	—	8	17A	—
$\overline{BUSAK}$	23	—	—	—	—	—	—	18A	—
$\overline{WAIT}$	24	—	—	—	—	—	—	19A	Collector of TR1. R17 to +5v
$\overline{BUSRQ}$	25	—	—	—	—	—	—	20A	R16 to +5v
$\overline{RESET}$	26	—	—	—	—	—	—	21A	R15 to +5v. C5 to ground.

ZX81 Signal List (4)

$\overline{M1}$	27	—	—	—	—	—	10	22A	—
$\overline{RFSH}$	28	—	—	—	1	—	—	23A	—
ϕ	6	—	—	—	—	—	—	6B	Collector of TR2, R5 to +5v
+5v	11	18	18	24	2/26/28	24	40	1B	Numerous points.
Ground	29	9	9	12	14	12	34	4B/5B	Numerous points.
$\overline{RAMCS}$	—	—	—	—	—	—	12	—	R2 R2:680 ohms
$\overline{RAMCS}'$	—	8	8	18/20	20/22	—	—	2A	R2
$\overline{ROMCS}$	—	—	—	—	—	—	13	—	R28 R28:680 ohms
$\overline{ROMCS}'$	—	—	—	—	—	20	—	23B	R28
KBD0	—	—	—	—	—	—	33	—	RP3 (10K to +5v). Keyboard
KBD1	—	—	—	—	—	—	31	—	RP3. Keyboard
KBD2	—	—	—	—	—	—	29	—	RP3. Keyboard
KBD3	—	—	—	—	—	—	27	—	RP3. Keyboard
KBD4	—	—	—	—	—	—	25	—	RP4. Keyboard
USA/UK	—	—	—	—	—	—	22	—	Linked to Ground for USA standard
CLOCKOUT	—	—	-	—	—	—	14	—	Base of TR2 via R6/C7
TV/TAPE	—	—	—	—	—	—	16	—	Modulator. MIC. (via filters)
TAPE IN	—	—	—	—	—	—	20	—	EAR via filter.
CERAMOSC	—	—	—	—	—	—	35	—	R4 to Ground. Ceramic filter via C6
+9v	—	—	—	—	—	—	—	2B	C3 to Ground. Regulator. PSU.

2/26/28

through series resistors.

The clock oscillator consists of a 6.5 MHz ceramic filter working in conjunction with circuits in the ULA. This generates the 'dot frequency' for the display, and the ULA must also divide the clock down to 3.25 MHz to serve the needs of the processor. The lower frequency appears on pin 14 of the ULA, and is amplified by TR2.

The other transistor, TR1, is used in a rather odd circuit. The transistor base goes to $\overline{\text{HALT}}$, and the emitter goes to $\overline{\text{NMI}}$, so collective current will only be drawn if $\overline{\text{HALT}}$ is false (at a positive voltage) and $\overline{\text{NMI}}$ is true (at near-zero volts). When current is drawn, $\overline{\text{WAIT}}$ is pulled down, becoming true. This puts the processor into a wait condition when $\overline{\text{NMI}}$ is true and $\overline{\text{HALT}}$ is not, another specialised function used by the display system.

Oddest of all, however, is the connection of the $\overline{\text{INT}}$ line to address line A6. This generates an interrupt when the current address is of the form XX0X, XX1X, XX2X, XX3X, XX8X, XX9X, XXAX, or XXBX, another element in the Machiavellian complexities of the display system. Unfortunately, this debars the use of interrupt by add-on circuits, a severe limitation.

The video output system caters for three national standards, those of France, Britain and America. In view of the origin of the equipment, it is not surprising that the British standard is the simplest, pin 16 of the ULA being connected directly to the modulator. For the French standard, a diode and two resistors are added, while the American version involves a different diode and resistors, a channel select switch on the corner of the board next to the regulator, and a connection earthing pin 22 of the ULA. The purpose of this link is discussed in the section covering the keyboard.

Very little change is permissible in the parts of the circuit which have been described. The system is too closely integrated to permit change without disruption. The areas yet to be examined, the RAM, the Keyboard, and associated parts of the system, allow more scope for ingenuity.

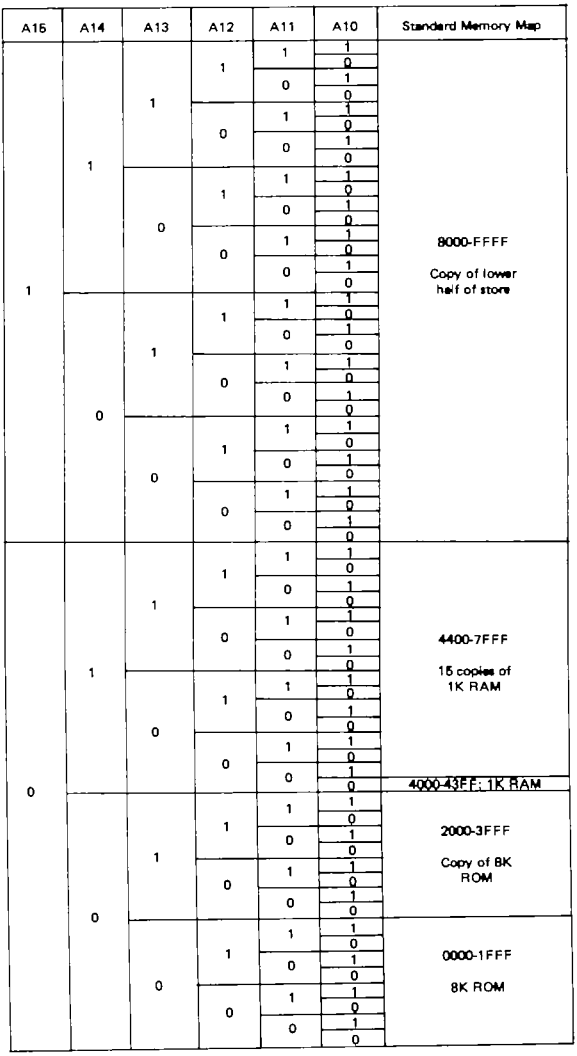


FIG 2 – ADDRESS BITS A10-A15 DIVIDE STORE INTO 64 1024-BYTE SEGMENTS

No reference has been made to the Printer, which is a standard accessory for the ZX81, but which may not be available for use with the TS1000. Virtually no circuitry at all is provided to serve this accessory, which connects to the external connector. All its needs are catered for by a section of the ROM program, which passes out data in the simplest possible form. However, the printer cannot be ignored completely, as certain input/output addresses are dedicated to it, and these may be called if LPRINT or LLIST are invoked by mistake. An output to port FB enables or disables the printer, and also conveys data, while an input on the same port provides an indication that the printer is (or is not) ready to respond. Note that FB in binary is 11111011. The low bit is the significant fact, when coupled with the high state of the other bits. A number of other input/output addresses with bit 2 low will serve in place of FB, and all such addresses should be avoided for other purposes.

The Memory Map

The processor generates sixteen address bits, which appear on lines A0-A15. This allows 65536 addresses to be defined, in the manner illustrated in Fig. 2.

Normally, bit 15 determines whether the upper or lower half of the address range is to be used, and bit 14 determines which quarter within that half is to be used, and so on. The ZX81 works in a slightly different way.

In the basic machine, A15 is ignored, so the contents of the lower half of the address range can be found repeated as a 'ghost' in the upper half of the range. Add 32768 to an address in the lower half of the range, and the contents of the address so formed will be same as the contents of the lower address.

A14 is used to select ROM when it is zero, and RAM when it is 1. In the basic machine, A13 is ignored, so there are again ghost copies within the respective quarters of the memory range.

Internal 1K RAM uses bits A0-A9, A10 being added with 2K memory. External 16K memory uses A0-A13. Each added bit

suppresses part of the ghosting, and some add-on memories suppress the ghosting entirely.

The 8K ROM uses A0-A12, and in the basic machine produces a ghost at 2000-3FFF.

Those who want to explore the ghosts will find the DUMP programs in Appendix A useful. The hexadecimal form may call for more thought by those not familiar with that form of notation, but it will be found to give a clearer picture of the memory structure.

The ghosts must be eliminated if the memory area they occupy is to be put to other use, and this is done by pulling $\overline{\text{ROMCS}}$' or $\overline{\text{RAMCS}}$' high, according to whether the ghost originates from ROM or RAM. The elimination must be selective, on the basis of the address pattern for the area to be cleared. For example, the upper half of memory could be cleared by pulling both chip-select lines ($\overline{\text{ROMCS}}$' and $\overline{\text{RAMCS}}$') high when A15 = 1, but they must not be forced low when A15 = 0. Even in this relatively simple case it would be wise to make a check before firm installation, as it appears that suppression of the ghost might affect the operation of the display routine.

The input/output address map is more difficult to fathom. Most Z80 systems use only bit A0-A7 to define I/0 addresses, but the processor does generate nominal sixteen bit addresses for IN and OUT instructions:

IN A, (N): The contents of register A define the upper eight bits, the value of N defines the lower eight bits.

OUT (N), A: The contents of register A define both the data and the upper eight address bits, the value of N determines the lower eight bits.

IN, r (C): The contents of register BC define the address.

OUT (C), r: The contents of register BC define the address.

The addresses used by the basic ZX81 system have lower bytes FB, FD, FE and FF. However, the decoding is simplified

to a point where any even-numbered address will read the keyboard, and if the four addresses are written out in binary it becomes clear that only the low bit is being checked:

FB: 11111011
FD: 11111101
FE: 11111110
FF: 11111111

It is therefore necessary to limit addresses used for external equipment to those with the three least significant bits high, and bit 7 should preferably be low, to avoid conflict with FF. The concise rule is that addresses may be used if they are one less than a multiple of eight in the range 0-127. This allows sixteen addresses. Some expansion might be possible by utilising the upper eight bits, but these are also used by the ZX81, and caution is necessary.

It would be fair to say that the addresing systems of the ZX81 have been whittled down to a minimum, and that this can produce problems when the system is extended. In the case of memory addresses, the two chip select lines allow clearance of ghosts, but there is no comparable facility for input/output. One user did experiment with the insertion of a resistor in the $\overline{IORQ}$ line to the ULA, hoping to use it to allow the input to the ULA to be pulled high, making internal input/output inactive, but the scheme failed on timing grounds, and also because the display system got itself into a terrible tangle, with interesting but unreadable results.

There is one important way out. The ZX81 provides no IN or OUT functions in its BASIC, which means that special machine code routines have to be provided to drive input/output functions. On the other hand PEEK and POKE provide convenient access to all memory locations, so it would be much more satisfactory to be able to use memory addresses as channels for input/output data.

Nothing could be simpler. Where a normal input/output circuit uses $\overline{IORQ}$ as a qualifier, $\overline{MREQ}$ is used instead. Any ghosts in the memory area used must be suppressed, but the

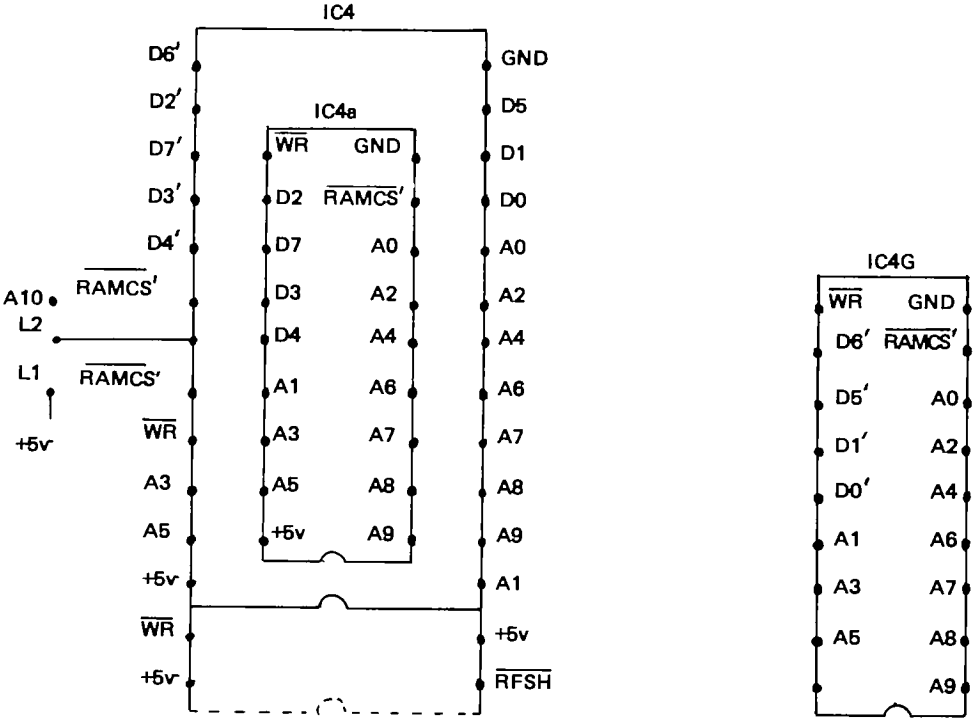


FIG 3 – ALTERNATIVE RAM LAYOUTS

limitation in available addresses is overcome, and no machine code need be provided. It may be a coward's way out, but there are times when bravery is wasted . . .

Internal RAM

The provision of a thousand or two thousand bytes of storage sounds ample for simple programs until the way the bytes are used is examined in detail. The Display File alone can absorb up to 792 bytes, with a full screen, and this is usually the first area to suffer when memory is used up. The system variables take up another 125 bytes, and room still has be to found for the program, the input buffer and the stacks. As programs become more ambitious, the need for extra store rapidly becomes evident, especially if only 1K is available.

The advice from Sinclair, not unnaturally, is to buy a RAM pack, but that is not the only option.

The RAM layout is shown in Fig. 3. Provision is made for two 1K X 4 memories, using 18-pin 2114 chips, or one 24-pin chip can be used, giving either 1K X 8 or 2K X 8. For the 2K memory the small link beside the 24-pin RAM position must be moved from L1 to L2 in order to bring address line A10 into play.

This allows uprating of a ZX81 1K memory to 2K, as in the TS1000, but there is scope for further development. As Fig. 3 shows, the connection pads for a 28-pin chip have been designed into the layout, and this is almost connected in the way required by the 4K × 8 static chips which are beginning to surface. Fig. 4 shows how the pinouts of the 24 and 28 pin RAMS compare. Some of the connections are already provided, and the rest would not be too difficult to make. It would not be impossible to accomodate an 8K × 8 RAM, but the '64K' RAMs are still a mite expensive.

The big bugbear in any internal RAM change is the need to remove existing RAM from the circuit board, unless a chip socket is provided. The correct approach is to clear the solder from each pin in turn, using a 'solder sucker' after just melting the solder on a pin with a brief touch of the iron. When most of

$\overline{CE}$ = CHIP ENABLE; $\overline{OE}$ = OUTPUT ENABLE; $\overline{WE}$ = WRITE ENABLE (ALL ACTIVE LOW)

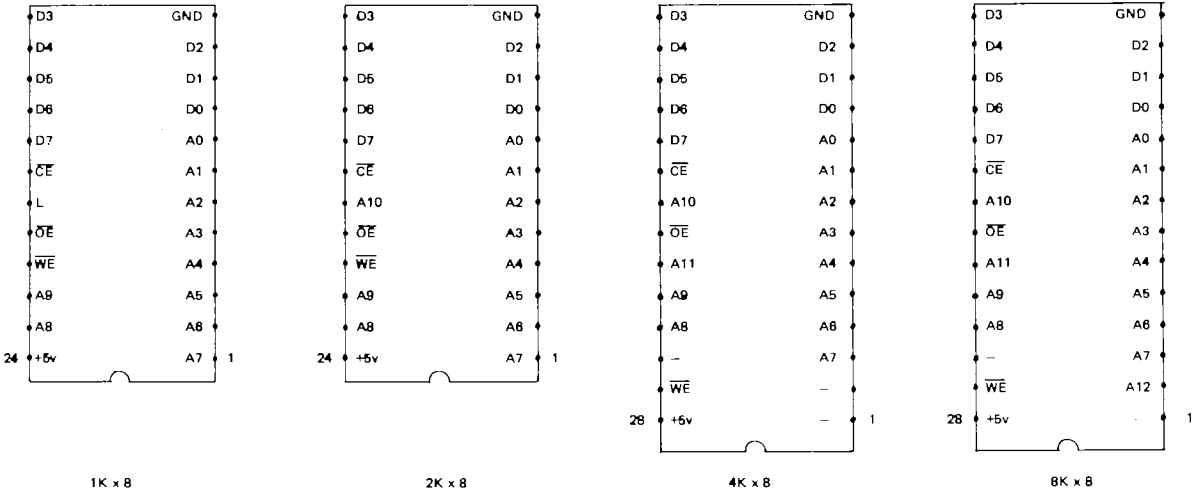


FIG 4 – PINOUTS FOR COMPATIBLE RAM RANGE

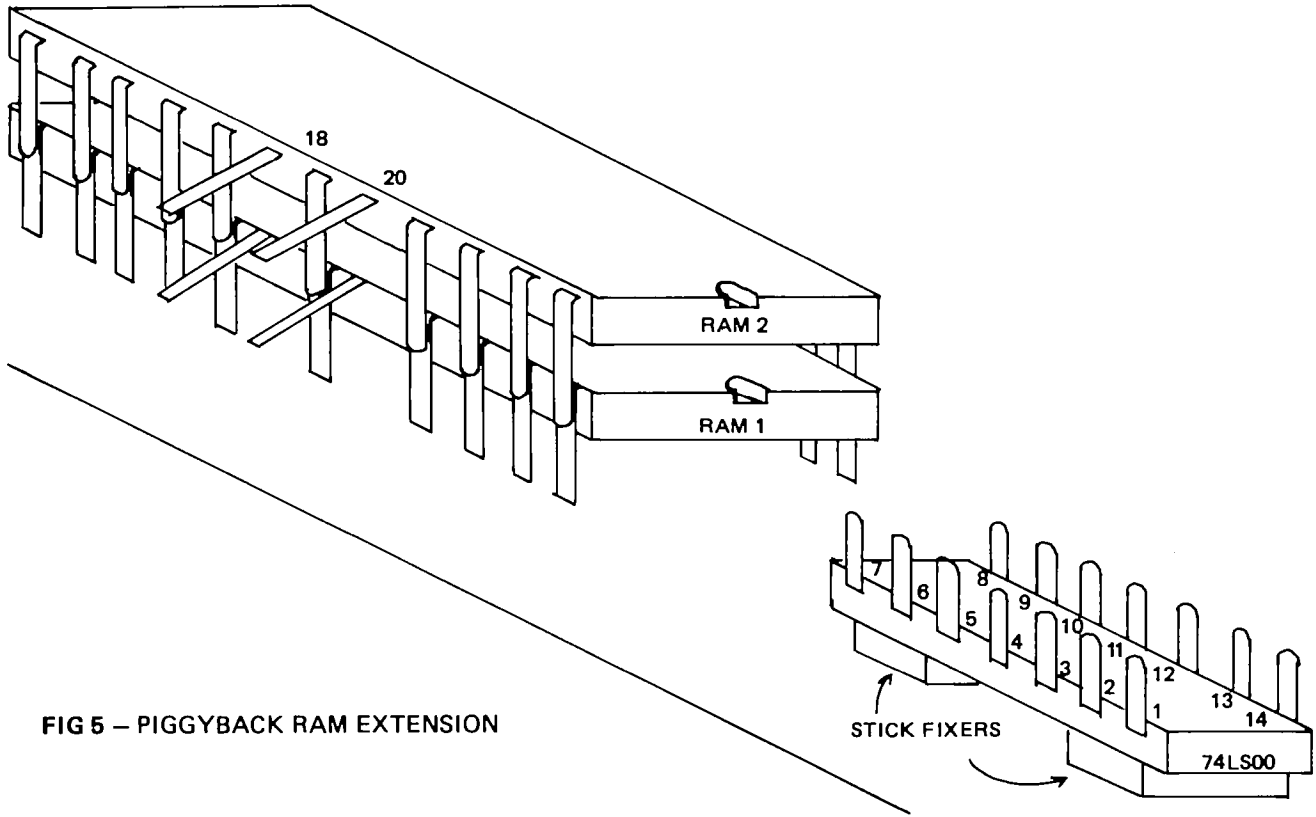


FIG 5 – PIGGYBACK RAM EXTENSION

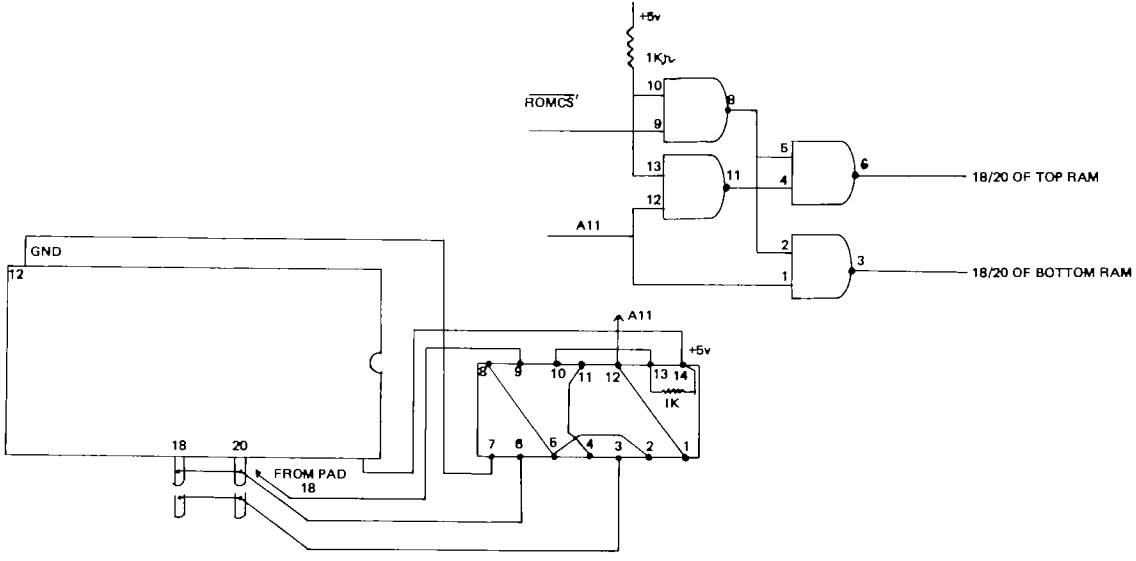


FIG 6 – CONNECTIONS FOR PIGGYBACK RAM EXTENSION

the pins are free, the last few will probably yield to a bit of gentle leverage combined with short dabs of the iron, but all the surplus solder needs to be cleared away to allow a replacement chip to be inserted.

The coward's approach is to snip the pins of the original chip with a pair of very fine-nosed cutters, so that the pins can be removed independently, but that means the chip has to be thrown away. Solder clearing is still essential.

When a change has been made, the check PRINT PEEK 16388 + 256* PEEK 16389 should give a RAMTOP value expressing the full size of the enlarged memory. The figures are:

1K memory: RAMTOP = 17408
2K memory: RAMTOP = 18432
4K memory: RAMTOP = 20480
8K memory: RAMTOP = 24576
16K memory: RAMTOP = 32768

If the expected figure fails to appear, a careful check for shorts between pins may discover the reason, that being the commonest cause.

Those who have no objection to the somewhat dubious practice of soldering one IC to the pins of another may take a different line towards memory expansion, though they will need to keep a wary eye on available headroom.

Fig. 5 shows the arrangement for a pair of 2K × 8 RAMs (e.g. Hitachi 6116). Pins 18 and 20 of both RAMs must be bent out sideways so that they can be connected separately, and a lead must be brought away from the pad to which one of the lower pins would otherwise be linked. Selection logic must then be provided, as shown in Fig. 6, the chip being a 74LS00 quad NAND gate mounted upside down on the circuit board by 'sticky fixers', the little pads with adhesive on both sides. Remember that the pins will be seen in reverse, making the numbering look unusual.

When the original chip enable, taken from the unused pad for pin 18 or 20 of the RAM, is low (active), one or other of the

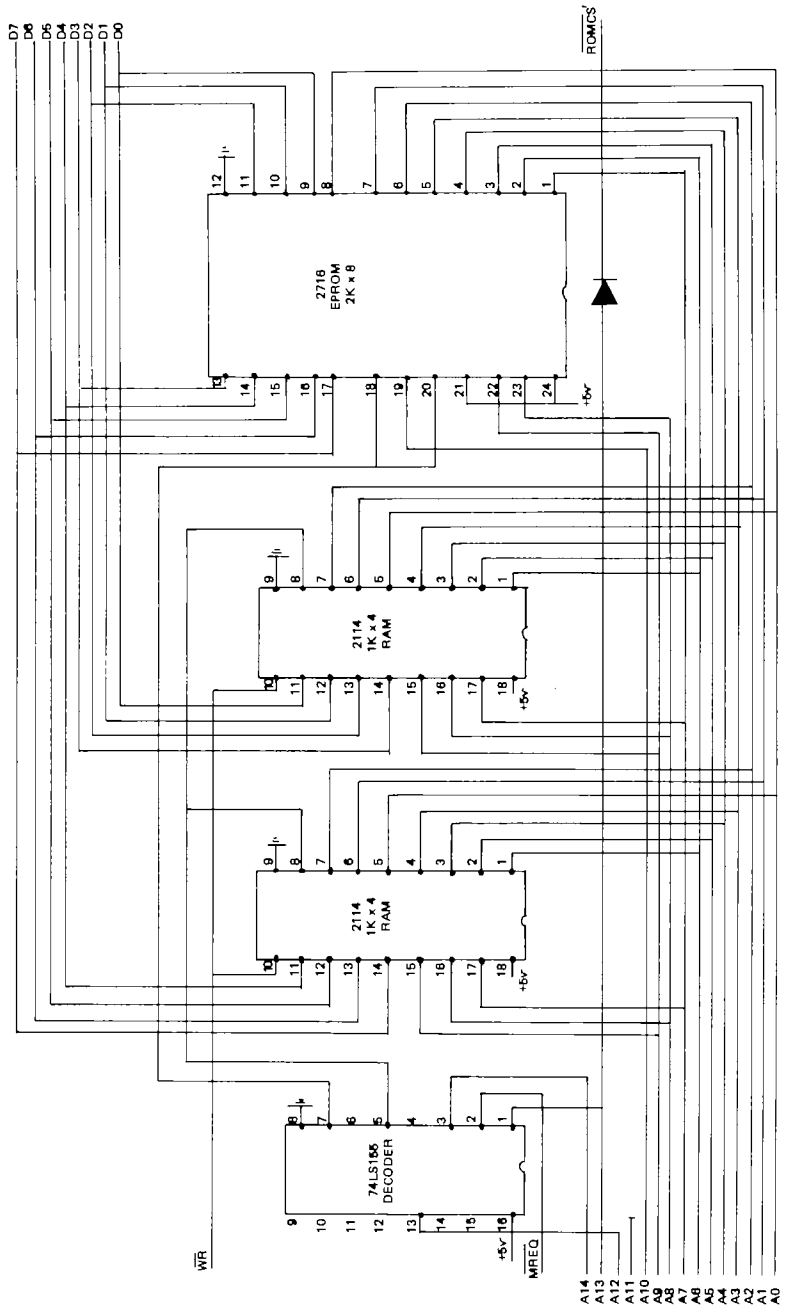


FIG 7 – ROM/RAM ADDITION

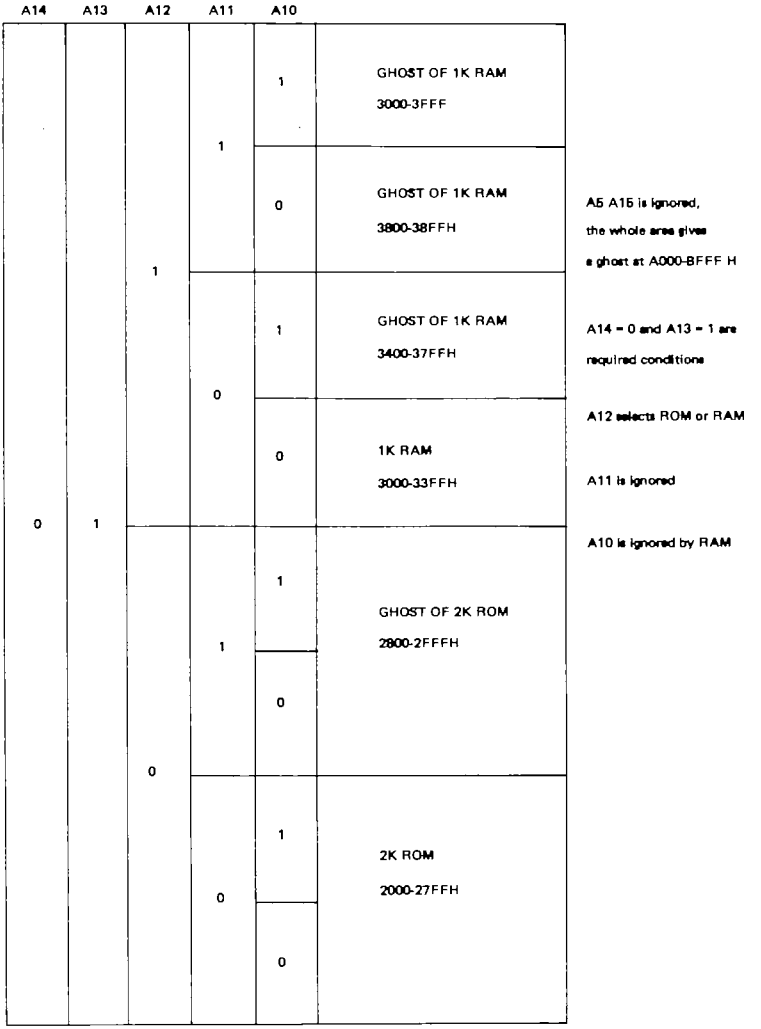


FIG 8 – MEMORY MAP OF ROM/RAM ADDITION

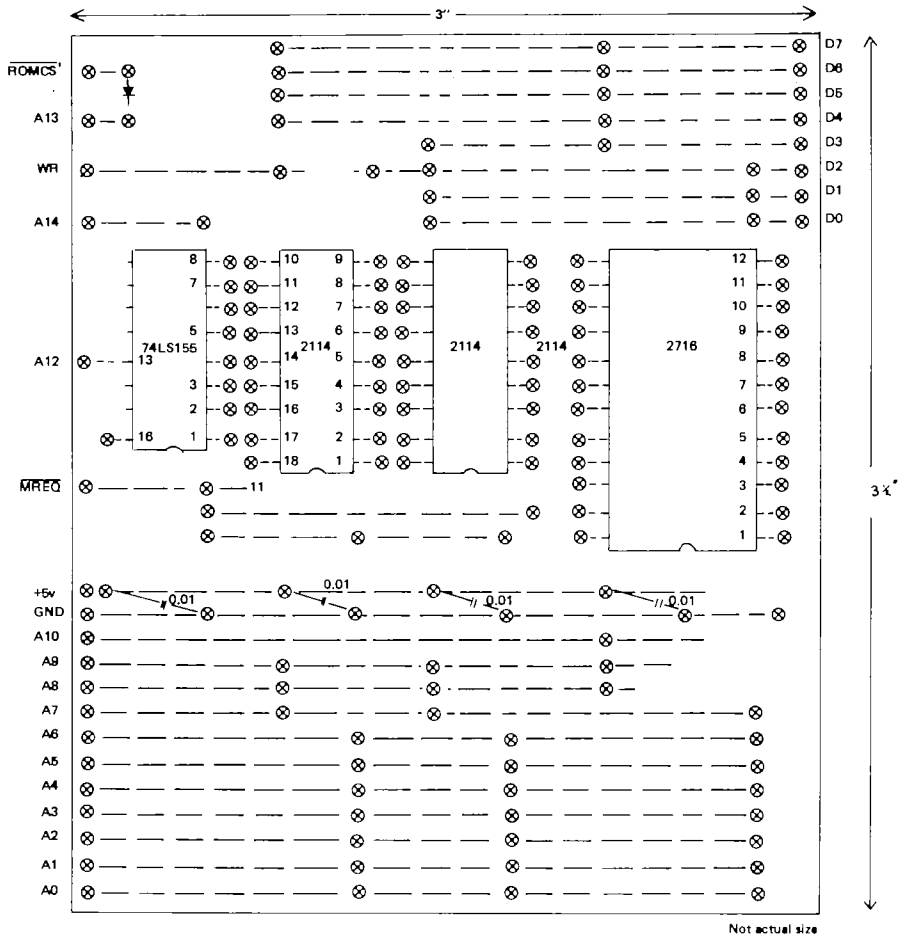


FIG 9 – POSSIBLE LAYOUT FOR ROM/RAM ADDITION, USING 0.1" VEROBOARD

(⊗ indicates a wrong pin)

outputs of the NAND gate complex will be low, according to the state of A11, and one of the two RAMs will be enabled. The same scheme can be used with 1K RAMs, in which case A10 should replace A11.

Having considered possible ways of extending RAM, it is reasonable to look at possible ways of extending ROM, too. Many useful snippets of machine code have been proposed in the journals, and it is much better to have them available in ROM than to have to load them at the start of a session or after a power break.

The obvious choice for the ROM is a 2716 EPROM (Texas use 2516 for the equivalent.) This provides storage of 2K bytes of fixed data. It has to be programmed, but some suppliers will set up customer's programs for a reasonable charge, or a friendly owner of a programming device might help.

It is convenient to have some RAM associated with the ROM, and this should be set in a memory area where it will not be open to corruption by the programs in the main ROM. A pair of 2114 RAMs, each giving 4 × 1K storage, will serve very well, and if both ROM and RAM are placed in the 2000-3FFF region they will be out of harms way.

The circuit is shown in Fib. 7, with the memory map in Fig. 8. The diode connecting A13 to $\overline{\text{ROMCS}}$' suppresses the ghost of the main ROM when A13 is high, and the ROM/RAM combination is disabled when A13 is low, avoiding any clash between the two store sections. The 74LS155 decoder sets up the memory map, using A12, A13 and A14. As A11 is ignored, and A10 is used only with the EPROM, there will be ghosting, and additional decoding is necessary if the ghosts need to be cleared to make way for other store or memory-mapped addresses.

Construction is a matter for personal taste, but Fig. 9 shows a possible arrangement. This would be a tight fit inside the case, but a smaller layout would make the wiring more difficult. Note that the conductive strips on the Veroboard need to be cut where a given strip is used for more than one signal. If the

assembly is mounted inside the ZX81 case, make sure that no shorts can occur by placing a sheet of insulating material between the two circuit boards.

If the problem of programming the EPROM can be solved, this assembly can be used to create a completely independent operating system, removing some of the limitations of the BASIC interpreter program, especially where machine code is concerned. There are so many possible routines that could be included that it would be a pity to describe only a small sample, but Appendix B gives an indication of what can be done, while some of the input/output routines which are described in the section on Externals could be useful.

Access to the EPROM programs is made by the USR function, as described in the manual.

The Display System

The method of producing the display is very complex, involving both hardware and software, and uses some obscure or unadvertised characteristics of the Z80 processor. Only a superficial description can be given here, but it should suffice as a useful guide for those who wish to explore more deeply.

The display phase is initiated by a pulse generated by the ULA which is passed to the processor as a non-maskable interrupt (NMI). This causes the processor to jump to location 0066H, where there is an interrupt servicing routine. This increments A', the alternative A register, and if the result is not zero the interrupted program is resumed. If A' = 0, however, all the main registers are pushed on to the stack to preserve the data being used by the interrupted routine, HL is set to the Display File address held in 400CH, and then bit 7 of HL is set. A HALT instruction follows.

The processor now begins to execute two actions alternately: One action performs a NOP (no operation) instruction, which puts the contents of the program counter on to the address lines. The other action is a Refresh cycle, which puts out an address with the lower eight bits determined by the contents

of the Refresh register, and the upper eight bits determined by the contents of the I (interrupt vector) register, which are initialised to 1EH.

It should be noted, however, that the circuit using transistor TR1 holds the processor in a WAIT state until NMI becomes false. The length of NMI therefore determines a period during which display action is held in abeyance.

When the output of a line begins, there is no time to hang about. The processes involved are much too rapid to be handled by the processor, which continues to output the two addresses described above, one after the other. The program counter address is set to point to the display file by performing a nominal jump to the address contained in HL, and the ULA uses this address to read character codes from the display file. As soon as a code has been read, it is used to form an address based on the processor output during refresh, the character code, and the output of a three bit counter in the ULA. The address is:

I*256 + CODE*8 + COUNT

where I is the number in the I register (1EH), CODE is the code read from the display file, and COUNT is the output of the three-bit counter. The address points to a pattern byte stored in the top of ROM at 1EOO-1FFF. The byte is read by the ULA and output serially to the video modulator at a 6.5 MHz clock rate.

COUNT is incremented by each television line sync pulse, and determines which of the eight bytes forming the character is to be used. Each line of characters in the display file is read eight times before the system moves on to scan the next line.

The following BASIC program illustrates this action, and may help to make it comprehensible:

```
10 INPUT CODE
20 CLS
30 FOR C = 0 TO 7
40 LET PATTERN = PEEK (7680 + CODE*8 + C)
```

```
 50 FOR B = 1 TO 8
 60 LET A = INT (PATTERN/128)
 70 LET PATTERN = 2* (PATTERN - 128*A)
 80 LET Z$ = CHR$ 8
 90 IF A = 1 THEN LET Z$ = CHR$ 128
100 PRINT Z$;
110 NEXT B
120 PRINT
130 NEXT C
140 GOTO 10
```

A character code (0-63) is input. C represents the binary counter, and the PEEK address is that defined above for pattern access in the real system. The bits of the pattern are represented on the screen.

The BASIC performs the progress slowly, but the actual output of data handles a new byte every 1.23 uS, filling out one pattern row of a 32-character line in 40 uS, the time which the television scan takes to traverse the display area. It all fits in beautifully, but working it out must have cost a few headaches . . .

When 256 bytes have been output, making up a complete line of characters, it is necessary to move on to the next line of characters in the display file, and this is handled by the main interrupt routine at 0038H.

For the present purpose, however, the point of most importance is the effect which this system has on extensions of the basic ZX81. One slightly unexpected point is that if more than 16K of RAM is provided it is important to make sure that the display file does not lie across the boundary set at address 8000H. If it does, the system will fall apart, because the significance of setting bit 15 in the display file address will be different for the two parts of the file, and the pointer to the file will get lost.

In FAST mode, there is no need to worry about display system for most of the time. NMI is suppressed, so that display action does not occur.

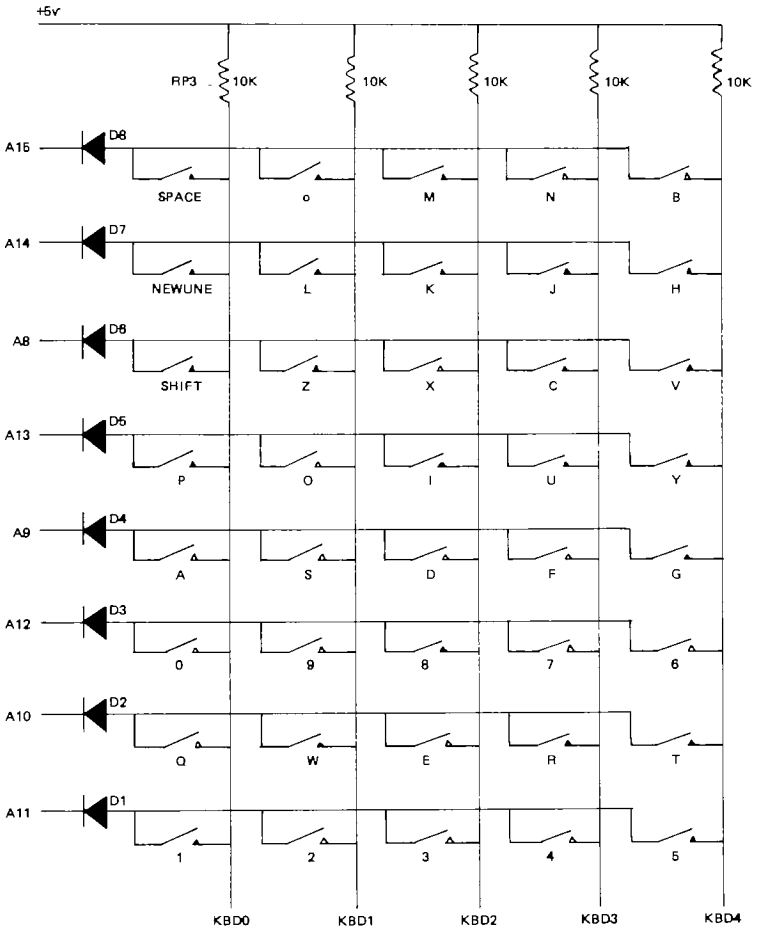


FIG 10– KEYBOARD CIRCUIT

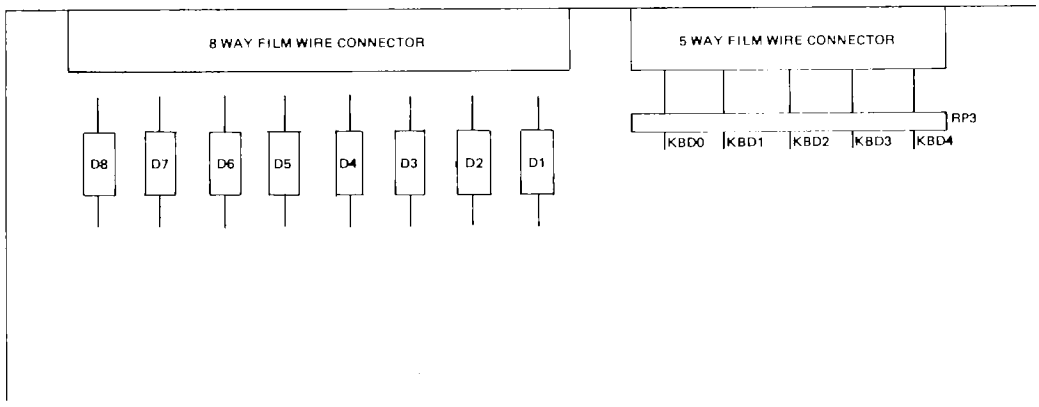


FIG 11 – LAYOUT OF KEYBOARD CONNECTIONS

In SLOW mode, the main point to watch is that any of the pointers used in the display system are left well alone. Attempts to poke into the display file can cause a crash if the character displaced is a NEWLINE, but poking is possible if a check for NEWLINE is made first.

On the whole, it is best to skirt round the display system wherever possible. This is yet another case where discretion is the better part of valour.

The Keyboard

Whatever else may be said about it, the keyboard system of the ZX81 is an outstanding example of simple ingenuity. Forty pressure pad switches and eight diodes, plus connections, account for all the visible hardware.

As shown in Fig. 5, the switches are connected in a matrix of eight rows and five columns. The rows are connected through diodes to address lines A8-A15, and the columns are connected to input port FE, which is hidden in the 'dogsbody' chip.

To interrogate the keyboard, it is necessary to read from port FE while holding one of the top eight address lines low and the others high. That may sound as if a conjuring trick is needed, but instruction ED 78 makes it all easy. The instruction mnemonic is IN A, (C), and the action is to input data from a port defined by the contents of the C register, the data passing to the accumulator. The small print remarks in passing that the upper eight address lines happen to be determined by the contents of the B register.

The instruction sequence to check a key might be:

```
LD   BC, FEFEH
IN   A, (C)
BIT  O, A
```

The contents of B would set address line A8 low while holding A9-A15 high. Row 3 of the keyboard would therefore be pulled low, and if any switch in that row was pressed the corres-

ponding column output would be pulled low. The check on bit 0 would give a zero result if the Shift key, in column 1, is pressed.

The keyboard input routine is located at 02BB on, though this section only sets low bits in HL to indicate which key is pressed. Further processing, taking note of the current working mode, is needed before the actual meaning of the depression can be determined.

Providing a more orthodox keyboard is no great problem. A well-spaced layout is so large that the usual practice is to mount the circuit board inside the keyboard case, taking care that the power, TV and tape recorder connections and the external interface are conveniently accessible. The replacement keyboard can be wired by ribbon cable to the pins of the film wire sockets, since suitable replacement film wire would be more difficult to provide.

In passing, it may be remarked that the film wire may kink or otherwise suffer damage if it is pushed into the sockets carelessly. In such a case the remedy is to clip off the ends and so provide fresh material, but this cure will only work a limited number of times.

In addition to providing the forty replacement key switches, some further improvements can be made with a new keyboard. One of the most obvious is the provision of a proper Reset key.

The reset line of the ZX81 is pulled towards +5v by a resistor, and tied to ground through a large capacitor. At switch-on, the capacitor delays the rise in voltage, so that the processor is initially held in the reset state. As the voltage on the reset line rises, the processor is freed to work normally.

All that is needed to provide a manual reset is to connect a switch between the reset line and ground. When the switch closes, the line is grounded, and the reset state is entered. Open the switch, and the line voltage rises towards +5v, allowing normal action to be resumed. This can, of course, be

provided without changing the keyboard, the switch being fitted at a convenient position in the ZX81 case.

A slightly less obvious change is the provision of single keys for functions that normally need two. Double pole switches are needed, and these are connected in parallel with the normal switches, as follows:

Function select:	Shift and Newline
Rubout:	Shift and 0
Edit:	Shift and 1

Similar treatment may be convenient for the 'arrow' functions, Graphics, etc. This is a matter of taste.

Unfortunately, a general repeat function does not seem feasible, though it would be very useful. It would require a method of interrupting all the keyboard paths to simulate repeated key depressions. The permissible repeat rate varies widely, too, as the response to some key actions can take a relatively long time.

Finally, some other uses of the keyboard input port must be noted.

The keyboard column outputs appear on bits 0-4 of the port, corresponding to pins 33, 31, 29, 27 and 25 of the ULA. Bit 5 of the port is similarly related to ULA pin 22, which is tied to ground in the American version. When the keyboard routine has checked the keyboard inputs, it goes on to check bit 5, and if the bit is high (MARGIN) is set to 37H (55), the value for a 50Hz television frame scan. If the bit is low, (MARGIN) is set to 1FH (31), the value for a 60Hz frame scan.

Bit 6 of the port signals, by a high state, that data from tape is available, one bit at a time, and Bit 7 provides the data.

THE EXTERNAL INTERFACE

The connector which forms part of the rear edge of the ZX81 circuit board carries most of the signal lines which are likely to be needed for communication with external devices. However, the utility of some of the signals is restricted by the way in which they are used by the internal system, so the connection of external equipment must be approached with a measure of caution. A detailed examination of the signals here will lay the ground for the descriptions of specific add-on facilities.

Address Lines: A0-A15

Sixteen active high outputs.

The address lines are driven directly by the processor, and should not be too heavily loaded. If more than two TTL inputs have to be driven, the lines should be buffered, but the buffering must be simple, or there may be unacceptable signal delays.

The lines serve both memory and input/output, the latter normally using only lines A0-A7, though the state of lines A8-A15 can be determined by certain input/output instructions, as described in relation to the ZX81 keyboard system.

In theory, all the address lines ought to be taken into account, but the discussion of the ZX81 memory map showed that this was not entirely essential. What is essential is that there should be no overlap between the memory or input/output address areas used by different devices. This can be assured by adequate decoding, and the disablement of internal ZX81 memory where necessary, using $\overline{\text{RAMCS}}$' and $\overline{\text{ROMCS}}$' (see p45).

The external lines are coupled to the ROM address inputs via 1K ohm resistors. This allows the ULA to take over control during screen refresh.

Data Lines: DO'-D7'

Eight active high lines, bidirectional

The main data lines are not brought to the external connector, which receives the secondary data lines, coupled to the main lines via 470 ohm resistors. The main lines connect only to the processor and ULA, the subsidiary lines serving the memory.

Use of the data lines is controlled by strict rules. When data is being output by the ZX81, no external device may drive the lines. When data is being input by the ZX81, only one external device may drive the lines. (Though this second rule can be broken if the implications are properly understood.)

The usual way of satisfying the rules is to use devices with 'tristate' outputs, which disconnect themselves from the output lines except when they are enabled. The enable signals are obtained by decoding address and control line signals.

Loading on the data lines should be kept light. The use of buffering is complicated by the bidirectional use of the lines, but is not impossible.

Memory Request: $\overline{\text{MREQ}}$

One active low line, output.

The processor makes this line low when it wishes to communicate with memory, whether reading or writing.

Input/Output Request: $\overline{\text{IORQ}}$

One active low line, output.

The processor makes this line low when it wishes to communicate with an input/output channel.

Read: $\overline{\text{RD}}$

One active low line, output.

The processor makes this line low when it wishes to receive data.

Write: $\overline{WR}$

One active low line, output.

The processor makes this line low when it wishes to transmit data.

ROM Chip Select: $\overline{ROMCS}$'

One active low line.

This line is connected to the chip enable pin of the ROM, and is driven by the ULA through a 680 ohm resistor. If the line is pulled high by an external device, the ROM will be disabled, and both the normal output and the 'ghosts' will disappear. This must be done selectively, on a basis of the address in use. If the line were pulled high by a direct connection to +5v, the system would fail to work. At the least, the line must be allowed to go low for addresses in the 0000-1FFF range, i.e. when A13, A14 and A15 are all low, unless an alternative ROM program is supplied externally.

However, it is both permissible and useful to disable the internal ROM when A13 or A14 or A15 is in a high state, as this eliminates all the 'ghosts' of the ROM.

RAM Chip Select: $\overline{RAMCS}$'

One active low line.

This line works on the same principle as $\overline{ROMCS}$', but is connected to the RAM chip enable pins. If external RAM running up from address 4000 is provided, $\overline{RAMCS}$' may be tied to +5v or otherwise taken high. This disables the internal RAM completely. (If +5v is used to pull the line up, it should be applied through a 330 ohm resistor as a safety precaution. This also applies to $\overline{ROMCS}$'.)

If the internal RAM is to remain in use, with external RAM in other areas, the internal RAM should be enabled only when A14 is high and A10, A11, A12, A13, A15 are all low. If the 2K internal modification has been carried out, A10 must be deleted from the address lines used.

Reset: $\overline{\text{RESET}}$

One active low line.

If this line is taken low, the processor action stops, and when the line is released the processor will restart by reading an instruction from location 0000. When power is first applied, the Reset line is held low by capacitor C5, which charges up slowly through R15 until the processor is enabled to start. Shorting the Reset line to ground discharges the capacitor, and releasing the short produces a normal restart from 0000.

The line may be used to reset external devices in the same circumstances.

In some Z80 systems, the start location is varied by creating artificial data as Reset is released, but this requires direct access to the processor data lines, which is not available at the external connector. Those who are interested in using an alternative ROM program which can be entered automatically may like to investigate the possibility of supplying an artificial absolute jump instruction (e.g. C3 XX XX) in response to the first three processor reads after reset is released . . .

Halt: $\overline{\text{HALT}}$

One active low output line.

If $\overline{\text{HALT}}$ is low, the processor is obeying a HALT instruction. The line goes high in response to an interrupt or the release of Reset.

This signal is used by the internal system, and may be of possible use in determining whether processing or display refresh is being executed.

Wait: $\overline{\text{WAIT}}$

One active low input line.

Making this line low puts the processor in an idle state. The line is normally used for gaining extra time for slow memory or input/output devices to respond, but in the ZX81 it has a special use, being controlled by TR1. It is therefore doubtful whether

external use is practicable, though some experiments might be worthwhile if response time proves a problem.

Bus Request: $\overline{\text{BUSRQ}}$

One active low input.

Bus Acknowledge: $\overline{\text{BUSAK}}$

One active low output.

If $\overline{\text{BUSRQ}}$ is made low, the processor is asked to yield control of the system. When the current instruction has been executed, the processor will make $\overline{\text{BUSAK}}$ low to signal acceptance, and will then allow the lines which it controls to 'float', so that they can be controlled by other means.

Making use of this state is a complex matter, and is made more so by the presence of the ULA. However, the signals may be of use in FAST mode, when the ULA is less active.

Clock: $\overline{\phi}$

One active low output.

The clock generator runs at 6.5 MHz, but a divider in the ULA generates a 3.25 MHz clock for the Z80, and this is available externally.

Refresh: $\overline{\text{RFSH}}$

One active low output.

The Refresh signal is intended for use with dynamic RAM, and the circuit diagram proposes its use for that purpose, but it is also affected by the ROM program, and may not be trustworthy in SLOW mode. It has been assumed that static memory is safer.

Non-Maskable Interrupt

One active low input.

The $\overline{\text{NMI}}$ line is controlled by the ULA, calling a jump to location 0066 when it is low. It is not available for external use.

Interrupt: $\overline{INT}$
One active low input.

The interrupt line is coupled to A6 inside the ZX81, and is not available for external use.

Machine Cycle 1: $\overline{M1}$
One active output.

The processor makes this signal low during the first cycle of an instruction fetch. It is also made low, with $\overline{IORQ}$, to indicate that an external interrupt vector is required. No external use can be envisaged.

Ground
Two connector pins are used, to ensure good contact.

+5 volts
This line is supplied from the output of the internal regulator, and may be used for loads of up to 50-100 mA. Higher loads may damage the regulator, and even 100 mA will add to the internal heat dissipation to an undesirable degree. This line must not be connected to +5v derived from an external source, as one or the other will tend to take the whole load.

+9 volts
This line comes from the external power pack. The off-load voltage is appreciably above the nominal value, falling sharply as load is applied. However, it can be fed to a regulator in external equipment to derive a stabilised 5v supply, providing the total current drawn by external equipment does not exceed 400 mA. For higher currents, a separate power source is necessary.

Using the Interface
It may appear that there was no sense in bringing out some of the interface lines, as they cannot be used by external equipment, but it should be noted that they have a value for the manufacturer, in that the board can be plugged into test gear

for a comprehensive check.

In terms of user requirements, there is a great deal that can be done with the signals provided, as will be shown on the following pages.

EXTERNALS

Before getting down to the creation of add-ons, it is as well to consider how far you may want to go. A random addition of one section after another may lead to chaos, so some advance planning is useful, taking the limiting factors as a starting point.

The most important limitation may well be the power supply. A small system drawing 50-100 mA may rely on the internal regulator, but heavier demands, up to about 400 mA, will require a separate regulator driven from the +9v supply. If more than 500 mA will be drawn, then a separate power source is necessary, such as that shown in Fig. 12. More than one of these sources may be used in conjunction, but the various 5v lines must not be linked, or something is likely to cook.

The next limitation to consider is the available range of addresses. Relatively few input/output addresses are available for external use, but memory mapping may provide an alternative.

An extremely elementary but important limitation is that the external connections of the computer need to be made available to all the add-on devices. Some professionally made devices use the system illustrated in Fig. 13, which allows one device to be plugged into another. This is quite a satisfactory scheme, providing the individual units are coupled by spacers and through-bolts, so that the connector is not under mechanical stress.

Another approach uses a motherboard with sockets to connect up the individual devices. This has the advantage that an extra power supply can be mounted on the motherboard, but it is likely to be more expensive, due to the cost of the connectors.

One problem with large systems that may not become apparent until too late is that the first connector — and the part

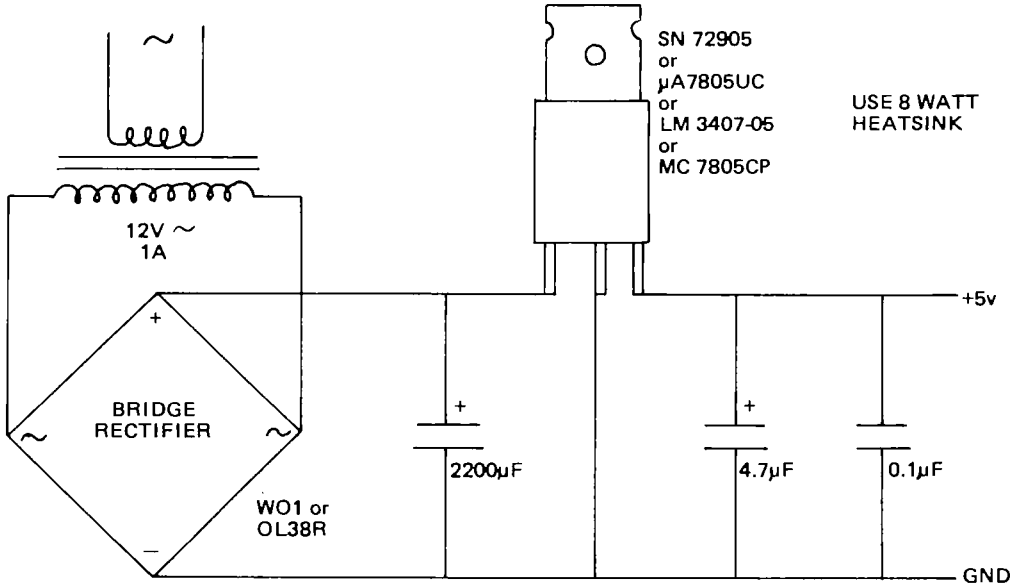


FIG 12 – ADDITIONAL POWER SUPPLY (1A AT 5v)

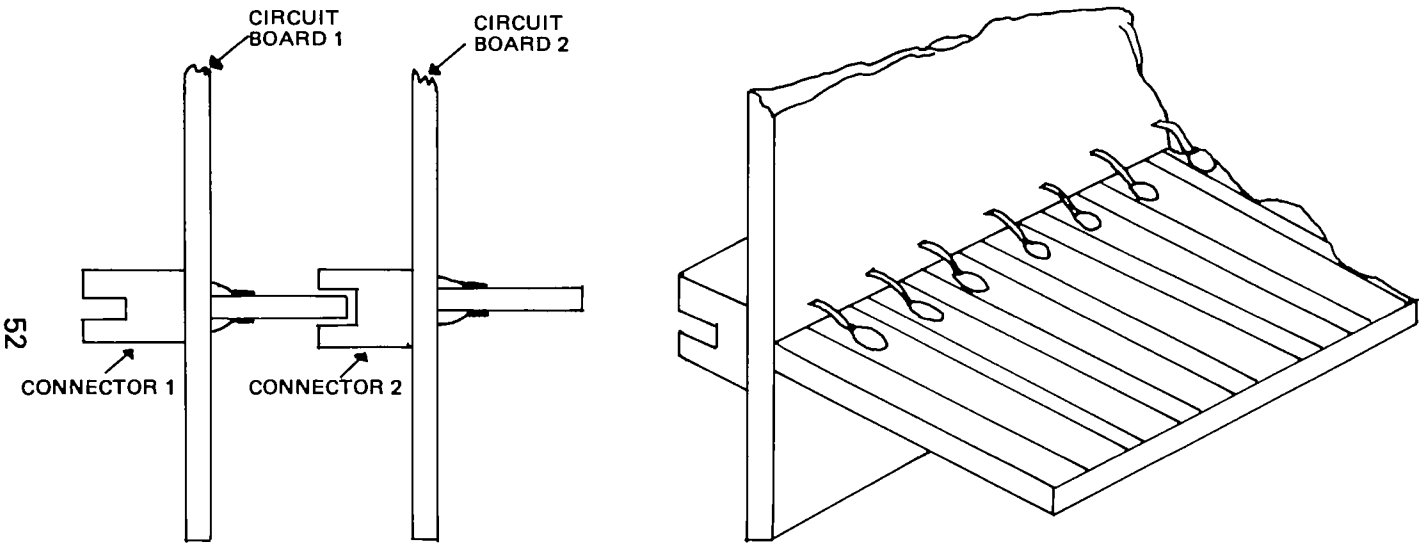


FIG 13 – NESTED CONNECTION SYSTEM FOR ADD-ONS

of the circuit board with which it mates — may be put under strain, or the actual computer may not be easy to position conveniently for use. The obvious answer is to insert a connection lead between the computer and the add-on complex, but this needs to be done properly. If a casual bunch of wires is used, there may be cross-talk between different lines, and that can be a serious problem.

One approach which has been found to work well is to use ribbon cable, earthing alternate conductors to form a sort of screen. Two such cables will probably be needed, one for the upper connections and the other for the lower.

More generally, it is only too easy to forget that computer circuits work at frequencies well above those which were regarded as the top radio bands a few years ago. This means that the layout of wiring is important. Excessive neatness obtained by lashing bunches of wires together is a sure road to failure.

A good recipe is to start with one of the standard Veroboard layouts, keeping the connections on one side of the board parallel as far as possible. Solder pins can be used to simplify wiring, though they are not essential.

Since the add-on systems which are to be described are basic types open to considerable variation, it will not be possible to describe the construction in full detail, but in most cases the circuit diagrams have been drawn in such a way that layout is suggested. What the diagrams cannot show effectively is the need to put decoupling capacitors, usually of 0.01 uF, across the power feeds to each integrated circuit. These should be near the component they serve.

Test add-ons as much as possible before linking them to the computer, using lash-up test rigs, and there will be a lot less difficulty in getting the overall system going.

External Store

The first add-on that occurs to mind is external memory, and the first memory component that you think of may be the 2114

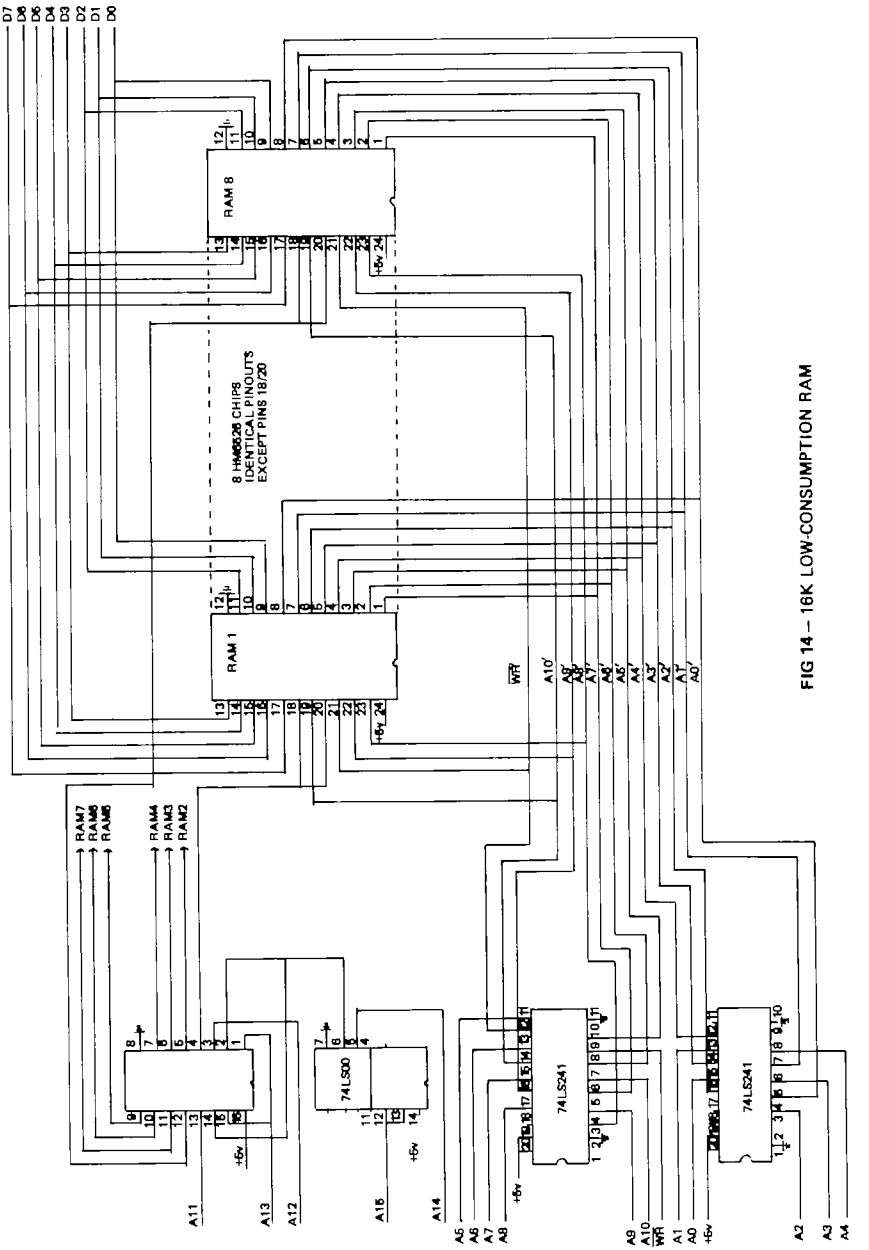
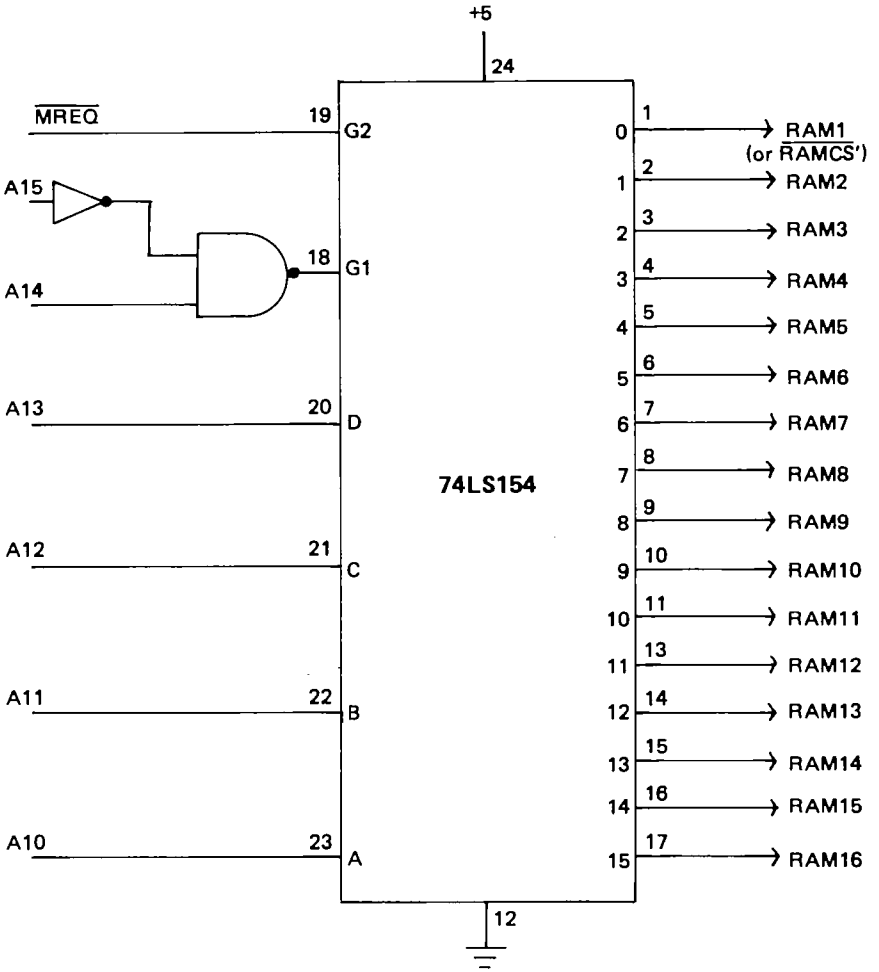


FIG 14 – 16K LOW-CONSUMPTION RAM



For any RAM to be enabled, G1 and G2 must be low, i.e. $\overline{MREQ}$ = 0, A14 = 1, A15 = 0. If the A15 input is omitted, ghosting will occur at COOO—FFFFH, which may not be important.

FIG 15 – ENABLE DECODER FOR SIXTEEN IK RAMS

1K × 4 RAM, mainly because it is cheap and readily available. Unfortunately, it is also rather greedy where power is concerned. Intel quote 50/70 mA, which would soon absorb available current. A 16K RAM extension, using the internal RAM chip to provide 1K, would involve 30 2114s, which would draw at least 1.5 A.

Some of the more recent static RAM chips are much more modest in their requirements. The HM-6515 1K × 8 CMOS RAM draws around 10 mA, and so does its 2K companion the HM-6516. These devices cost more, but are easier to use.

Fig. 14 shows a 16K RAM built up from HM-6516 components. Address lines AO-A10 go to all eight RAM chips, and so does the $\overline{WR}$ line, so these signals are buffered by 74LS241 octal buffers. The data lines also go to all the RAM chips, but the loads in this case will be light, so no buffering is needed.

With +5V and ground, that accounts for 22 of the 24 RAM pins. The remaining pins are chip enable and output enable, and these are driven selectively by a 74LS155 decoder. Two elements of a quad NAND gate chip, 74LS00, enable the decoder when A15 = 0 and A14 = 1, to put the RAM array in the 4000 - 7FFF address range.

Each RAM needs a decoupling capacitor of 10 nF connected to +5v and ground close to the chip.

This is one possible configuration of external RAM among many, but it should provide a useful guide to the form of other arrangements. If A15 and A14 are exchanged, the RAM will respond to addresses in the 8000-BFFF range, and if the inverter in the A15 line is removed the address range will be COOO-FFFF.

Consider the circuit for a moment: Using 2K × 8 RAMs allows simpler decoding. With 1K RAMS, a 74LS154 16-way decoder would have been needed. (Fig. 15) In this case, the internal 1K RAM could have been used, its select line being connected to one output of the 74LS154. With 2k RAMS in use, the internal RAM must be permanently disabled.

Setting up a good external memory is thus a matter of compromise and balance. A more expensive type of RAM chip may save cost elsewhere, and is likely to simplify the wiring. Cut down on RAM cost, and there will be a penalty elsewhere.

The use of dynamic RAM is not viewed with enthusiasm. According to the circuit diagram, RFSH can be used in its intended role, but there must be a question mark over that. Dynamic RAM also involves address switching to place AO-A6 and A7-A13 on the seven address pins in turn, and though this minimises the task of address decoding it is a complicated alternative. If you want to experiment in this area — Good luck!

The provision of external ROM is very little different from the provision of RAM. The $\overline{RD}$ signal replaces $\overline{WR}$, but the rest is essentially the same.

A point worth making, however, is that with RAM it is permissible to scramble the address lines or the data lines, since no one cares exactly which of the internal storage elements is used for a particular purpose. This is not permissible with ROM, which has to be set up on the assumption that the address lines are correct. This is a question of compatibility between the equipment which creates the ROM and the equipment which uses it.

It should also be remembered that if ROM is placed in the middle of a RAM area, the initialisation program will ignore the existence of the higher block of RAM, even if an attempt is made to bring it into action by modifying RAMTOP.

The best place for ROM is in the 2000-3FFF area, and since the presence of ROM usually implies the need for associated RAM, a small block of RAM may be placed in the same area.

A last word on memory: Don't be too eager to fill up all the available memory space at once. You may later find that there is a need to use it for some other purpose, and — to be frank — the idea of keying in programs long enough to use 48k of RAM does not appeal. The challenge of squeezing as much as

possible into 1K seems far more attractive . . .

A Simple Input Port

An unwise lecturer on a computer course once tried to dismiss input/output systems by saying casually that they were really just the same as memory systems. He later had cause to regret that remark.

In a sense, what he said contained an element of truth. From the processor point of view, the only difference between memory and input/output accesses is that for one $\overline{MREQ}$ is made low, while for the other the signal is $\overline{IORQ}$. An input port should come into action when it recognises its address, put out in combination with $\overline{IORQ} = 0$ and $\overline{RD} = 0$. It should respond by placing data on the appropriate lines, probably by enabling tristate outputs.

That said, there is room for variation. The port may be of various kinds. It may contain a clocked latch buffer, or a transparent buffer, or — less probably — it may pass on an instantaneous picture of what it is receiving on its input lines. The clocked latch enables data to be frozen at a moment determined by a timing system. The transparent latch can freeze data at the moment when the port is read, so avoiding change during the read process, which may cause trouble if the third option is used, the chance snapshot.

Now, suppose that data is presented to the port: How does the computer know when to read it? Once again, there can be a number of possible answers. The computer may take a reading at regular intervals, relying on its own internal timing system to decide the right moment. The computer may perform an output as a request for an input to be got ready. In almost any computer but the ZX81, the input channel could raise an interrupt to request the computer to read its data, but interrupt is barred in this case.

This leaves us with the 'polling' system, which relies on the 'handshake'.

Polling means that the computer makes a periodic check of its

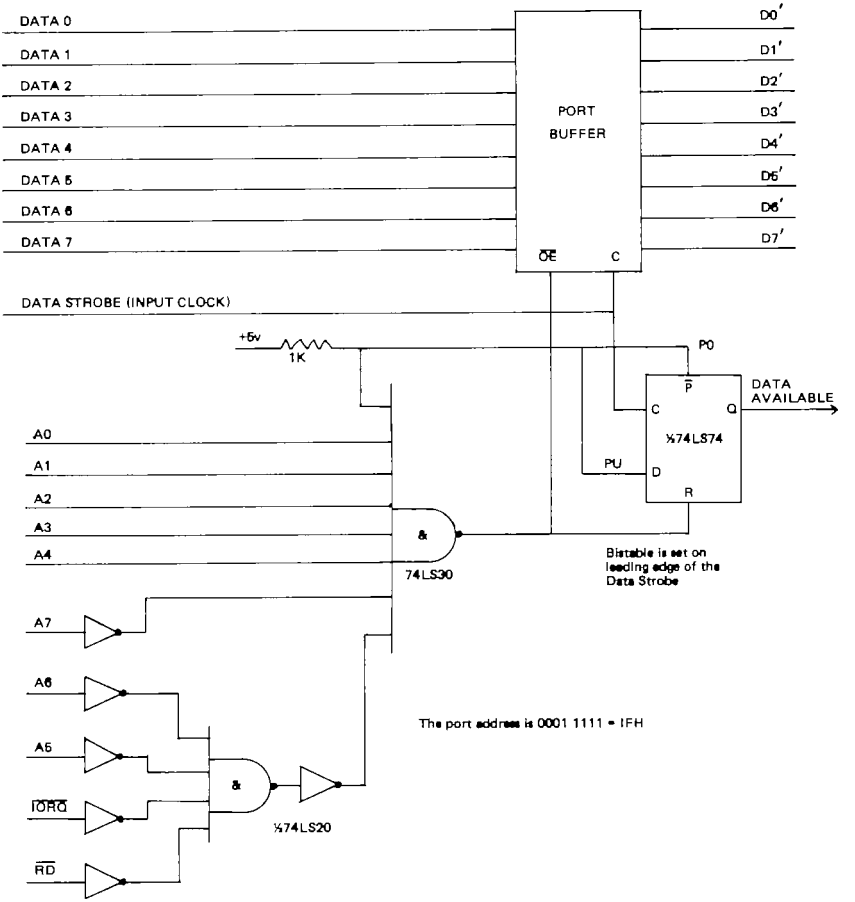


FIG 16 – INPUT PORT

input channels to see if any of them have data available. Instead of looking at the actual input ports, it will look at an extra input port which is driven by bistables, one bistable to each real input port. When data is sent to a port it is strobed into a latch, and the strobe also sets the associated bistable. This makes the extra port show a 1 in the bit position corresponding to the real input port which has been set. The computer checks the extra port now and then, and when a bit is set the data on the corresponding port is read in. This also resets the bistable.

To avoid losing data, the state of the bistable is signalled back to the data source, which is prevented from sending more data until the bistable is cleared. This is the 'handshake', an essential process in controlling computer data paths that handle only one byte at a time.

Although the system described could serve up to eight input ports, it is equally applicable with only one. The ultimate simplification is to limit the data to seven bits and use the eighth bit to show the state of the bistable. The data read is discarded if the eighth bit is not true. This method is convenient with 7-bit ASCII data, and is a basis for one form of the 'Centronics' interface.

The input port will thus, in general terms, take the form shown in Fig 11.6. This shows the actual port component as a plain block, which could be a latch, transparent latch, or simple buffer, but which must have a tristate output. The output goes to the data lines, and the output enable is conditional on the correct address, $\overline{IORQ} = 0$, $\overline{RD} = 0$. The extra port is assumed to be on an associated address, created by reversing one address bit, so that decoding arrangements can be kept simple.

This is a theme on which many tunes can be improvised. The main limitation is the shortage of permissible addresses, caused by the simple ZX81 input/output address decoding, but up to seven input ports and a 'handshake' port could be set up.

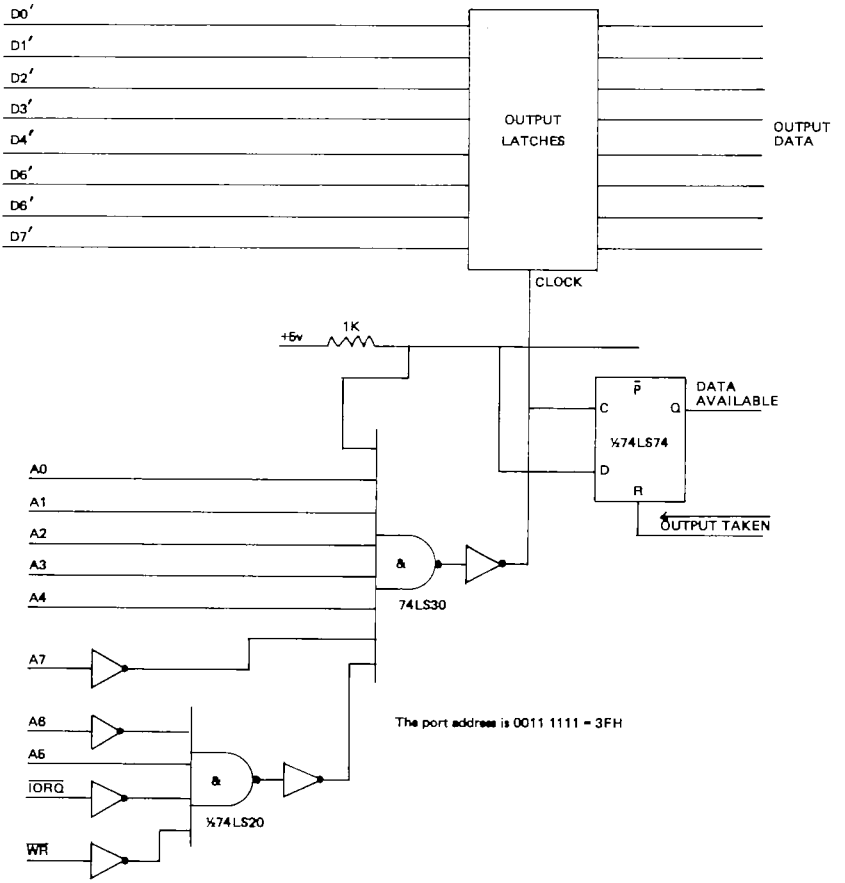


FIG 17 – OUTPUT PORT

As the ZX81 BASIC has no IN or OUT commands, these have to be generated by machine code, and simple routines will be offered for this purpose after output ports have been discussed.

It should be remembered, however, that input ports can be 'memory mapped', $\overline{MREQ}$ replacing $\overline{IORQ}$, and this allows PEEK to be used as a convenient means of access from BASIC.

A Simple Output Port

The system arrangement for an output port is very similar to that for an input port, but there are some fundamental differences.

Firstly, an output port must always latch its data, because the computer only makes the data available briefly. Secondly, the 'handshake' port remains an input port, but the control of the bistable is changed. It is now set when the computer performs an output action, and cleared when the external device reads the data. While the bistable is set, the computer is required to refrain from executing an output to that port, and the external device is required to take the data as soon as possible.

Fig. 17 shows the output port configuration, which should need no comment.

Once again, the lack of an OUT comand means that machine language is required to access the port, or it can be memory-mapped to allow access by POKE.

It should be appreciated that the power which can be supplied by an output port latch is very small, and if any heavy duty control work is required there must be some power amplification.

Another point that may not be obvious is that the setting of the port takes place so quickly that lines which are taken to the same state as before are barely disturbed, which allows individual bits to be changed independently.

Machine Code for Input/Output

The simplest way to set up machine code on a ZX81 is to hide it inside a REM statement. For this, an elementary loader in BASIC is convenient:

```
 1 REM XXXXXXXXXXXXXXXXXXXX
10 FOR X = 16514 TO 16534
20 SCROLL
30 PRINT X,
40 INPUT Y
50 PRINT Y
60 POKE X, Y
70 NEXT X
```

This allows for the insertion of 20 bytes of machine code. The number of bytes can be varied by altering the length of the REM statement and the second parameter in line 10. For n bytes of code, there should be n characters following REM, and the second parameter should be 16514 + n.

When the machine code has been entered, lines 10-70 may be deleted and the remainder of the working program may be added.

An alternative method of entering machine code is given in Appendix A. The method given here has the advantage that the code is loaded as part of the BASIC program.

For an input command, the code is simple:

```
16514   219 XXX  DB XX   IN      A, (XX)
16516   79       4F      LD      C,A
16517   6   0    06 00   LD      B,0
16519   201      C9      RET
```

The statement let Y = USR 16514 will set the input data in variable Y. XXX is the input port address. It is read into A, which is copied in C, B = 0, and then Y = BC.

An output command is slightly less straightforward.

```
16520   62  0     3E 00    LD      A,0
16522   211 XXX   D3 XX    OUT     (XX),A
16524   201       C9       RET
```

In this case, the output data is set by POKE 16521,n, which changes LD A,0 to LD A,n. USR 16520 will then perform the output.

The input address can be changed by a POKE into 16515, and the output address can be changed by a POKE into 16523, so these routines are quite flexible, though a little clumsy to use.

The only rule to remember in the use of a REM to hide machine code is that a HALT instruction (76H) will terminate the REM prematurely, as it will produce NEWLINE code, the BASIC line terminator.

Finally, a simple handshake routine may be useful, though it could be constructed out of the two routines already given, plus some BASIC logic.

```
16525   219 XXY   DB XY    In A,XY   Extra port
16527   40  252   28 FC    JR Z,-2
16529   219 XXX   DB XY    In A,XX   Input port
16531   6   0     06 00    LD B,0
16533   79        4F       LD C,A
16534   201       C9       RET
```

LET Z = USR 16525 will set Z from the input port data, but the routine will loop if port XY shows a zero state. The routine 'locks up' if port XY remains stubbornly at zero, but that is the nature of a handshake system. You have to arrange for the incoming data to warn the computer when the data stream is complete, by putting up a special code or sequence of codes.

Enough material has been given for some useful work on input and output ports, and it is time to turn to the more complex varieties.

Special Input/Output Ports

Most microprocessor manufacturers produce special

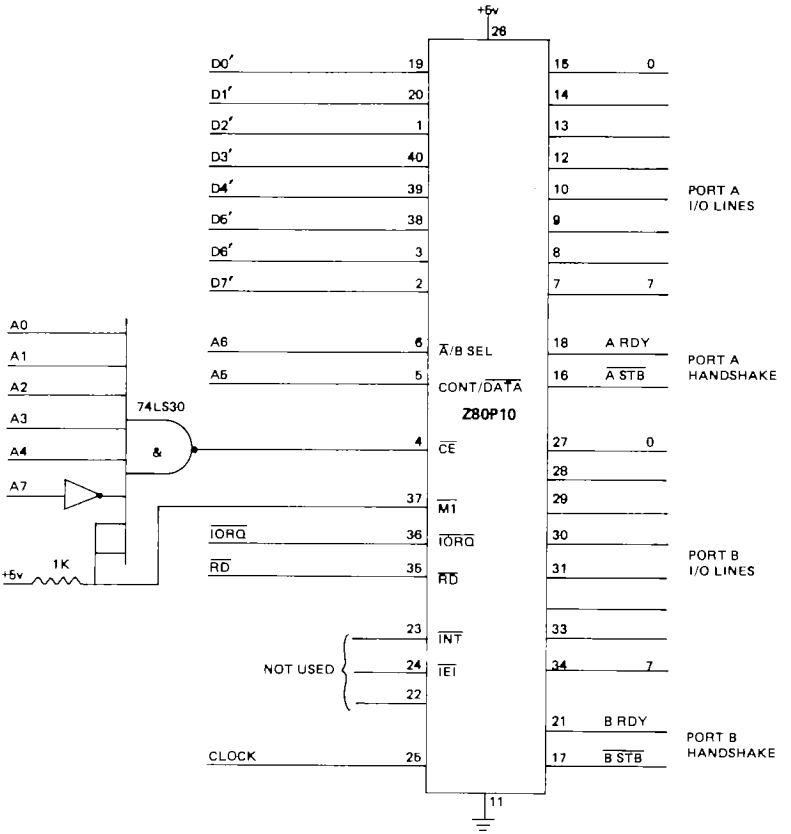


FIG 18 – Z80 P10

'support chips' for use with their processors, and Zilog are no exception. Among the devices they provide, the Z80 PIO is worth examination here. (Fig. 18).

The unit is a Parallel Input/Output device, providing two ports which can be used in a number of modes. With the address connections shown, the system uses four addresses;

1F Port A data
3F Port A command
5F Port B data
7F Port B command

If either port is given the command word 00XX1111, it is set in output mode. As data is set in the port by an output, its RDY line goes high, showing data available. The external device responds by making $\overline{\text{STB}}$ low.

If either port is given the command word 01XX1111, it is set in input mode. The RDY line is initially high, and goes low when $\overline{\text{STB}}$ strobes data into the port. RDY goes high when the data is input to the computer.

If port A, but not port B, is given the command word 10XX1111, it is set in bidirectional mode. Both sets of hand-shake lines are used, those proper to port A for output and those belonging to port B for input. Input and output data can co-exist within the PIO.

If either port is given the command word 11XX1111, a second control word is required. Where this contains a low bit, the corresponding bit of the port becomes an output: Where the control word contains a 1, the corresponding bit of the port becomes an input. An IN instruction will read all lines, but an OUT instruction only affects output lines.

There is a slight snag to all this. The normal use of the PIO depends on interrupt, which signals that a port is ready for output or has data to be input. The actual meaning of the interrupt is indicated by the supply of a particular interrupt vector. None of this can be used by the ZX81.

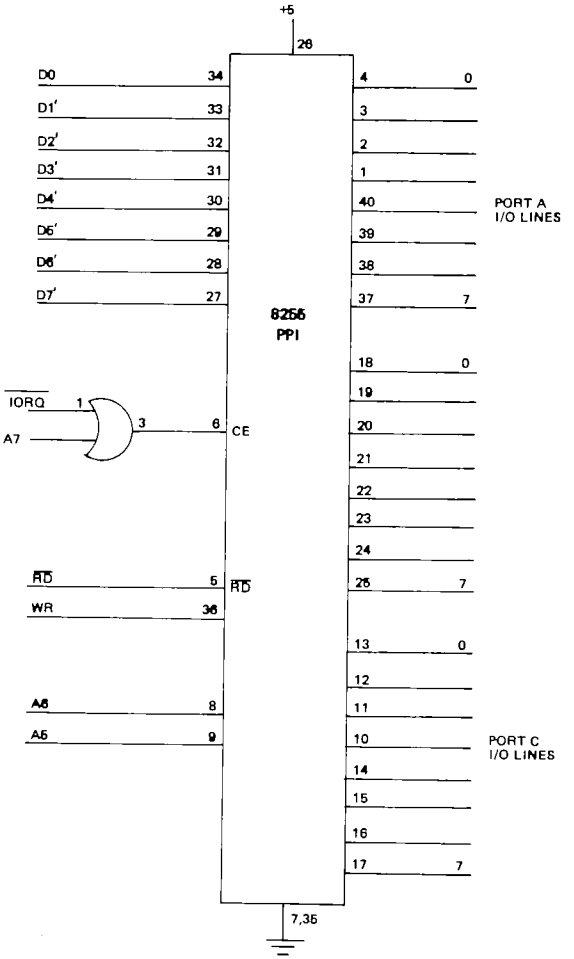


FIG 19 – 8255 PPI

Nevertheless, the ports can be useful in themselves, and the lack of interrupt facilities does not rule them out completely.

An alternative is the 8255 PPI. (Fig. 19) This is a very similar chip, but it provides three ports, the addresses used being:

1F Port A
3F Port B
5F Port C
7F Control.

The control word has the form 100XX0XX. Bit 0 controls the lower half of Port C, bit 1 controls Port B, bit 3 controls the upper half of Port C, and bit 4 controls Port A. In each case a 1 selects input and a 0 selects output.

A sensible arrangement would make Port A output, Port B input, and Port C half-and-half. Port C could then be used for sensing handshake lines and setting and clearing bistables.

The main purpose of this short visit to the more complex side of input/output techniques was to provide a warning: Almost every application in this area has slightly special requirements, and these multi-function chips may not quite match what is wanted. The most important lesson to learn about making up microcomputer equipment is that the hardware that is readily available must not be allowed to determine the design. The hardware must do the job that is needed, even if that makes design more difficult and shopping round for the bits even worse.

Driving a Standard Printer

The ZX81 uses its own special character code, and its printer driving routines are tailored to work with the special Sinclair printer, which has to be spoon-fed with data, one bit at a time, and it is therefore rather difficult to connect a standard ASCII printer, but the difficulty can be overcome. The method described here can be adapted and expanded in a number of ways, but it has its attractions.

Just to make the problem a little more difficult, it was decided

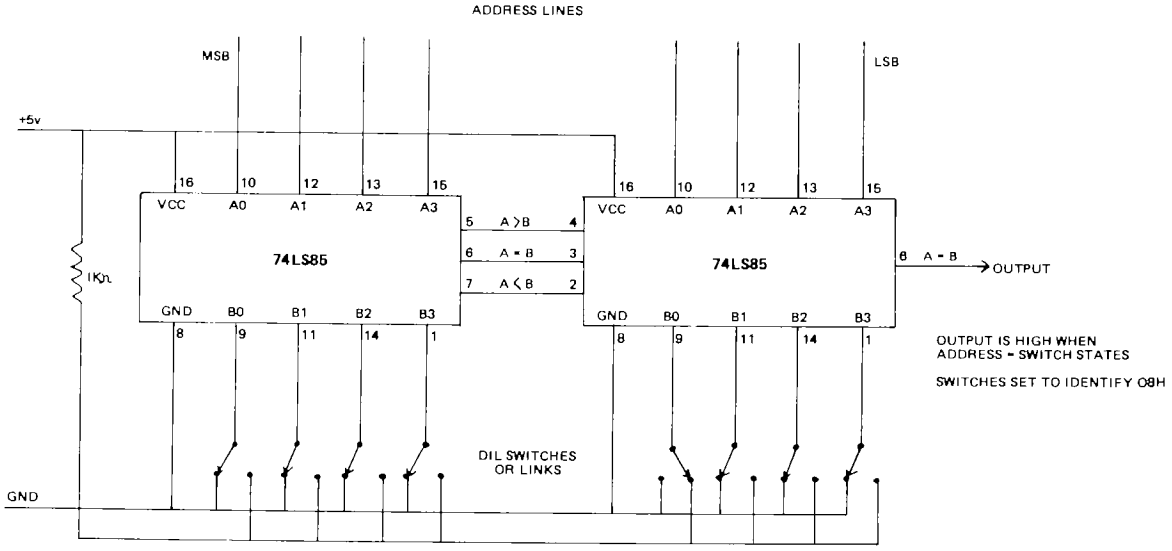


FIG 20 – COMPARATOR FOR IDENTIFYING ADDRESS BLOCK

that the LLIST, LPRINT, and COPY routines should still be effective, and that meant that the entries to the original printer routine at 0851 — 08F4 must be maintained. The entry points are;

0851: Entry to the routine loading the print buffer. The Sinclair printer needs the buffer, because there must be no hesitation in the steady output of data bits, but a standard printer is less particular, so this routine can output the data directly.

0869: This is the entry used by COPY. It prints 22 lines of text, using the Display File as a source.

0871: This entry sets up conditions for printing one line of text from the printer buffer.

There is one other point to note, and that is an entry from 0620 at 08E2, which means that the last part of the print routine, concerned with buffer clearance, must be retained. Putting the necessary routines in the remaining area is a rather tight squeeze.

First, there is the question of how these routines are to replace the original routines in ROM. An external ROM is needed, and this must be brought into action for addresses of the form 08XX. Decoding eight bits could be rather complex, so a pair of comparator chips are used (see Fig 20). These will produce an output signal when the top eight address bits show the 08H state, and the signal can be applied to the main ROM enable ($\overline{\text{ROMCS}}$) to put the ROM out of action, and will also serve to activate the external ROM.

Since it might appear inconvenient to set up a ROM with only 256 locations, it should be said that the ROM could also be brought into action for addresses in the 2000-2800 range, so that other parts of its contents could be used for other purposes. If this is done, the ROM ghost in that area must be suppressed.

The contents of the original ROM in the 0800-0850 and 08E2-08FF areas must be reproduced in the external ROM, and the revised printer driver must be entered into the intervening locations.

The first section of the program is straightforward, but the next routine has to loop back on itself to maintain the normal entry points.

```
PRINT   0851  CD 78 08  CALL  RECODE  ;Subroutine to get
                                      ASCII
        0854  F5        PUSH  AF      ;Save A during ready
                                      check.
L1      0855  DB FB     IN    A,(FB)  ;Check ready
        0857  28 FC     JR    Z,L1    ;loop if not ready
        0859  F1        POP   AF      ;Restore data
        085A  D3 FB     OUT   (FB),A  ;Output data
        085C  C9        RET
L2      085D  7E        LD    A,(HL)  ;Get byte
        085E  FE 76     CP    76H     ;NEWLINE ?
        0860  20 02     JR    NZ,L3   ;
        0862  15        DEC   D       ;If so, decrement D,
                                      and
        0863  C8        RET   Z       ;return if D = 0
L3      0864  CD 51 08  CALL  PRINT
        0867  18 05     JR    L4      ;Loop
COPY    0869  16 16     LD    D,16H   ;Set up for 22 lines.
        086B  2A OC 40  LD    HL,(400C);Pick up Display File
                                      pointer.
L4      086E  23        INC   HL
        086F  18 EC     JR    L2      ;Get and print a byte.
LINE    0871  16 01     LD    D,1     ;Set up for one line
        0873  21 3C 40  LD    HL,403C ;Point to Print Buffer
        0876  18 E5     JR    L2
RECODE  0878  FE OB     CP    OBH
        087A  38 5D     JR    C,L7    ;00-0A = space
        087C  06 25     LD    B,25H
        087E  D6 OE     SUB   OEH     ;A=Code - OE
        0880  38 5B     JR    C,L6    ;OB-OD:+25-OE=
                                      +17H
        0882  06 3A     LD    B,3AH
        0884  28 55     JR    Z,L5    ;OE = 3A
        0886  06 3F     LD    B,3FH
        0888  3D        DEC   A       ;A=Code - OF
        0889  28 50     JR    Z,L5    ;OF = 3F
        088B  06 2A     LD    B,2AH
        088D  D6 03     SUB   3       ;A=Code - 12
        088F  38 4C     JR    C,L6    ;10-11: +2A-12=
                                      +18H
```

```
         0891   06 3C    LD      B,3CH
         0893   28 36    JR      Z,L5        ;12 = 3C
         0895   06 3E    LD      B,3EH
         0897   D6 03    SUB     3           ;A=Code-15
         0899   38 9B    JR      C,L6        ;13-14:
                                             +3E-15=+29H
         089B   06 2B    LD      B,2BH
         089D   28 3C    JR      Z,L5        ;15 = 2B
         089F   06 2D    LD      B,2D
         08A1   3D       DEC     A           ;A=Code-16
         08A2   28 37    JR      Z,L5        ;16 = 2D
         08A4   06 2A    LD      B,2AH
         08A6   3D       DEC     A           ;A=Code - 17
         08A7   28 32    JR      Z,L5        ;17 = 2A
         08A9   06 2F    LD      B,2FH
         08AB   3D       DEC     A           ;A=Code - 18
         08AC   28 2D    JR      Z,L5        ;18 = 2F
         08AE   06 3B    LD      B,3BH
         08B0   3D       DEC     A           ;A=Code - 19
         08B1   28 28    JR      Z,L5        ;19 = 3B
         08B3   06 2C    LD      B,2CH
         08B5   3D       DEC     A           ;A=Code - 1A
         08B6   28 23    JR      Z,L5        ;1A = 2C
         08B8   06 2E    LD      B,2EH
         08BA   3D       DEC     A           ;A=Code - 1B
         08BB   28 1E    JR      Z,L5        ;1B = 2E
         08BD   06.3A    LD      B,3AH
         08BF   D6 OB    SUB     OB          ;A=Code - 26
         08C1   38 1A    JR      C,L6        ;1C-25: + 3A-26=
                                             +14H
         08C3   06 5B    LD      B,5BH
         08C5   D6 1A    SUB     1A          ;A=Code - 40
         08C7   38 14    JR      C,L6        ;26-3F: +5B-40=
                                             +1BH
         08C9   06 OD    LD      B,ODH
         08CB   FE 36    CP      36          ;76?
         08CD   28 OC    JR      Z,L5        ;76 = OD
         08CF   D6 66    SUB     66          ;A=Code - A6
         08D1   38 06    JR      C,L7        ;40-A5 = space
         08D3   06 7B    LD      B,7BH
         08D5   D6 1A    SUB     1A          ;A=Code - CO
         08D7   38 04    JR      C,L6        ;A6-BF: +7B-CO=
                                             +BBH
L7       08D9   06 20    LD      B,20H
L5       08DB   78       LD      A,B
         08DC   C9       RET
```

```
L6      08DD  80      ADD   A,B
        08DE  C9      RET
```

If the translations in RECODE are traced through, it will be found that codes OB-3F are reproduced as they are shown in the ZX81 display, while the reverse video alphabetic codes A6-BF are printed as lower case letters. All other codes produce a space. As with the normal display, token codes are translated to text before output.

There is no provision for BREAK, or for running in FAST mode, due to the restricted memory area available. Since the number of bytes to be handled is only one eighth of the number required by the Sinclair printer, the lack of FAST mode should not be important, and the lack of BREAK should only prove a temporary annoyance. Those who wish to add the missing features must move RECODE to another address area. The routine is freely relocatable.

Minor changes may be needed for some printers. For example, an active low READY signal requires that the jump at 0857 must be changed from JR Z to JR NZ.

So much for the software. What hardware is needed? Apart from the ROM, which will require the sort of circuit described elsewhere in this book, an output port is needed to drive the printer, and an input port to check the READY line. The 8255 PPI in the last section might be used, but simple ports would be quite sufficient.

The essential connections for a printer with the 'Centronics' type of interface are;

- The data lines from the output port
- The data strobe from the output port
- The READY line to the input port
- Ground

Details of the connection pins involved will be supplied with the printer. Unused pins of the input port should be grounded to avoid random or high states in the related bits.

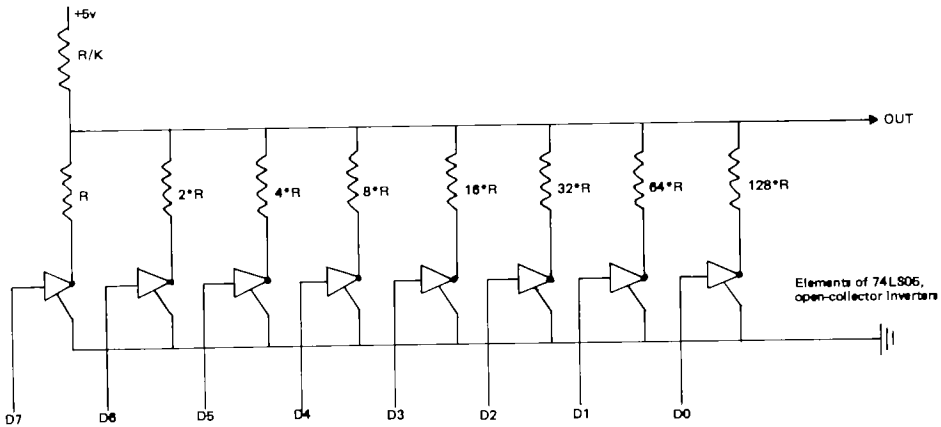


FIG 21 – DIGITAL-TO-ANALOGUE CONVERTER

This complex project needs to be undertaken with care, but should not prove impossibly difficult.

Analogue Output

Digital outputs can be used in a number of different ways, from data transfer to the control of power circuits, but conversion from digital to analogue form opens up a further range of options.

The basic digital-to-analogue converter is shown in Fig. 21, relative resistor values being used because the actual values should be chosen to suit particular requirements. If K is small, the output voltage across the series resistor is a maximum, but the relationship between the digital input and the output voltage is non-linear. If K is large, the linearity is much better, but the output voltage is small. A typical value of K is 10-20.

Those who wish to examine this further can do so by using the program 'D/A' in Appendix A. This uses a FOR loop to calculate:

$Y = X_7 + X_6/2 + X_5/4 + ... + X_0/128$
where x_n is bit n of the digital input. (= X)
Then $V = 5\ (Y/(K + Y)$
gives the output voltage across the series resistor, and V/X shows the ratio of output voltage to digital input.

The hardware required is simple, though it is sometimes inconvenient that the output is developed across a resistor attached to +5V, rather than to earth. If you need to disable the output, replace the open-collector buffers by open-collector AND gates.

Initial experiments may well consist of comparing the readings on a meter placed across the series resistor with digital output bytes, but for maximum effect the system needs to be fed with a string of output bytes to produce a varying voltage, which can be displayed on an oscilloscope or fed to a sound system.

If you have enough store space, set up a sine-wave table as an

array, so that you can create a stream of outputs from the table. This will produce a rounded tone, the pitch depending on the speed with which the table is scanned. To get the higher notes, you will probably need to use machine code, BASIC being too slow.

Then set up a second scanning program running through the table at a different speed. Add the two outputs, and feed them to the analogue output channel. Two notes are produced.

Expanding on this, it is possible to produce quite complex musical sounds. A fully detailed explanation would be out of place here, because computer music can be a major study in its own right, but a little experimentation can work wonders.

Remember to use FAST mode when generating waveforms, as the stop-go action in SLOW mode will make the output stream irregular.

If music fails to appeal, the use of analogue output to control servo systems can be interesting. The principle is that the position of the driven member of the servo is indicated by a voltage, perhaps generated by a potentiometer. This voltage is compared with the analogue output voltage, and the difference between the two voltages is amplified and used to drive a motor which will move the driven member in such a way that the voltage difference is reduced. When the two voltages match, the motor stops. You can therefore send out instructions from the computer to tell the system what position the driven member should take up. A particular application is to a model grab crane, which can be made to pick up objects with remarkable precision.

This quick look at the possibilities of analogue output has necessarily been brief: A whole book could be devoted to the subject.

Analogue Input

Analogue input requires the conversion of analogue data to digital form, which is more difficult than the reverse process.

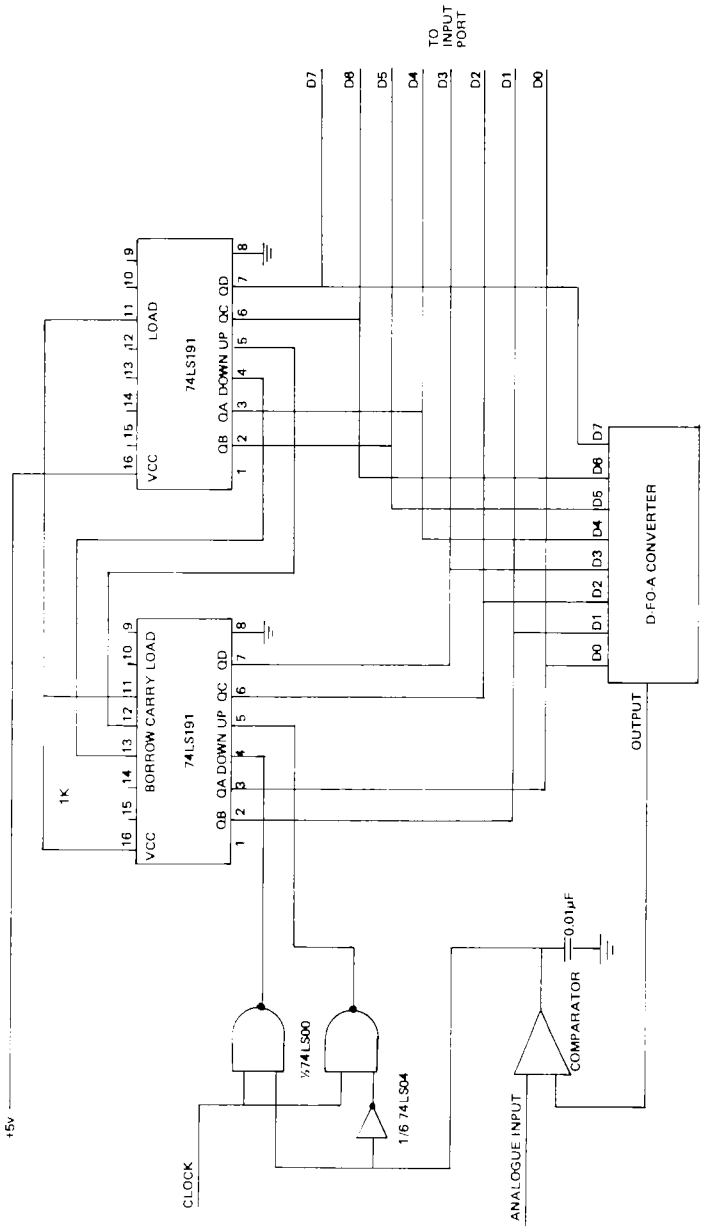


FIG 22 – ANALOGUE TO DIGITAL CONVERTER

If the speed of response required is not high, then a system of the form shown in Fig. 22 will serve. An up-down counter, using a pair of 74LS191s, is coupled to a simple digital-to-analogue converter of the type already described. The analogue output is compared with the incoming voltage, and if the incoming voltage is higher the counter is counted up, if the incoming voltage is lower the counter is counted down.

The counter output is passed to an input port, which must be of the latching type, and must be set during the interval between two counter clocks, to ensure that there are no timing problems. Even then, there is a risk that the port may be set to a new value as it is being read, so the setting action must be blocked during an input read.

For faster response, a successive approximation system is needed. This generates a series of sample voltages, the first equal to half the working range, the rest equal to half the voltage that precedes them. If the resulting total voltage is lower than the input voltage, the latest sample is retained, otherwise it is discarded. The digital output contains a 1 for each sample retained, a 0 for each sample discarded.

The problem here is that the system depends on the input voltage remaining constant while the comparison is being made. Unlike the counter method, it is not possible to follow changes in input voltages continuously. It is therefore necessary to use a 'sample and hold' circuit, which samples the input voltage at a given moment, and holds that voltage constant at the comparator input during the comparison. The sample and hold action can be initiated by a special input command preceding that which reads the digital output. There must be enough time between the two inputs for the conversion to take place.

The more complex forms of analogue input system are best based on purchased modules, as the techniques of laser-etching resistors to correct values, essential for maximum accuracy, are only available to specialists in the field.

Where samples need only be taken fairly slowly, a single

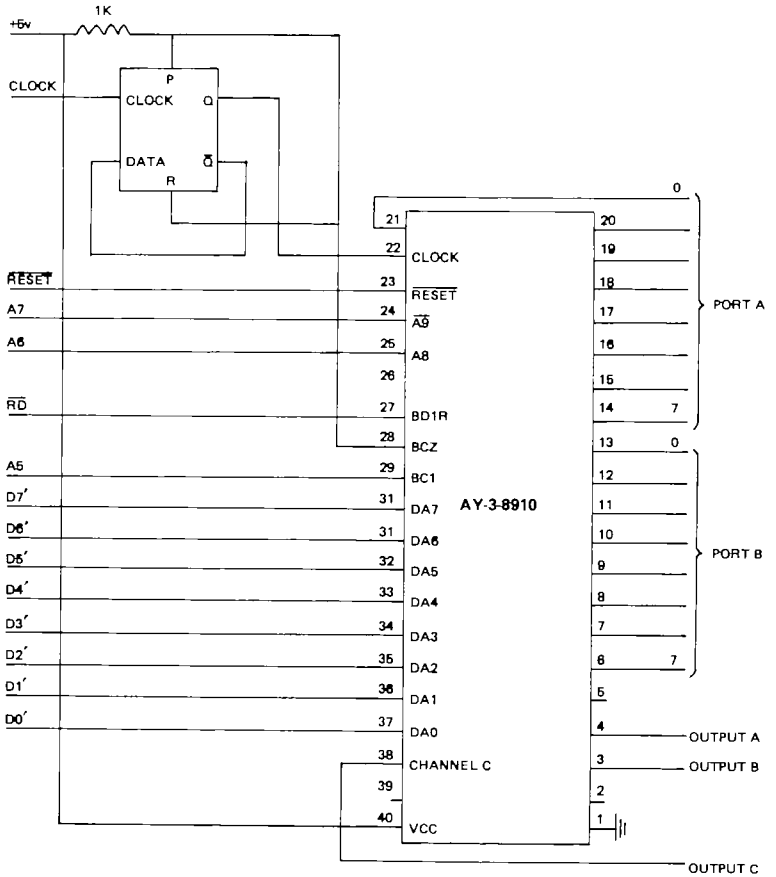


FIG 23 – CONNECTIONS FOR PROGRAMMABLE SOUND GENERATOR AY-3-8910

analogue input system can be made to serve a number of voltage sources by the use of multiplexing. The sources are selected in turn by a digital output which switches in the sample and hold for that source, setting up the sample at the same time. A reading is taken, and the next source is then selected.

One of the biggest problems in both analogue input and analogue output is the scaling of the digital/analogue relationship. With an 8-bit system, this becomes especially important, because such a system can only have a resolution of one part in 256, and if that resolution is not fully used there will be a loss of accuracy. Values should be chosen so that the input voltage range runs close to the maximum for the system. Once a digital word has been read into the computer, it can be handled more freely, using multipliers to give it a representative range.

As with all boundaries between slightly different technologies, analogue work in association with a computer can produce many headaches of this sort. In general, the rule to follow is simple: Forget about precise numbers, and the normal systems of units. Just make sure that a maximum analogue level corresponds to a digital value of a little less than FFH, and worry about the scaling afterwards.

A Special Peripheral

The AY-3-8910 Programmable Sound Generator is a very versatile component, though its use is slightly hampered by the fact that it was designed for use with a processor having common address and data lines. Sixteen registers are provided, each with its own address. Fourteen determine the sound output and the other two form input/output ports.

It might be possible to multiplex the Z80 data and address lines together, but a lot of logic would be needed and there might be timing problems. The solution adopted here (Fig 23) is to supply the address and data on two output ports, with an optional input port if the full facilties of the PSG are required. This makes control more complex but simplifies the logic.

The 8910 has four working modes, determined by the state of inputs BDIR and BC1;

BDIR	BC1	
0	0	Inactive
0	1	Read from PSG
1	0	Write to PSG
1	1	Latch Address.

BDIR is connected to $\overline{RD}$, and BC1 is connected to A5. An input on an address of the form XX1XXXXX will read from the PSG, an output on the same address will latch the PSG address, and an output on XX0XXXXX will pass data to the PSG.

The working addresses are further defined by connecting A7 to pin 24, a chip enable in the active low sense, so A7 in the address must be low. A6 is connected to pin 25, which is an active high chip enable, so A6 in the address must be high. Remembering that bits A0-A2 must be high to avoid calling ZX81 internal functions, the required addresses become 01011111 (5FH) and 01111111 (7FH), the latter being used for reading from the PSG and latching addresses.

Setting up a tone on Channel A requires that register 0 and the lower half of register 1 be set to form a 12-bit word representing the 8910 clock frequency divided by sixteen times the required tone frequency. The clock is half the frequency of the ZX81 clock, i.e. 1.625 MHz, so to produce a note A = 440 Hz the word must be 231, or OE7H. The transfers required are;

```
LD    A,0
OUT   (7F),A    Latch address 0
LD    A,E7
OUT   (5F),A    Set data E7H in Register 0
LD    A,1
OUT   (7F),A    Latch address 1
LD    A,0
OUT   (5F),A    Set zero in register 1
```

A similar process with registers 2 and 3 will set channel B tone, while registers 4 and 5 serve channel C.

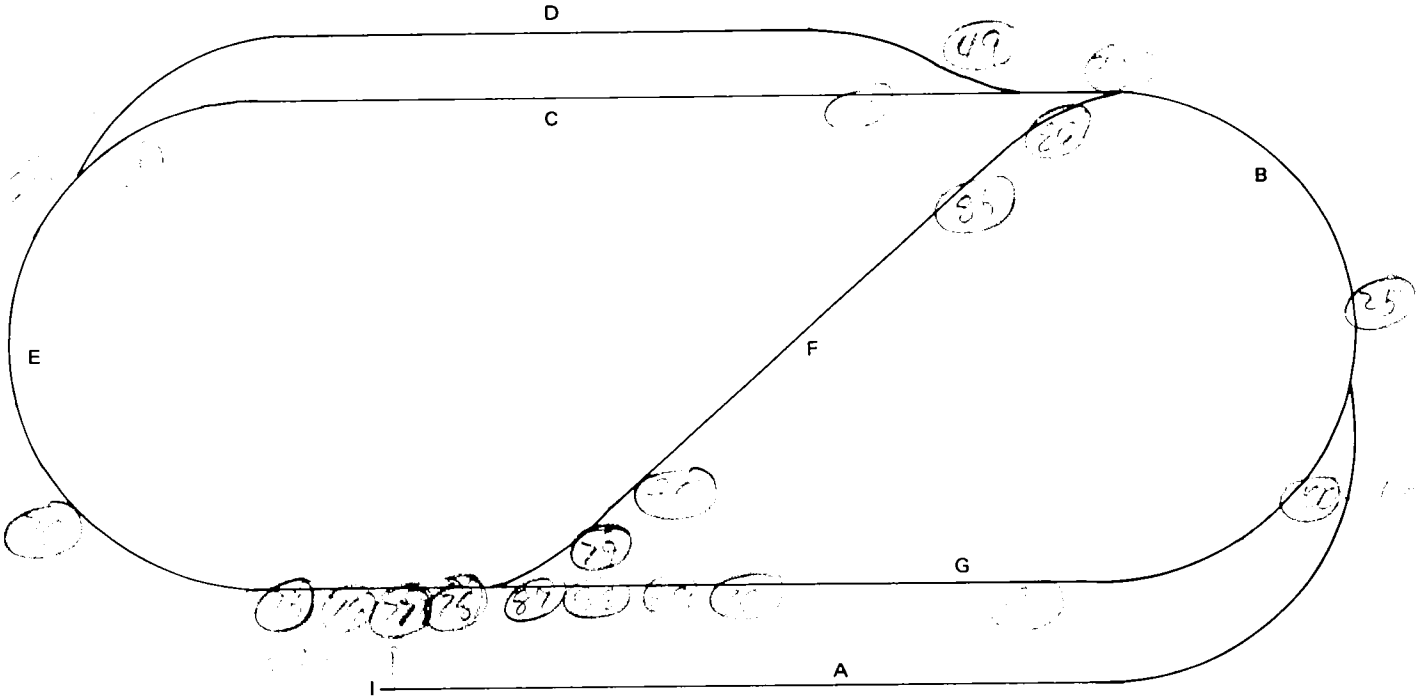


FIG 24 – RAILROAD LAYOUT

The required channels must now be selected by setting register 7, placing high bits in positions 0,1 and 2 for channels A,B and C. A high bit in positions 3,4,5 will enable the noise channels, the pitch of which is set by register 6. (Bits 6 and 7 of register 7 control the input/output ports, 0 selecting input and 1 selecting output).

The amplitude of the output on the three sound channels is set by registers 8 – 10. The four lowest bits of each register set steady amplitude, but if bit 4 =1 the envelope control takes over. The envelope period is determined by registers 12 and 13 as a 16-bit word, and the type of envelope is determined by register 13.

The ports are registers 14 and 15.

This short outline does scant justice to a versatile device, but provides an illustration of the approach to interfacing special peripherals. In the final analysis, the hardware is relatively simple in relation to the software needed . . .

A Control System

To complete this study of ZX81 add-ons, it seems appropriate to outline a fairly complex control system, the methods used being applicable to other types of system.

The physical system to be controlled is a model railroad shown in Fig. 24. Though apparently simple, it is in fact a classic configuration, and its five turnouts allow no less than thirty-two route combinations to be established. With two trains, the seven track segments allow up to forty-two situations to be created. Fortunately, control can be on a relatively simple basis.

It will be stipulated that a train on section C can only proceed towards E, a train on section D can only proceed towards B. It will be assumed that a train will always want to proceed to the next section it can enter.

The control program must be a continuous loop, so that it is continually checking the situation and making decisions

about permissible moves. First, it needs to know where the trains are, and it will store that information in seven single-letter variables, e.g. E = 1 means a train on segment E moving clockwise, E = −1 means a train on segment E moving the other way, and E = 0 means that the segment is unoccupied.

Two-letter variables indicate the status of routes. AB = −1 means that the route from A to B is barred, AB = 0 means that it is permitted, AB = 1 means that it has been allocated.

The program must then analyse the situation and adjust the variables accordingly. This process needs to reflect simple logical thought. For example, if a train is at A, waiting to depart, it can only move when AB = 1, and this allocation must set the turnout giving the train access to section B.

The routine might be;

```
1000  IF AB = 1 AND A = 1 THEN GOTO 1040
1010  LET AB = 0
1020  IF B<>0 OR DB =  OR FB = 1 OR GB = 1 THEN LET
AB = −1
1030  IF A = 1 AND AB = 0 THEN LET AB = 1
```

Line 1000 ensures that route AB remains unaltered while the train using it has not cleared section A.

Line 1010 makes route AB permissible on a tentative basis.

Line 1020 reverses that if B is occupied or a route into B is already established.

Line 1030 allocates route AB if it is permissible and there is a train in A heading towards B.

This module is repeated for all the other routes. In a control system, the route check would be followed by a section setting up the turnouts and applying power in accordance with the authorised routes.

However, there is no need to have an actual layout running to check the performance of the program, and the version given here shows how that is done. It produces a mimic diagram

showing the position of two trains as they move round the tracks. Even if a full control system was planned, the mimic diagram would give the owner of the precious rolling stock welcome confidence that there would be no damage in serious use.

If a third train is added, the system can come to a stalemate, with all the trains approaching, say, the A/B/G turnout at the same moment. You may care to work out how this might be avoided . . .

TRAINS

```
100 DIM A (100,2)
105 LET FLAG = 1
110 PRINT AT 12,13; "TRAINS
120 GOSUB 2000
130 CLS
140 LET AB=0
150 LET BA=0
160 LET BC=0
170 LET BF=0
180 LET BG=0
190 LET CE=0
200 LET DB=0
210 LET ED=0
220 LET EF=0
230 LET EG=0
240 LET FB=0
250 LET FE=0
260 LET GB=0
270 LET GE=0
280 LET A=1
290 LET B=0
300 LET C=1
310 LET D=0
320 LET E=0
330 LET F=0
340 LET G=0
350 GOSUB 1000
360 LET X1=0
370 LET X2=0
380 LET Y1=0
390 LET Y2=0
400 LET P1=1
410 LET P2=44
420 IF FLAG=1 THEN GOSUB 3000
430 IF FLAG=1 THEN GOSUB 5000
440 LET PA = P1
450 LET X=X1
460 LET Y=Y1
470 LET A$="1"
480 GOSUB 6000
490 LET P1=PA
```

The long initialisation is necessary because so many variables are used. The letters A — G indicate the track sections. The two letter variables indicate routes.

X and Y give the screen co-ordinates for the two trains, subscripts 1 and 2 being used, with no subscripts for the common variables used by sub 6000.

The P variables similarly give the position pointers used to read X and y from the array A().

FLAG is set to 1 only when the situation changes, so that routes are updated only when necessary.

The main execution loop of the program consists of lines 420 to 600.

```
500 LET X1=X
510 LET Y1=Y
520 LET PA=P2
530 LET X=X2
540 LET Y=Y2
550 LET A$="2"
560 GOSUB 6000
570 LET P2=PA
580 LET X2=X
590 LET Y2=Y
600 GOTO 420


1000   FOR N = 0 TO 30
1010   LET ANG=PI*N/30
1020   PLOT 12-10*SIN ANG,24-10*COS ANG
1030   PLOT 37+10*SIN ANG,24+10*COS ANG
1040   IF N = 15 THEN PLOT 37+10*SIN ANG, 14+10*COS ANG
1050   NEXT N
1060   FOR N = 1 TO 27
1070   PLOT N+10,4
1080   PLOT N+10,14
1090   PLOT N+10,34
1100   NEXT N
1110   FOR N = 1 TO 18
1120   PLOT 18+N, 15+N
1130   NEXT N
1140   FOR N = 0 TO 7
1150   PLOT 47,N+14
1160   NEXT N
1170   PLOT 12,35
1180   PLOT 13,36
1190   FOR N = 14 to 29
1200   PLOT N,37
1210   NEXT N
1220   PLOT 30,36
1230   PLOT 31,35
1240   PRINT AT 18,15;"A"
1250   PRINT AT 8,22;"B"
1260   PRINT AT 6,10;"C"
1270   PRINT AT 1,10;"D"
1280   PRINT AT 10,2;"E"
```

This subroutine plots the track layout on the screen.

```
1290  PRINT AT 8,13;"F"
1300  PRINT AT 13,15;"G"
1310  RETURN
```

```
2000  FOR N = 1 TO 15
2010  LET A(N,1)=21
2020  LET A(N,2)=N+6
2030  NEXT N
2040  FOR N = 1 TO 3
2050  LET A(N+15,1)=21-N
2060  LET A(N+15,2)=21+N
2070  NEXT N
2080  FOR N = 1 TO 12
2090  LET A(N+18,1)=18-N
2100  LET A(N+18,2)=24
2110  NEXT N
2120  FOR N = 1 TO 3
2130  LET A(N+30,1)=6-N
2140  LET A(N30,2)=24-N
2150  NEXT N
2160  FOR N=1 TO 3
2170  LET A(N+33,1)=3
2180  LET A(N+33,2)=21-N
2190  NEXT N
2200  FOR N = 1 TO 12
2210  LET A(N+36,1)=5
2220  LET A(N+36,2)=17-N
2230  NEXT N
2240  FOR N = 1 TO 10
2250  LET A(N+48,1)=2
2260  LET A(N+48,2)=16-N
2270  NEXT N
2280  FOR N = 1 TO 3
2290  LET A(N+58,1)=3
2300  LET A(N+58,2)=6-N
2310  NEXT N
2320  FOR N = 1 TO 3
2330  LET A(N+61,1)=3+N
2340  LET A(N+61,2)=3-N
2350  NEXT N
2360  FOR N = 1 TO 6
2370  LET A(N+64,1)=6+N
2380  LET A(N+64,2)=0
```

This subroutine sets up an array defining the positions at which the train numbers appear beside the track.

```
2390    NEXT N
2400    FOR N= 1 TO 3
2410    LET A(N+70,1)=12+N
2420    LET A(N+70,2)=N
2430    NEXT N
2440    FOR N = 1 TO 4
2450    LET A(N+73,1)=15
2460    LET A(N+73,2)=3+N
2470    NEXT N
2480    FOR N = 1 TO 9
2490    LET A(N+77,1)=14-N
2500    LET A(N+77,2)=10+N
2510    NEXT N
2520    LET N = 1 TO 13
2530    LET A(N+86,1)=15
2540    LET A(N+86,2)=8+N
2550    NEXT N
2560    LET A(100,1)=14
2570    LET A(100,2)=22
2580    RETURN
```

This is the route check logic routine

```
3000    IF AB=1 AND A=1 THEN GOTO 3040
3010    LET AB=0
3020    IF B <> 0 OR DB=1 OR FB=1 OR GB=1 THEN LET AB=-1
3030    IF AB=0 and A=1 THEN LET AB=1
3040    IF BA=1 and B=-1 THEN GOT 3090
3050    LET BA=0
3060    IF A <> 0 THEN LET BA = -1
3070    IF RND > .5 AND G=0 AND EG <> 1 THEN GOTO 3090
3080    IF BA=0 AND B = -1 THEN LET BA = 1
3090    IF BC=1 AND B=1 THEN GOTO 3140
3100    LET BC=0
3110    IF C=1 OR BF=1 THEN LET BC=-1
3120    IF RND > .5 AND F=0 AND EF <> 1 THEN GOTO 3140
3130    IF BC=0 AND B=1 THEN LET BC=1
3140    IF BF=1 AND B=1 THEN GOTO 3180
3150    LET BF=0
3160    IF F <> 0 OR BC=1 OR EF=1 THEN LET BF=-1
3170    IF BF=0 AND B=1 THEN LET BF=1
3180    IF BG=1 AND B=-1 THEN GOTO 3220
3190    LET BG=0
```

```
3200  IF G <> 0 OR EG=1 OR BA=1 THEN LET BG=-1
3210  IF BG=0 AND B=-1 THEN LET BG=1
3220  IF CE=1 AND C=1 THEN GOTO 3260
3230  LET CE=0
3240  IF E <> 0 OR FE=1 OR GE=1 THEN LET CE=-1
3250  IF CE=0 AND C=1 THEN LET CE=1
3260  IF DB=1 AND D=-1 THEN GOTO 3300
3270  LET DB=0
3280  IF B <> 0 OR AB=1 OR FB=1 OR GB=1 THEN LET DB=-1
3290  IF DB=0 AND D=-1 THEN LET DB=1
3300  IF ED=1 AND E=-1 THEN GOTO 3340
3310  LET ED=0
3320  IF D <> 0 THEN LET ED=-1
3330  IF ED=0 AND E=-1 THEN LET ED=1
3340  IF EF=1 and E=1 THEN GOTO 3390
3350  LET EF=0
3360  IF F <> 0 OR BF=1 OR EG=1 THEN LET EF=-1
3370  IF RND > .5 AND G=0 AND BG <> 1 THEN GOTO 3390
3380  IF EF=0 AND E=1 THEN LET EF=1
3390  IF EG=1 AND E=1 THEN GOTO 3430
3400  LET EG=0
3410  IF G <> 0 OR BG=1 OR EF=1 THEN LET EG=-1
3420  IF EG = 0 AND E=1 THEN LET EG=1
3430  IF FB=1 AND F=1 THEN GOTO 3470
3440  LET FB=0
3450  IF B <> 0 OR AB=1 OR DB=1 OR GB=1 THEN LET FB=-1
3460  IF FB=0 AND F=1 THEN LET FB=1
3470  IF FE=1 AND F=-1 THEN GOTO 3510
3480  LET FE=0
3490  IF E <> 0 OR CE=1 OR GE=1 THEN LET FE=-1
3500  IF FE= 0 and F=-1 THEN LET FE=1
3510  IF GB=1 AND G=1 THEN GOTO 3550
3520  LET GB=0
3530  IF B <> 0 OR AB=1 OR DB=1 OR FB=1 THEN LET GB=-1
3540  IF GB=0 AND G=1 THEN LET GB=1
3550  IF GE=1 AND G=-1 THEN GOTO 3590
3560  LET GE=0
3570  IF E <> 0 OR CE = 1 OR FE=1 THEN LET GE=-1
3580  IF GE=0 AND G=-1 THEN LET GE=1
3590  RETURN
```

```
5000  PRINT AT 0,26;"A=";A;" "
5010  PRINT AT 1,26;"B=";B;" "
5020  PRINT AT 2,26;"C=";C;" "
5030  PRINT AT 3,26;"D=";D;" "
5040  PRINT AT 4,26;"E=";E;" "
5050  PRINT AT 5,26;"F=";F;" "
5060  PRINT AT 6,26;"G=";G;" "
5070  PRINT AT 8,26;"AB=";AB;" "
5080  PRINT AT 9,26;"BA=";BA;" "
5090  PRINT AT 10,26;"BC=";BC;" "
5100  PRINT AT 11,26;"BF=";BF;" "
5110  PRINT AT 12,26;"BG=";BG;" "
5120  PRINT AT 13,26;"CE=";CE;" "
5130  PRINT AT 14,26;"DB=";DB;" "
5140  PRINT AT 15,26;"ED=";ED;" "
5150  PRINT AT 16,26;"EF=";EF;" "
5160  PRINT AT 17,26;"EG=";EG;" "
5170  PRINT AT 18,26;"FB=";FB;" "
5180  PRINT AT 19,26;"FE=";FE;" "
5190  PRINT AT 20,26;"GB=";GB;" "
5200  PRINT AT 21,26;"GE=";GE;" "
5210  LET FLAG = 0
5220  RETURN
```

This subroutine displays the current situation.

```
6000  IF PA > 25 THEN GOTO 6110
6010  IF PA=1 THEN LET A=1
6020  IF PA=1 THEN LET FLAG=1
6030  IF PA=22 AND A=1 AND AB <> 1 THEN GOTO 7500
6040  IF PA <> 25 OR A <> 1 THEN GOTO 6090
6050  LET A=0
6060  LET B=1
6070  LET FLAG=1
6080  GOTO 6320
6090  LET PA=PA+A
6100  GOTO 7500
6110  IF PA >36 THEN GOTO 6340
6120  IF PA <> 26 OR B <> -1 THEN GOTO 6210
6130  LET B=0
6140  LET FLAG=1
6150  IF BA<>1 THEN GOTO 6180
6160  LET A=-1
6170  GOTO 6090
```

Train movement subroutine

```
6180  LET G=-1
6190  LET PA=100
6200  GOTO 7500
6210  IF PA<> 36 OR B <> 1 THEN GOTO 6300
6220  LET B=0
6230  LET FLAG=1
6240  IF BC <> 1 THEN GOTO 6270
6250  LET C=1
6260  GOTO 6410
6270  LET F=-1
6280  LET PA=86
6290  GOTO 7500
6300  IF PA=27 AND B=-1 AND (BA <> 1 AND BG <> 1) THEN
        GOTO 7500
6310  IF PA=34 AND B=1 AND (BC <> 1 and BF <> 1) THEN
        GOTO 7500
6320  LET PA=PA+B
6330  GOTO 7500
6340  IF PA > 48 THEN GOTO 6430
6350  IF PA <> 48 THEN GOTO 6400
6360  LET C=0
6370  LET FLAG=1
6380  LET E=1
6390  GOTO 7500
6400  IF PA=46 AND CE <> 1 THEN GOTO 7500
6410  LET PA=PA+C
6420  GOTO 7500
6430  IF PA > 58 THEN GOTO 6530
6440  IF PA <> 49 OR DB <> 1 THEN GOTO 6500
6450  LET D=0
6460  LET B=-1
6470  LET FLAG=1
6480  LET PA=36
6490  GOTO 6320
6500  IF PA=49 AND DB <> 1 THEN GOTO 7500
6510  LET PA=PA+D
6520  GOTO 7500
6530  IF PA > 77 THEN GOTO 6730
6540  IF PA=61 AND E=-1 AND ED<> 1 THEN GOTO 7500
6550  IF PA=74 AND E=1 AND EF <> 1 AND EG <> 1 THEN
        GOTO 7500
6560  IF PA <> 59 OR E <> -1 THEN GOTO 6620
```

```
6570  LET E=0
6580  LET D=-1
6590  LET PA=58
6600  LET FLAG=1
6610  GOTO 7500
6620  IF PA <> 77 OR E <> 1 THEN GOTO 6710
6630  LET E=0
6640  LET FLAG=1
6650  IF EF <> 1 THEN GOTO 6680
6660  LET F=1
6670  GOTO 6880
6680  LET G=1
6690  LET PA=87
6700  GOTO 7500
6710  LET PA=PA+E
6720  GOTO 7500
6730  IF PA > 86 THEN GOTO 6900
6740  IF PA=80 AND F=-1 AND FE <> 1 THEN GOTO 7500
6750  IF PA=84 AND F=1 AND FB <> 1 THEN GOTO 7500
6760  IF PA <> 78 OR F <> 1 THEN GOTO 6820
6770  LET F=0
6780  LET E=-1
6790  LET PA=77
6800  LET FLAG=1
6810  GOTO 7500
6820  IF PA <> 86 OR F <> 1 THEN GOTO 6880
6830  LET F=0
6840  LET B=-1
6850  LET PA=36
6860  LET FLAG=1
6870  GOTO 7500
6880  LET PA=PA+F
6890  GOTO 7500
6900  IF PA=89 AND G=-1 AND GE <> 1 THEN GOTO 7500
6910  IF PA=98 AND G=1 AND GB <> 1 THEN GOTO 7500
6920  IF PA <> 87 OR G <> -1 THEN GOTO 6980
6930  LET G=0
6940  LET E=-1
6950  LET PA=77
6960  LET FLAG=1
6970  GOTO 7500
6980  IF PA <> 100 or G <> 1 THEN GOTO 7040
```

```
6990   LET G=0
7000   LET B=1
7010   LET PA=26
7020   LET FLAG = 1
7030   GOTO 7500
7040   LET PA=PA+G
7500   PRINT AT X,Y;" "
7510   LET X=A(PA,1)
7520   LET Y=A(PA,2)
7530   PRINT AT X,Y;A
7540   RETURN
```

Subroutine 6000 would be replaced in an actual control situation by a routine to set the turnouts as indicated by the routes, and to detect the presence of trains in a given section, applying power as appropriate. The real control routine would, in fact, be somewhat simpler.

APPENDIX A
BASIC PROGRAMS

DUMP (DECIMAL)

This program will display store contents in decimal. At RUN, an input of the starting address is invited. Then a block of 22 locations will be displayed, the address of each location followed by its contents. CONT will produce the next block of 22 locations.

The program will run in 1K RAM.

```
10  INPUT A
20  LET B = PEEK A
30  PRINT A; "  ";
40  PRINT B
50  LET A = A + 1
60  GOTO 20
```

DUMP (HEXADECIMAL)

This version works in hexadecimal notation, and the conversion routines used will be of value elsewhere. The initial input must be a four-digit hexadecimal number, the leading zeroes included where necessary.

The program will just run in IK RAM.

```
10  DIM G(4)
20  LET Q = 16
30  LET R = 28
40  LET L = 0
50  INPUT A$
60  FOR N = 1 TO 4
70  LET L = L*Q + CODE A$(N) — R
80  NEXT N
90  LET B = PEEK L
100 LET C = INT(B/Q)
```

```
110 LET D = B - Q*C
120 LET H = L
130 FOR N = 1 TO 4
140 LET F = INT(H/Q)
150 LET G(5-N) = H - Q*F
160 LET H = F
170 NEXT N
180 FOR N = 1 TO 4
190 PRINT CHR$ (G(N)+R);
200 NEXT N
210 PRINT "  "; CHR$(C+R);
220 PRINT CHR$(D+R)
230 LET L = L + 1
240 GOTO 90
```

16K DUMP (HEXADECIMAL)

This is a more complex dump for 16K store. It will run in IK, but runs out of memory half way down the screen.

```
10  DIM G(4)
20  LET Q = 16
30  LET R = 28
40  LET L = 0
50  INPUT A$
60  FOR N = 1 TO 4
70  LET L = L*Q + CODE A$(N) - R
80  NEXT N

100 LET H = L
110 FOR N = 1 TO 4
120 LET F = INT(H/Q)
130 LET G(5-N) = H - Q*F
140 LET H = F
150 NEXT N
160 FOR N = 1 TO 4
170 PRINT CHR$(G(N) + R);
180 NEXT N

200 FOR N = 1 TO 8
210 LET B = PEEK L
220 LET C = INT(B/Q)
```

```
230 LET D = B - Q*C
240 PRINT "  "; CHR$(C + R);
250 PRINT CHR$(D + R);
260 LET L  + 1
270 NEXT N
280 PRINT
290 GOTO 100
```

Note that in these programs the definition of Q and R at the start saves significant space, as numeric constants are entered into the program in full floating point form wherever they occur.

LOAD

This program simplifies the loading of machine code data, and provides for checking existing store contents with the option of leaving them as they are or entering new values. At RUN, an input is invited to define the starting address in hexadecimal format. That address and its contents are then displayed. A Newline with no data will produce the address and contents of the next location, but a Newline following the input of a hexadecimal byte will change the location contents and display the new value.

```
100 INPUT U$
110 DIM G(4)
120 LET A = 0
130 FOR N = 1 TO 4
140 LET A = 16*A + (CODE U$(N)) - 28
150 NEXT N
160 LET B = A
170 FOR N = 1 TO 4
180 LET C = INT(B/16)
190 LET G(N) = B - 16*C
200 LET B = C
210 NEXT N
220 FOR N = 1 TO 4
230 PRINT CHR$(G(5-N)+28);
240 NEXT N
```

```
250 LET D = PEEK A
260 LET E = INT(D/16)
270 PRINT "  ";CHR$(E+28);CHR$(D-16*E+28);"  ";
280 INPUT U$
290 IF U$ = " " THEN GOTO 330
300 LET F = 16*(CODE U$(1) - 28)+CODE U$(2) - 28
310 PRINT U$;
320 POKE A,F
330 LET A = A + 1
340 PRINT
350 GOTO 160
```

In a IK machine, this would leave very little room for machine code, even if the expedient of pre-defining 16 and 28 were adopted, as was done in DUMP.

D/A

This program works out the relation between digital input and analogue output for a D/A Converter, using various values of K (see Fig 15). K is input at the start, and will normally be in the range 1-100.

```
10  DIM G(8)
20  PRINT "INPUT K"
30  INPUT K
40  LET X = 1
50  LET H = X
60  FOR N = 1 TO 8
70  LET F = INT(H/2)
80  LET G(N) = H - 2*F
90  LET H = F
100 NEXT N
110 LET Y = 0
120 FOR N = 8 TO 1 STEP - 1
130 LET Y = Y + G(N)/(2**(8-N))
140 NEXT N
150 LET V = 5*(Y/(K+Y))
160 LET R = V/X
170 LET V = (INT(V*1000))/1000
180 PRINT "X=";X;"V=";V;"R=";R
```

```
190 LET X = X + 10
200 IF X 256 THEN GOTO 50
210 STOP
```

The increments in X set in line 190 can be changed to suit personal views.

X represents the digital input, and this is broken down into bit form in array G. Y is then calculated to represent the effective conductance of the shunt network, multiplied by R, and then V is the voltage developed across the series resistor. R is the ratio of V to X, which will change more with small values of K, while larger values of K will give better linearity but less output.

APPENDIX B
MACHINE CODE PROGRAMS

Programming the Z80 processor in machine code is not unduly difficult if a start is made with a limited set of instructions which is expanded as confidence grows, but the ZX81 is not designed to make the use of machine code easy, with provision for SAVE and LOAD of BASIC programs alone. Such expedients as hiding machine code in REM statements have been widely publicised, but is only satisfactory on a fairly limited basis. For example, here is a useful renumber routine which cannot be held by that means because it includes two 76H bytes, which would terminate the REM prematurely;

```
START 0000  3E 76      LD    A,76H      ;Newline Code
      0002  21 76  40  LD    HL,4076    ;Program start
      0005  11 64  00  LD    DE,0064    ;100 is first line number
 LOOP 0008  72         LD    (HL),D     ;write in line number
      0009  23         INC   HL
      000A  73         LD    (HL),E
      000B  01 00  30  LD    BC,3000    ;Set BC out of the way
      000E  ED B1      CPIR             ;Look for Newline
      0010  E5         PUSH  HL         ;Save pointer.
      0011  21 0A  00  LD    HL,000A    ;Line increment=10
      0014  19         ADD   HL,DE      ;Update line number
      0015  EB         EX    DE/HL
      0016  E1         POP   HL         ;Retrieve pointer
      0017  BE         CP    (HL)       ;Newline again?
      0018  20 EE      JR    NZ,LOOP    ;If not, loop.
      001A  C9         RET
```

The routine can go anywhere in store, the addresses given being added to whatever starting address is chosen.

Enter this program using the BASIC LOAD routine given in Appendix A, enter it by a USR call, and it will renumber LOAD

in a flash. (For best effect, change the LOAD line numbers first, except those used as loop points.) GOTO and GOSUB links will not be changed, but putting them right is much easier than renumbering the whole program by hand.

A library of such small routines could be built up, but the only satisfactory way to store them is in a ROM, so that they are always available when wanted. That is the object of the ROM/RAM extension described in the main text.

Quite a useful way to get involved in machine code writing is to take BASIC programs as a starting point. The renumber routine started life that way, and was adapted into machine code when the principles used had been proved. A machine code version of the DUMP routine might be worth trying, though it would need to access the Print routine in the BASIC ROM — the Melbourne House books on the ROM coding would be an invaluable guide there, and would also suggest pieces of code that could be adapted to suit machine code needs, such as the SAVE and LOAD procedures.

Machine code is compact — the renumber routine only 27 bytes, less than a medium sized BASIC line, so a very large number of routines could be fitted into a 2K ROM. Since this book is about hardware, no more than an appetiser for machine code can be offered, but that should have encouraged you to consider the possibilities . . .

The TS1000/Sinclair ZX81 is a marvel of modern electronics design: First released in 1981, it is now available for less than $100, and yet has more computing power than the early computers of the 1950s!

The most remarkable aspect of the TS1000/ZX81 is that it is essentially comprised only of a television modulator and four chips: a RAM chip, which is the memory available to the user, a ROM chip, which contains the inbuilt instructions of the computer, a Z80 chip, which is the central processor, and a ULA chip, which is a chip specially designed and built for Sinclair.

Just how these components are put together and what they do is revealed in this book by Don Thomasson.

THE INS AND OUTS OF THE TS1000/ZX81 contains the complete circuit diagram of this amazing microcomputer, as well as a full discussion of the unorthodox methods which are used in making this microcomputer do what it does.

Don Thomasson discusses the structure of the TS1000/ZX81, and how you can add additional memory and expand its capabilities.

The book also includes a number of projects which demonstrate the potential of this microcomputer, from using it to control a model train set, right through to connecting it to a printer.

This book is an invaluable source of information about the hardware aspects of the TS1000/ZX81.

MELBOURNE HOUSE