

# The Signetics 2650

We continue our survey of microprocessor chips and systems with this article, which takes a more detailed look at the Signetics 2650 device and its currently available evaluation kits. Although a relatively recent entry into the market, the 2650 has a particularly powerful instruction set and very flexible interfacing requirements. It seems likely to become the preferred 8-bit device for general purpose microcomputers.

by JAMIESON ROWE

The Signetics 2650 is an 8-bit microprocessor which is made using well proven N-channel MOS technology. It runs from a single +5V supply, which tends to simplify power supply requirements. All inputs and outputs are TTL compatible, and the chip requires only a single-phase clock signal input. As the chip operates in static mode, there is no minimum clock frequency.

With the original 2650 chip, the maximum clock frequency was 1.25MHz, giving instruction cycle times of from 4.8 to 9.6 microseconds. However, the currently available 2650-1 chip is rated to operate up to 2MHz, reducing the instruction cycle times to the range 3.6 – 7.2us.

The broad architecture of the 2650 chip is shown in the block diagram below. It uses an 8-bit bidirectional data bus, and a separate 15-bit address bus. This gives a direct addressing range of 32,768 bytes

(32k), arranged in four pages of 8,192 bytes.

There are seven 8-bit addressable general purpose registers, one of which is the accumulator RO. The remaining six make up the register stack, and are arranged in two groups of 3 selectable by one of the bits in the Program Status Word register (PSW).

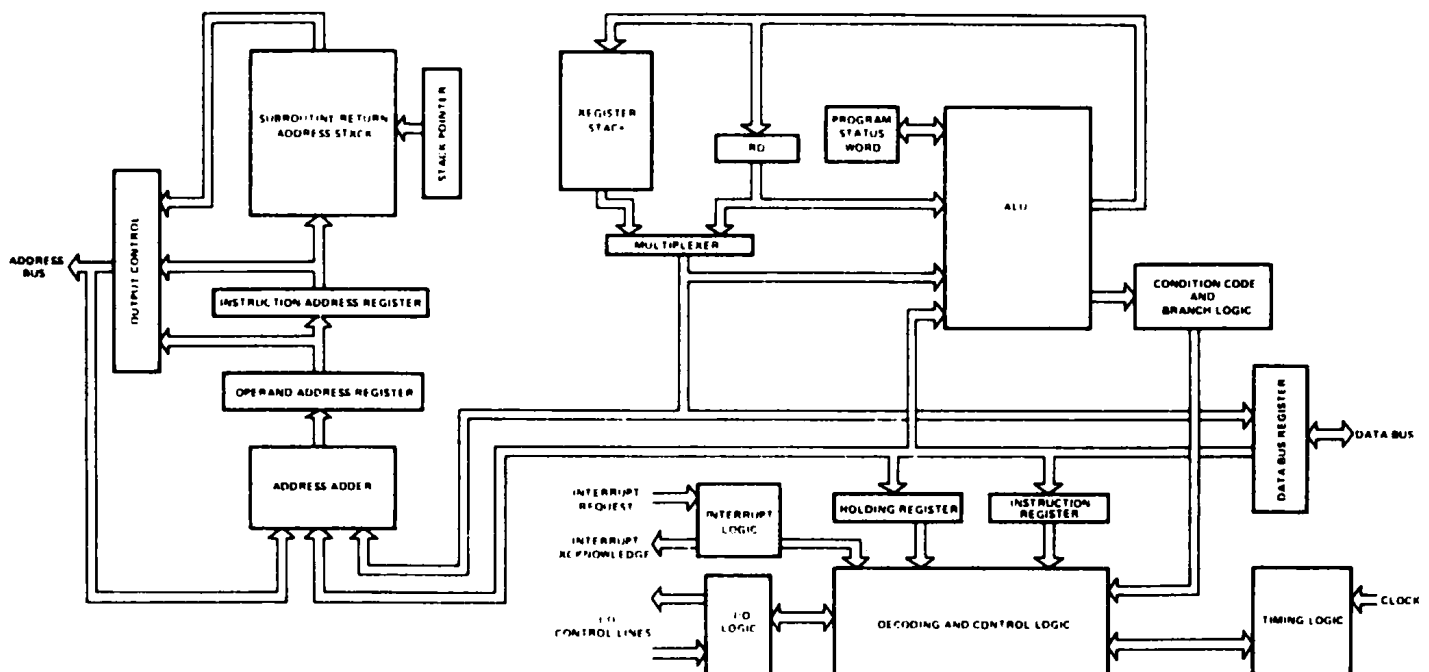
Apart from the register stack there is a subroutine return address stack, consisting of eight 15-bit registers. This allows storage of up to eight return addresses, and hence provides for up to eight levels of subroutine nesting. (A nested subroutine is a subroutine itself called by a subroutine.)

The arithmetic and logic unit (ALU) performs arithmetic, logic and shifting functions. It operates on 8-bits in parallel, and uses carry-look-ahead logic. A second adder is used to increment the instruction address register (program

counter), and also to calculate operand addresses for the indexed and relative addressing modes. This separate address adder, together with the separate instruction address and operand address registers allows complex addressing modes to be implemented with no increase in instruction execution time.

The PSW is a special purpose register which contains status and control bits. The PSW bits may be tested, loaded, stored, preset or cleared using instructions which affect the PSW. Three of the bits act as the pointer for the return address stack; two others act as a "condition code" register, which is affected by the results of compare, test and arithmetic instructions and may be used by conditional branch instruction; other bits store overflow, carry, the selection bit for the two register groups, an interrupt inhibit bit, a carry enable bit, a logical/arithmetic compare select bit, and flag and sense bits for external bit-serial interfacing.

It has been said that the 2650 has the most minicomputer-like instruction set of any microprocessor currently available. There are 75 basic instructions, but many of these are actually subdivided into a number of variations. For example the eight arithmetic instructions may be performed either with, or without car-



The basic architecture of the 2650 microprocessor chip showing major data, address and instruction control paths.

ry/borrow; this also applies to the two rotate instructions. Similarly the four compare instructions may perform either arithmetic or logical comparison, while four of the 12 branch instructions and six of the ten subroutine branch/return instructions are conditional upon the two PSW condition code bits—giving typically about 3 possible variants.

Also although there are nominally only six input-output transfer (IOT) instructions, as distinct from the memory reference instructions (which may also be used for IOT), two of these are "extended" instructions which may address any one of 256 distinct 8-bit input-output ports.

One, two and three-byte instructions are used, giving a high degree of programming efficiency. Register-to-register and simple IOT instructions are one byte, extended IOT instructions are two bytes, while memory-reference instructions are either two or three bytes long as required.

The memory reference addressing modes provided by the 2650 are generally agreed to be the most extensive and versatile of any micro-processor currently available. The modes are as follows:

1. Immediate addressing, with the data mask or value in the second byte of the instruction itself.
2. Direct addressing, either absolute or program relative with a displacement range of from -64 to +63.
3. Direct indexed addressing, absolute.
4. Direct indexed addressing with auto increment.
5. Direct indexed addressing with auto decrement.
6. Indirect addressing, either absolute or

program relative with a displacement range of from -64 to +63.

7. Indirect addressing with post indexing.
8. Indirect addressing with post indexing and auto increment.
9. Indirect addressing with post indexing and auto decrement.

Memory and IOT interfacing of the 2650 is asynchronous, using "handshaking" control signals. This makes the 2650 compatible with almost any type of

memory and peripheral device.

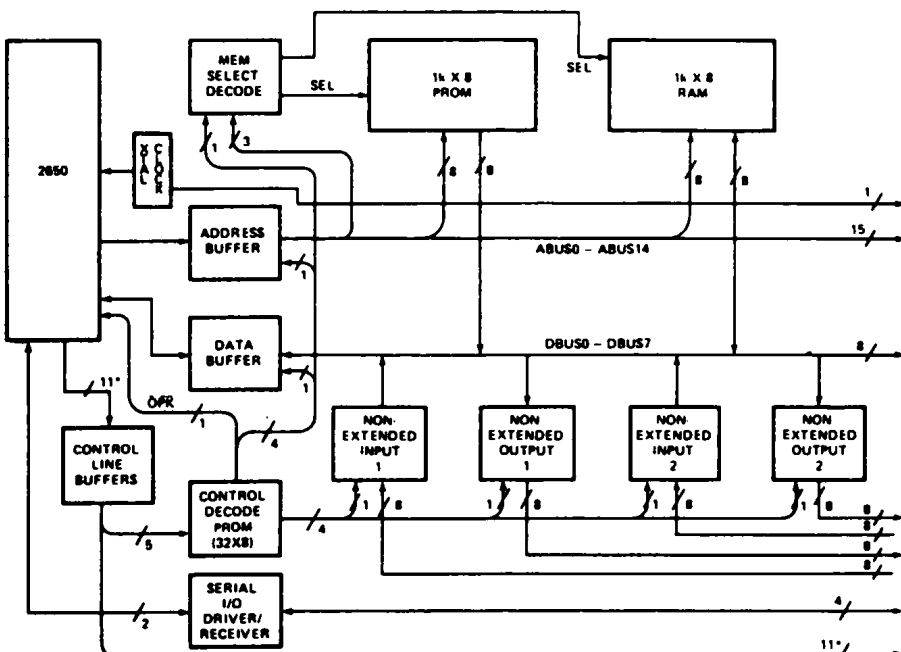
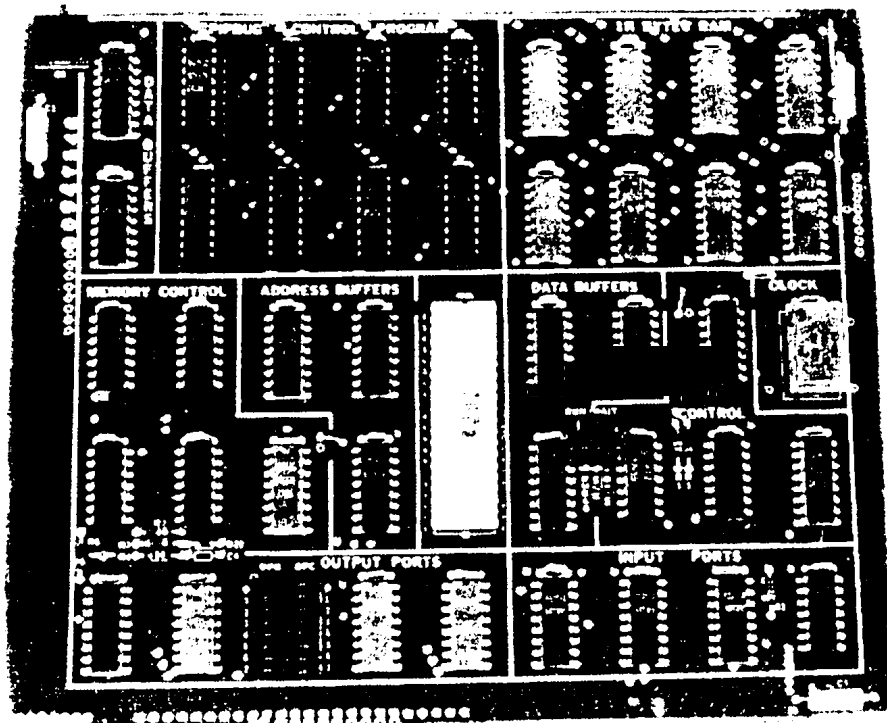
The 2650 has a single level vectored interrupt capability. When it enters the interrupt mode, the chip is able to input an 8-bit address vector via the data bus. This may be used with either direct or indirect addressing to access interrupt servicing routines in any part of the memory space.

As you should be able to see from this brief rundown of its salient features, the 2650 is a particularly flexible micro-processor, and one which is very well suited for general-purpose micro-computer applications. As such it would seem a good choice for anyone seeking to build up a minicomputer-type system based on a microprocessor.

At the same time, the relatively low cost of the basic chip (currently around \$20) and its ability to operate with little more than a clock generator and a ROM in dedicated mode would also make it a good choice for low level applications.

Signetics make two evaluation kits based on the 2650, and these are currently available in Australia from Philips. The more elaborate of the two is the PC1001, which comes as a ready-wired PC board together with edge connector socket and literature. The other kit is a little less elaborate, and comes as either a completely wired PC board or as an assemble-it-yourself kit. In wired form it is designated the PC1500, while the D-I-Y version is the KT9500.

Both kits are basically small microcomputers, capable of being used directly with a power supply and an ASCII



The PC1001 evaluation board system, which is also pictured at top of this page.

teleprinter to develop small 2650 programs in machine language. They could also be expanded into quite pretentious minicomputer systems, by adding further memory and IOT facilities.

An add-on RAM memory board is in fact available, and is directly compatible with either kit. Designated the PC2000, it provides an additional 4k bytes of memory.

At the time of writing this article, we have only had the opportunity to examine and use one of the PC1001 evaluation kits. This is on a PC board measuring 203 × 175mm, with a 100-way double sided edge connector along one of the longer sides. The PC board is pictured, and as you can see there are quite a few IC's apart from the micro-processor.

In fact the PC board is a three-layer assembly, with copper conductors sandwiched in between two layers of epoxy-fibreglass as well as on the two external surfaces. This has allowed Signetics to fit a surprisingly large amount of circuitry on the relatively modest PCB area.

On the PC1001 board is 1k bytes of RAM, capable of storing quite a respectable user program. In addition there is another 1k bytes of ROM, containing a resident monitor-debug program which Signetics have dubbed "PIPBUG". This will be described shortly.

There is an on-board serial asynchronous teleprinter interface, which may be adjusted by means of wire links for either 20mA current loop interfacing or RS232-type voltage interface.

In addition to the teleprinter interface there are four 8-bit parallel IOT ports—two inputs and two outputs. These are wired to be accessed via the two-byte "non-extended" IOT instructions, so that small systems with four or less peripherals (apart from the teleprinter) may be implemented with no further hardware.

The PC1001 board has a 1MHz crystal clock, and therefore is immediately compatible with a 110-baud teleprinter (serial formatting is done by firmware routines, so baud timing is derived from the system clock).

Full data and address bus buffering is provided on the PC1001 board, to simplify addition of further memory or peripherals. All of the control signals are also available at the edge connector in buffered form, which again simplifies any required interfacing.

Although at the time of writing we have not had the opportunity to examine and use the PC1500/KT9500 evaluation kit, we understand that this is based on a PC board identical in size to that of the PC1001. And although the second kit is nominally a less pretentious one, it still offers quite impressive facilities.

For example it still provides 1k bytes

of ROM, with the same resident monitor-debug program (PIPBUG) provided on the PC1001. The only difference in terms of on-board memory is the RAM, which in this case is of only 512 bytes. This is still adequate for a lot of modest programming, of course—and you can always add further memory, as the board again provides fully buffered data, address and control signal buses.

The serial asynchronous teleprinter interface is still provided, but there are now only two 8-bit parallel IOT ports. However, these are programmable in terms of direction, so that they may be used for both input and output.

In place of the crystal clock, the PC1500/KT9500 has an R-C clock oscillator using a 74123 dual monostable.

As not all of the PC board is used by the basic circuitry of the PC1500/KT9500 system, the unused area is provided with plated through holes on 0.3in centres, to allow fitting of additional memory/peripheral decoding ICs if desired.

In short, the PC1500/KT9500 evaluation kit is only a little less flexible than the PC1001. Both are in reality small development systems, capable of being used to develop and run 2650 programs. And in their basic form, each could be used to develop programs for running on the other—apart from the memory size difference. In that sense they are software compatible.

Not only this, of course, but because they use the same resident monitor-debug program they are also virtually identical in the operating sense.

As evaluation kit resident debug programs go, PIPBUG seems quite a flexible one. It recognises seven basic commands, each of which consists of an alphabetic character, any required numerical parameters, and a terminating carriage return. The parameters are given as hexadecimal characters, with leading zeroes unnecessary.

The seven commands and their functions are as follows:

- A — See and alter memory
- B — Set breakpoint (2 permitted)
- C — Clear breakpoint
- D — Dump memory to paper tape
- G — Go to address, run
- L — Load memory from paper tape
- S — See and alter registers

The D command may be used to punch out any desired range of memory locations, with leader, checksum and trailer to facilitate reloading. Both the A and S commands may be auto-incremented, by terminating with a line feed instead of a carriage return.

A full listing of PIPBUG is provided with the evaluation kits, which is very useful. Among other things, it allows the user to make use of the teleprinter servicing subroutines in PIPBUG, by arranging

SIMPLE ANSWER-BACK PROGRAM FOR SIGNETICS PC1001 SYSTEM  
DEVELOPED BY J. ROWE, "ELECTRONICS AUSTRALIA" MAGAZINE 11/7/76  
NOTE: PROGRAM STARTS AT LOCATION 500 (HEX)

## LISTING:

```

500 76 C0      PPSU      40      /SET UP TTY
502 3F 02 86   BSTA,UN CHIN /FETCH CHAR FROM TTY VIA PIPBUG RTN
505 C1         STRZ,R1      /SAVE
506 3F 02 B4   BSTA,UN COUT /ECHO VIA PIPBUG ROUTINE
509 01         LODZ,R1      /RESTORE IN R0
50A A4 0D     SUBI,H0 "CR"  /R0= CHAR -CR
50C 56 74     BRNR,R0 -12   /CR? IF NOT KEEP GOING
50E 04 0A     LODI,R0 "LF" /SUPPLY LF
510 3F 02 B4   BSTA,UN COUT
513 05 00     LODI,R1
515 0D 25 26   LODA,R1 526+ /SET RI=0
518 C3         STRZ,R3      /FETCH ANSWER CHAR
519 3F 02 B4   BSTA,UN COUT /SAVE
51C A7 0D     SUBI,R3 "CR" /PRINT
51E 5B 75     BRNR,R3 -11  /R3= CHAR -CR
520 04 0A     LODI,R0 "LF" /CR? IF NOT KEEP GOING
522 3F 02 B4   BSTA,UN COUT /YES: SUPPLY LF
525 1B 5A     BCTR,UN -36   /BACK TO LOOK FOR NEW INPUT
527 47 4F 20
52A 41 57 41
52D 59 2C 49
530 27 4D 20
533 42 55 53
536 59 21 0D

```

/ANSWER MUST END WITH CR (HEX 0D)

## SAMPLE OF OPERATION:

\*G500

HELLO THERE, WHAT AT PRESENT ARE YOU COMPUTING?  
GO AWAY, I'M BUSY!

DON'T BE LIKE THAT, PLEASE  
GO AWAY, I'M BUSY!

*This simple novelty program was written largely to verify that the teletype servicing routines in PIPBUG could be called by a user program. The listing shows instruction mnemonics and comments as well as the actual instructions in hexadecimal code.*

for application programs to call them as required.

To illustrate this, the author wrote a simple novelty program for the PC1001 system. Its listing and a sample of the operation are reproduced on these pages, and as you can see it does nothing more than monitor input from the teleprinter, waiting for the person at the keyboard to press the carriage return key. When this occurs, it responds by typing out a curt reply: "GO AWAY, I'M BUSY!"

I wrote this little program mainly for practice with the 2650 instruction set, and also to check out the use of the PIPBUG teleprinter servicing routines. The program inputs characters via the "CHIN" subroutine in PIPBUG, whose calling address is 0286, and outputs characters via the "COUT" subroutine whose calling address is 02B4. As you can see the program itself starts at location 0500.

Note that the program uses one, two and three-byte instructions, and requires only 57 bytes of memory including the

answer message buffer. This illustrates the programming efficiency possible with the 2650's powerful instruction set and wealth of memory addressing modes.

If you're interested in the PC1001 or the PC1500/KT9500 evaluation kits or the PC2000 add-on memory, they are available from the Electronic Components and Materials Division of Philips Industries, with offices in each state, or from their distributors. Prices for the kits are as follows:

PC1001 — \$345 plus tax  
PC1500 — \$245 plus tax  
KT9500 — \$165 plus tax  
PC2000 — \$400 plus tax

Each of the basic kits comes with all of the literature needed to use it. All you need is a power supply and a teleprinter. The teleprinter must communicate in ASCII code, as with most other kits. Here at EA we are currently working on a way to allow this to be done at low cost using a surplus Baudot teleprinter. ☐



# A "baby" system using the Signetics 2650

Here is surely the simplest and lowest-cost way of getting to know the Signetics 2650 microprocessor. A complete microcomputer system on a single small PC board, you can build it for around \$70, not counting a power supply or terminal. Despite its low cost, it offers the same debug and monitor program in ROM provided by more expensive systems, together with 256 words of RAM.

by JAMIESON ROWE

As we have noted in earlier articles, the Signetics 2650 microprocessor is a particularly powerful device. Its architecture and instruction set are very minicomputer-like, making it well suited for general-purpose computing as well as low-cost dedicated applications.

In their literature, Signetics note that the device may be used to implement a very low cost minimal "evaluation kit" type system, one which would be very suitable for those wishing to gain experience with the 2650 with the minimum outlay of both time and money. However they themselves have not made such a minimal evaluation system available, only larger systems such as the PC1001 and PC1500 systems.

This seemed rather a pity to us, as at least one other microprocessor has been available in a really minimal system, and this has proved very popular. However as the 2650 and its 2608 ROM chip have been in rather short supply until recently, there seemed little hope of being able to remedy the situation as far as the 2650 was concerned.

Happily this situation has now changed

for the better. Just a few weeks ago we learned from Philips Industries that the 2650 and 2608 chips were now readily available, and at relatively low cost. (Signetics is a US subsidiary of Philips.) We accordingly suggested to them that this would be an ideal opportunity to produce a low-cost "baby" 2650 system, based on the minimal system suggested by Signetics themselves. They agreed, and offered to make available a set of devices if we cared to try the idea.

This project is the result!

Basically, it is a complete general-purpose microcomputer, just like the larger evaluation kits. In fact it has the same debug and monitor program as the larger kits—"PIBBUG"—resident in the 2608 ROM (1k x 8-bit words). It communicates directly with a standard 20mA asynchronous data terminal, such as an ASR-33 Teletype or the video data terminal described in our January and February issues, and requires a single 5V DC power supply.

The main difference between this system and the larger systems is that there is only 256 bytes of RAM memory for

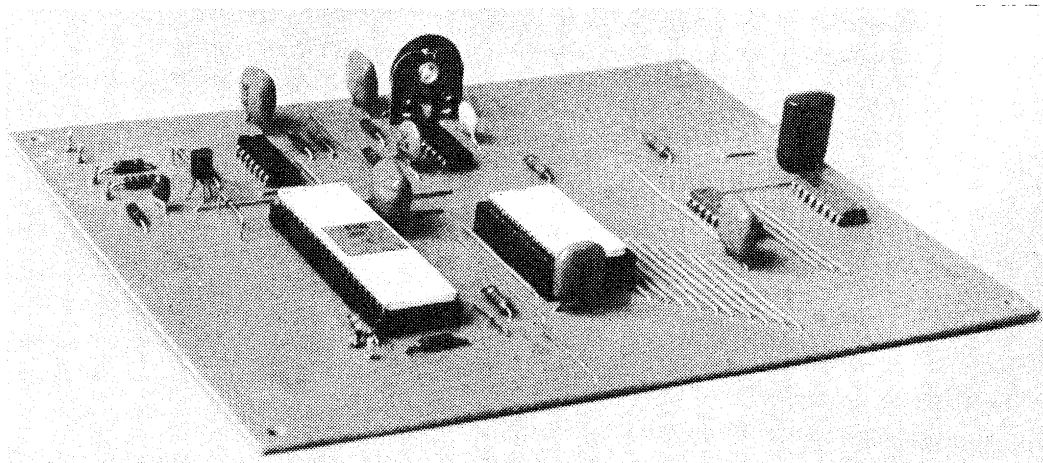
user program storage, and there is no on-board decoding or buffering for further memory or peripheral expansion.

In short, it is a "bare minimum" 2650 system, designed to be the cheapest and easiest way of getting a 2650 up and running. At the same time, it offers the full program development facilities of PIBBUG, including the ability to examine and alter memory from the terminal keyboard; the ability to dump programs to paper tape or cassette, and then load memory from tape; the ability to examine and set the processor registers; the ability to set and remove up to two breakpoints, for debugging; and the ability to run the user's program on command.

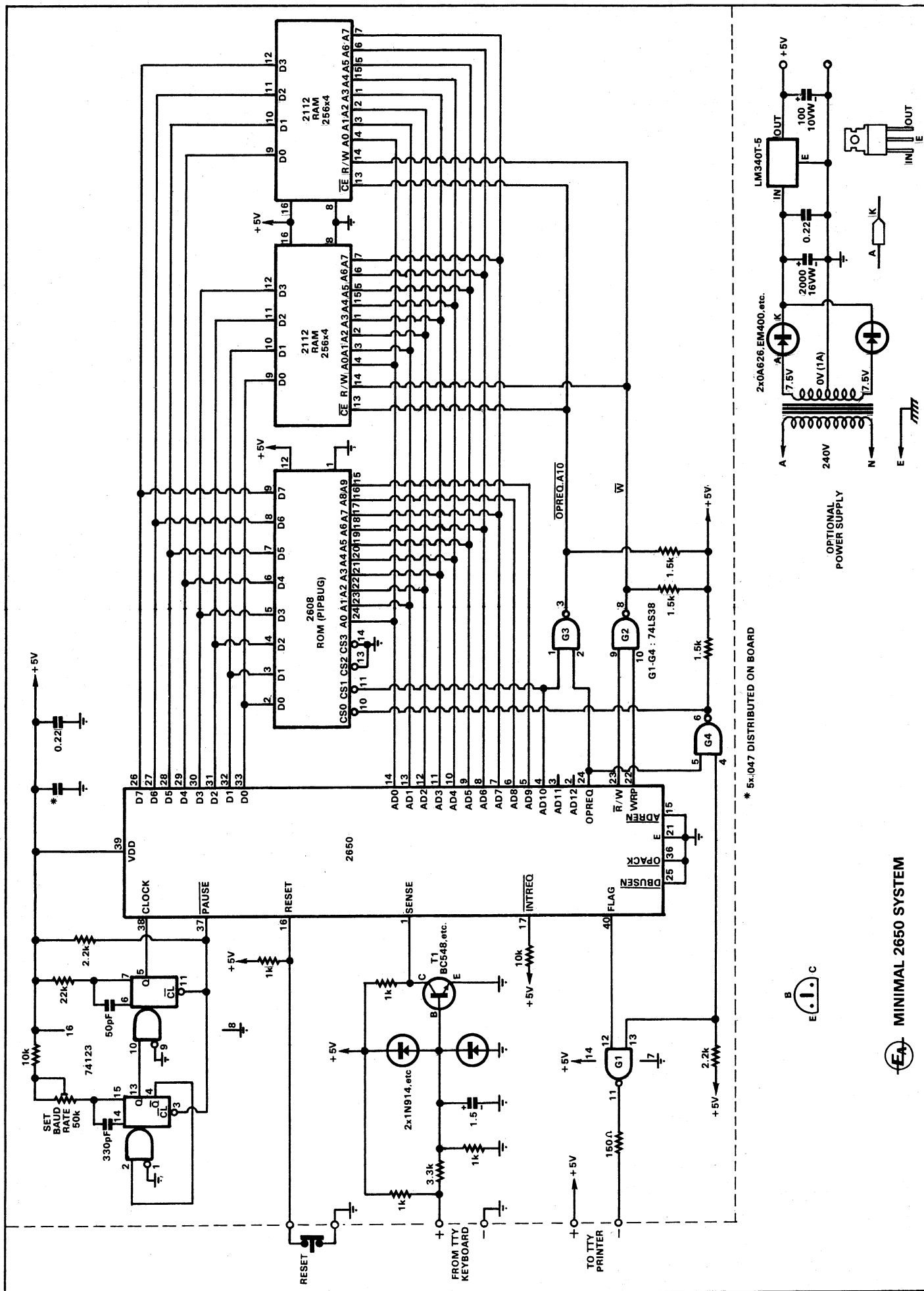
These are quite powerful program development facilities, not usually found on low cost systems. As a result, our "baby" 2650 microcomputer should be particularly suitable for educational and training purposes, whether by schools and colleges or by individual enthusiasts.

As you can see from the diagrams and photograph, it consists of only a handful of parts on a small PC board. There are only six ICs, one transistor and a few resistors and capacitors, and the PCB is single-sided to keep the cost down.

Heart of the circuit is the 2650 chip itself, a powerful 8-bit microprocessor with an instruction set of 75 instructions, and eight different addressing modes. Fabricated using low-threshold ion implantation, it is an N-channel silicon



At left is our new "baby" 2650 microcomputer, complete on its small PC board. It offers the same PIBBUG program in ROM as provided on the larger systems. The full circuit diagram is shown on the facing page, together with an optional power supply.



\* 5x.047 DISTRIBUTED ON BOARD



MINIMAL 2650 SYSTEM

gate device which operates from a single 5V supply and offers TTL compatibility on all inputs and outputs.

A 74123 dual monostable device is used to generate the single-phase 1MHz clock signals for the 2650. The clock oscillator is of the R-C type, but is easily adjusted to the correct operating frequency without the need for elaborate instruments. More about this later on...

As mentioned already, the PIPBUG debug-monitor program is resident in a 2608 ROM. This includes routines for servicing the data terminal input and output, so that the system "knows" how to communicate with a terminal as soon as it is initialised. The code suffix for the 2608 with PIPBUG resident is CN0035.

Two 2112 devices are used to provide the RAM memory of the system. These are low-cost static MOS RAMs, each organised as 256 words or 4 bits, so that the two together provide a RAM of 256 8-bit words. Some 63 words are used by PIPBUG as its scratchpad area, leaving 193 available for user programs.

The remaining IC in the circuit is a 74LS38 low-power Schottky quad NAND buffer, two gates of which are used for simple address decoding to allow the 2650 to differentiate between the ROM and RAM sections of memory.

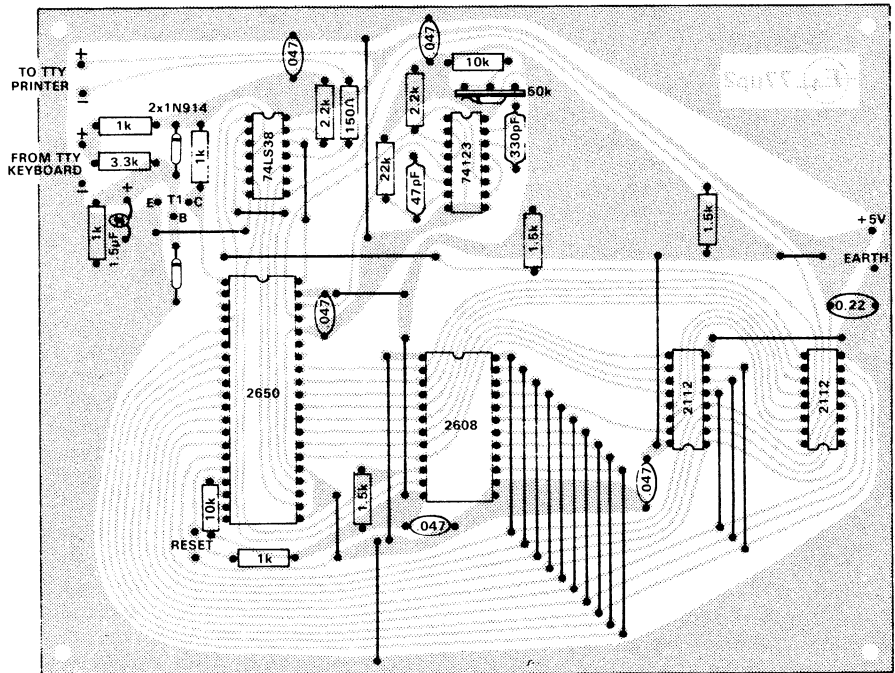
The ROM is allocated to the address range 000-3FF hexadecimal, or the first 1k of memory space. The RAM memory is allocated to the next 256 bytes of memory space, with hexadecimal addresses 400-4FF. Basically this means that when binary address bit AD10 is 0, the ROM is selected, while when it is 1 the RAM memory is selected.

Gate G3 is used to enable the two 2112 RAM devices when AD10 is at the 1 level. The second input of G3 is fed with the OPREQ signal from the 2650, which is a strobing signal used to indicate when bus information is valid.

When AD10 is at the 0 level, the RAMs are therefore disabled. At the same time the ROM is enabled, because the AD10 signal from the 2650 is also fed to the active-low chip-select input CS1 of the 2608 ROM device. Correct strobing of the ROM is achieved by using gate G4 as an inverter to feed an OPREQ-bar signal to the CS0 input of the ROM.

Note that this simple address decoding scheme is not completely unambiguous, because the ROM is enabled whenever AD10 is 0 and the RAMs whenever it is 1. Thus the ROM strictly occupies not only the nominal range of 000-3FF, but also higher ranges such as 800-BFF. Similarly the RAMs occupy not only their nominal range 400-4FF, but also higher ranges such as 500-5FF, 600-6FF, 700-7FF, C00-CFF, D00-DFF, E00-EFF, and F00-FFF.

This ambiguity need not cause any complications, however, providing you



Using this overlay diagram you should have no problems in fitting the components to the PC board. Sockets are used for the 2650 and 2608 devices.

### LIST OF PARTS

- 1 PC board, 175 x 135mm, code 77up2
- 8 PCB terminal pins (optional)
- 1 2650 microprocessor IC
- 1 2608 ROM (with PIPBUG: code CN0035)
- 2 2112B RAMs
- 1 74123 dual monostable
- 1 74LS38 low-power Schottky buffer
- 1 BC548 or similar NPN silicon
- 1 1N914, 1N1418 or similar diodes
- 1 40-pin IC socket, PC type
- 1 24-pin IC socket, PC type

### RESISTORS

Quarter watt, 5%: 150ohms, 4 x 1k, 3 x 1.5k, 2 x 2.2k, 1 x 3.3k, 2 x 10k, 1 x 22k.  
1 47k PC type tab pot, vertical mount

### CAPACITORS

1 47pF NPO ceramic  
1 330pF NPO ceramic  
5 .047uF LV polyester  
1 0.22uF LV polyester  
1 1.5uF 35VW electro or tantalum  
Wire for links, solder, etc.

remember it and take it into account when writing your programs. All it means is that if you forget and your program tries to address these non-existent higher memory locations, it will in reality still be talking to the same ROM and RAMs!

Many small systems use this type of simple address decoding, to simplify the circuitry and reduce costs.

The third gate, G2, is used to control the read-write function selection of the RAM devices. The inputs of the gate are fed from the R-bar/W and WRP outputs of the 2650, while its output goes to the R/W-bar control inputs of the 2112 RAM devices. The R-bar/W output of the 2650 provides its read-write control signal, while the WRP output provides a write strobe pulse designed to delay writing until memory devices have stabilised.

The remaining section of the circuit is that used to provide the serial com-

munication ports, which are associated with the flag (F) output and sense (S) input of the 2650. The output port uses the remaining gate G1 as a buffer, to control a 20mA output current in response to the F output of the microprocessor. The 150-ohm resistor in series with the gate output sets the output current level, which is sufficient to drive the normal current-loop input of an ASCII teleprinter or video data terminal.

The input port circuitry uses a BC548 or similar general-purpose NPN transistor T1 to provide level translation between a standard 20mA current loop input and the S input of the microprocessor. The input circuit provides its own 20mA source, and so is suitable for direct connection to the keyboard contacts of a teleprinter, or the corresponding output terminals of a video data terminal such as that described last month.

The 1.5uF capacitor in the base circuit of T1 is to provide contact bounce suppression in the case of teleprinter keyboards, and also to provide filtering of any noise induced in the input line. The two diodes are to protect the transistor from high amplitude noise impulses.

As you can see, the complete baby system is built up on a small PC board measuring 175 x 135mm. The pattern is coded 77up2, and PC boards etched to the pattern should be available from board manufacturers by the time you read this article.

Assembly of the system on the PCB should be fairly straightforward using the overlay diagram as a guide. Note that there are a number of wire links, necessary because the board has been kept single-sided.

In view of the fact that the 2650 microprocessor chip and the 2608 ROM are both fairly expensive, and are both MOS devices, I suggest that you use sockets for them. A 40-pin socket is required for the 2650, and a 24-pin socket for the 2608, both being of the 0.6in row-

*Below is the PCB pattern, actual size for those wishing to etch their own.*

spacing DIL type. Use high quality sockets if you can, to avoid contact troubles.

The remaining ICs are probably best soldered directly into the PC board.

I suggest that you wire in all of the links first, then add the IC sockets and the resistors and capacitors. Watch the polarity of the 1.5uF tantalum electrolytic, as this could cause malfunction if it is connected the wrong way around.

Now wire in the transistor, the two diodes and the two TTL ICs (74123 and 74LS38), taking care that these are also orientated correctly. Then finally add the two RAMs, after having connected the barrel and bit of your soldering iron to the PCB supply lines to ensure that the MOS ICs won't be damaged by static charge. It is a good idea to solder the supply pins of each IC first (pins 7 and 14), so that the internal protection diodes become operational as soon as possible.

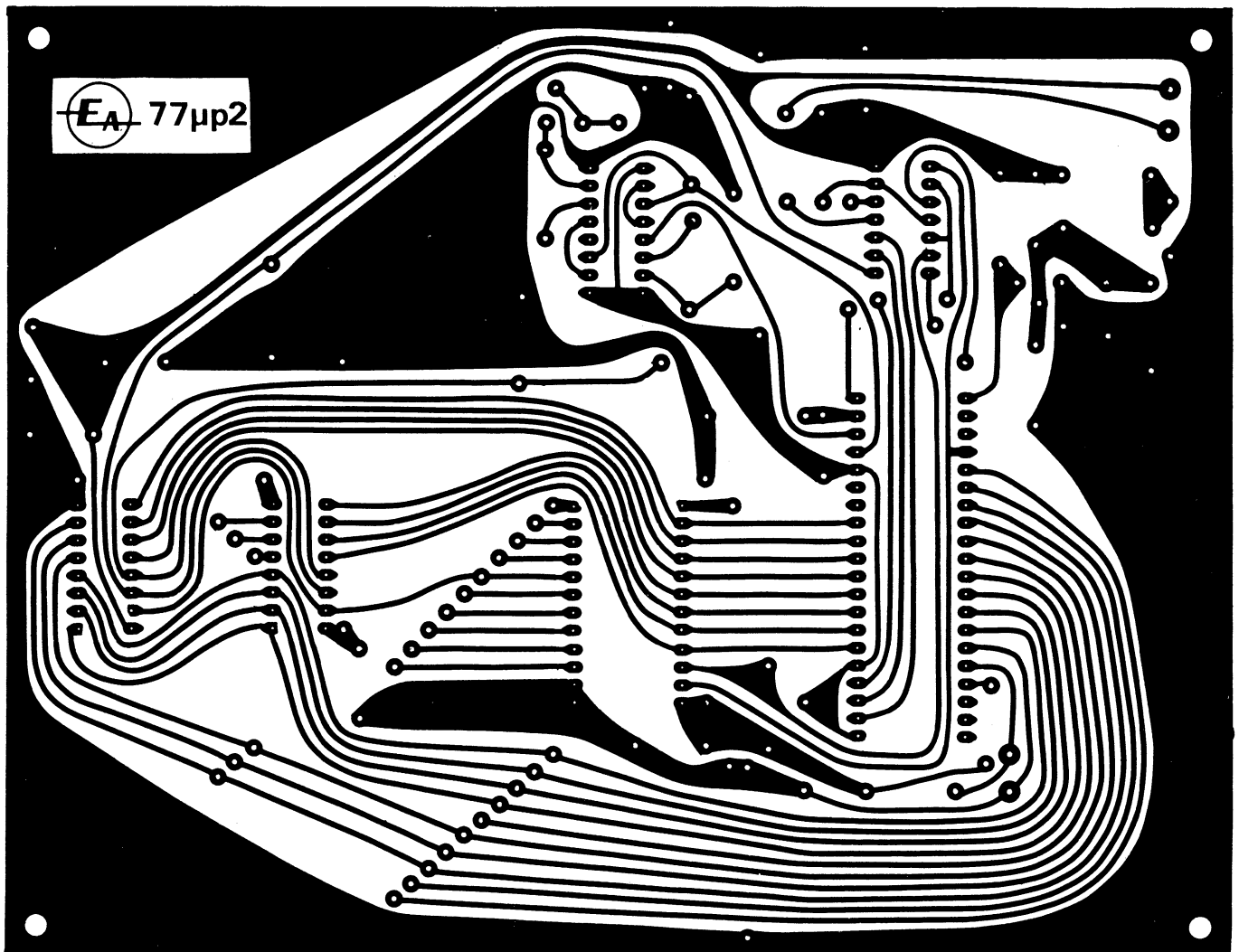
There are only eight external connections to the PC board. Two are for the power supply, which may be almost any reasonably well regulated and filtered 5V DC supply. The total current drain is around 250mA. If you don't have a suit-

able supply handy, the circuit shown in the small diagram would be quite suitable.

The four connection points adjacent to one another are for the serial input and output. These connect to the teleprinter or video data terminal, with polarities as shown. Whichever type of terminal is used, it should be connected for 20mA, full duplex operation.

The remaining two connections to the PCB go to the reset switch, which is a simple normally-closed pushbutton. When pressed, this button forces the microprocessor to reset its internal registers. Then when the button is released the microprocessor begins running from a known and predetermined state, fetching its first instruction from memory location 000—the start of the PIPBUG program residing in ROM.

The reset button therefore serves to initialise the system, and is used for this purpose both when power is first applied, and at other occasions whenever one wishes to return to PIPBUG from an applications program (apart from breakpoint returns, which take place automatically).



When you have completed wiring the PC board and connected it up to the terminal, reset switch and power supply, carefully remove the 2650 and 2608 chips from their conductive foam and plug them into their respective sockets (with the power turned off).

There is only one adjustment to be made, that in which the 74123 clock oscillator is set up to operate at the correct frequency of 1MHz. This is done with power applied.

If you have access to a frequency counter, it can be done by simply connecting the counter between pin 5 of the 74123 and the grounded negative supply rail, and adjusting the small tab pot until the counter reads 1MHz. This is the preferred way of setting the clock frequency.

However if you don't have access to a counter, the frequency can still be set up fairly accurately using the teleprinter or data terminal itself. This can be done because only when the clock frequency is the correct 1MHz will the software-timed serial output signals produced by the 2650 be at the correct 110-baud data rate required by the terminal.

To adjust the clock frequency using this method, apply power to both the system and the terminal. Then press the reset button, and release. The printer of the Teletype or the screen of the video terminal should print a couple of characters and then become static.

If by some lucky chance you have the correct clock frequency already, the printer or display screen should have displayed a carriage return (CR), a line feed (LF), and an asterisk. This is the programmed output of the PIPBUG program upon initialisation (the asterisk is its prompt signal, signifying readiness for an input command).

Most likely, you won't get this sequence of CR-LF-asterisk straight away. But the idea is to adjust the tab pot slowly and carefully from its maximum resistance extreme, pressing the reset switch after each change until you find the setting where the terminal shows that the characters are being fed to it at the correct rate.

There should be a small zone of the pot's travel in which the characters are printed correctly following the release of the reset button. For most reliable operation, try to set the pot in the middle of this zone.

With this adjustment made, your baby 2650 system is fully operational and ready to begin work (or play!). With the set of ICs, you should have received a Signetics Applications Memo (coded SS50) which explains how to use PIPBUG to feed applications programs into the system, run them, debug them, dump them on paper tape (or cassette), and re-load them. It also gives a listing of PIPBUG itself, which among other things lets you make use of some of its utility subrou-

# SIMPLE ANSWER-BACK PROGRAM FOR "BABY" 2650 MICROCOMPUTER WRITTEN BY J. ROWE, "ELECTRONICS AUSTRALIA" MAGAZINE

ADD.	CODE	MNEMONICS	COMMENTS
440	76 C0	PPSU 40	/SET TTY TO MARK
442	3F 02 36	BSTA,UN CHIN	/FETCH CHAR VIA PIPBUG RTN
445	C1	STRZ,R1	/SAVE
446	3F 02 B4	BSTA,UN COUT	/ECHO
449	01	LODZ,R1	/RESTORE IN R0
44A	A4 0D	SUBI,R0 "CR"	/TEST FOR CR
44C	58 74	BRNR,R0 -12	/KEEP GOING IF NOT
44E	04 0A	LODI,R0 "LF"	/IF YES, PROVIDE LF
450	3F 02 B4	BSTA,UN COUT	
453	05 00	LODI,R1	/SET R1=0
455	0D 24 66	LODA,R1 466+	/FETCH ANSWER CHAR
458	C3	STRZ,R3	/SAVE
459	3F 02 B4	BSTA,UN COUT	/PRINT
45C	A7 0D	SUBI,R3 "CR"	/TEST FOR CR
45E	5B 75	BRNR,R3 -11	/IF NOT, KEEP GOING
460	04 0A	LODI,R0 "LF"	/IF YES, SUPPLY LF
462	3F 02 B4	BSTA,UN COUT	
465	1B 5A	BCTR,UN -38	/BACK TO START AGAIN
467	47 4F 20		/ANSWER TEXT
46A	41 57 41		
46D	59 2C 49		
470	27 4D 20		
473	42 55 53		
476	59 21 0D		/ANSWER MUST END WITH A CR.

*This simple novelty program should help you get your system going. Only the code is actually fed into RAM, starting at location 440 hex.*

tines such as the serial input and output routines "CHIN" and "COUT".

To help you get your system up and running, a listing is shown for a modified version of the novelty answer-back program which the author originally wrote for the larger PC1001 system. Note that all you actually enter into the system are the two-digit hexadecimal machine code words; the mnemonics and comments are purely to help follow how the program works.

To feed the program into the system, you use the PIPBUG "A" command, typing first "A 440" and then a carriage return. PIPBUG will then type out on the next line "440 XX", where XX is the current content of location 440 (probably random). It then pauses. You then type "76", followed by a line feed, whereupon PIPBUG does a CR-LF, and then prints

out the next memory address and its current content. You then type "CO" and LF, and so on until all of the program has been fed in.

Then to run the program, type "G 440" followed by a CR. PIPBUG will then transfer you to the program in RAM.

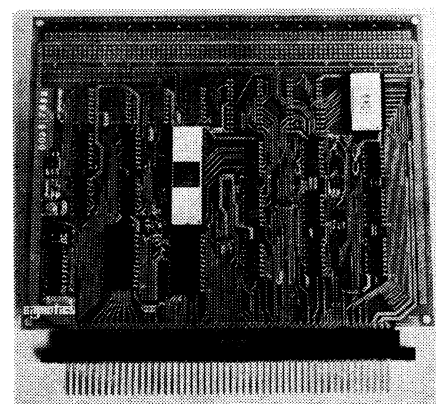
Try typing in a comment, ending it with a carriage return. The program should answer with a terse "GO AWAY, I'M BUSY!"

When you get tired of this reply, it can be changed by feeding in a new string of ASCII characters starting at address 467 hex. Note, however that the message must end with a CRR (hex 0D).

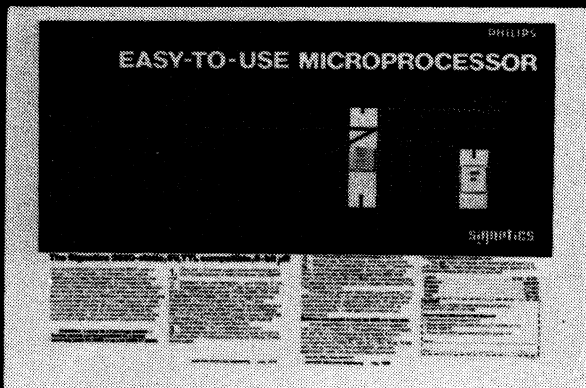
Of course this is just a demonstration program, to get you going. The next step is to write your own, using as a guide the Signetics 2650 programming book supplied with the kit. Happy computing!

## For those with a bigger system in mind:

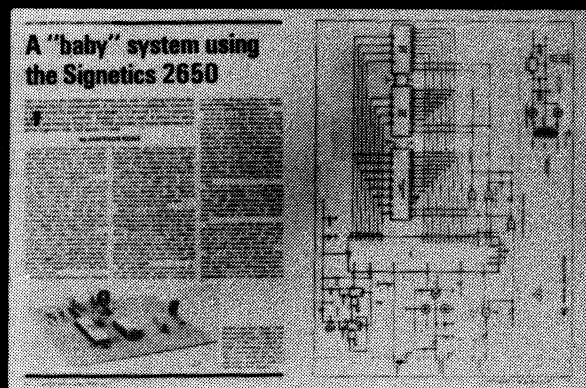
*If your ultimate aim is to build up a large 2650 system, it may be better for you to start with the single-board system shown at right, than with our "baby". Available both as an assembled system (code PC1500) and as a do-it-yourself kit (KT9500), it provides all of the features of the baby system together with full memory decoding, fully buffered data and address lines, and two bidirectional 8-bit input/output ports. Further details are available from Philips distributors.*



# SIGNETICS 2650 MICROPROCESSOR



Remember this ad?  
(last year)



Recall this feature?  
(last month)

## NOW EASY TO TRY NOW EASY TO BUY

START WITH THE INCREDIBLY SIMPLE  
**2650-KT95EA**  
"BABY MICROPROCESSOR KIT"

NOW AVAILABLE FROM THESE  
PHILIPS SIGNETICS STOCKISTS

**NEW SOUTH WALES:** Applied  
Technology, CEMA, Electronic  
Enthusiasts, ICS, Tecnico (Sydney),  
Digitronics (Newcastle),  
Macelec (Wollongong)

**ACT:** Daicom

**QUEENSLAND:** Electronic  
Components, Fred Hoe,  
NS Electronics

**VICTORIA:** CEMA, ICS, Sontron,  
Sycam, Tecnico

**SOUTH AUSTRALIA:** Gerard &  
Goodman, ICS, K-tronics, Protronics,  
World Imports

**WESTERN AUSTRALIA:**  
Atkins Carlyle, Willis Trading

**TASMANIA:** W. G. Genders,  
NS Electronics

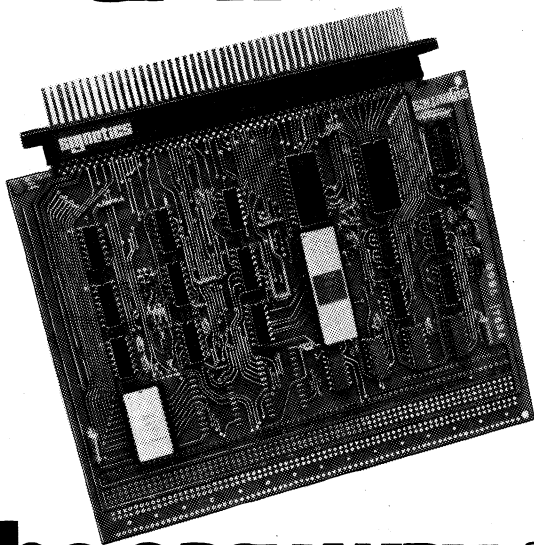


Electronic  
Components  
and Materials

# PHILIPS

153-0194

# ADAPTABLE BOARD COMPUTER PROTOTYPING CARD.



Two easy steps:  
Plug in, Hook up  
+ 5 volts, ground, and  
a teletype,...and you're  
in business!

Here is a short  
list of features.

- 1 K BYTES ROM  
(PIPBUG EDITOR  
AND LOADER) BOARD EXPANDABLE  
TO 2 K BYTES ROM/PROM
- 512 BYTES RAM (2112B – 256 x 4  
STATIC NMOS RAMS) BOARD  
EXPANDABLE TO 1 K BYTES OF RAM
- ON BOARD TTL CLOCK
- TWO – 8 BIT PARALLEL B1/D1  
I/O PORTS
- RS232/TTY SERIAL I/O PORT

**The easy way  
to start in  
Microprocessors**

● CONNECTOR  
SUPPLIED

● AVAILABLE PRE-  
ASSEMBLED AND

TESTED – 2650 PC1500  
OR IN KIT FORM  
– 2650 KT9500

Ask your local  
Philips stockist to

show you the 2650 PC1500 Adaptable  
Board Computer Prototyping Card, and  
get an easy start in Microprocessors.

PHILIPS ELECTRONIC  
COMPONENTS & MATERIALS,

P.O. Box 50, Lane Cove, 2066.

Sydney 427 0888,

Melbourne 699 0300, Brisbane 277 3332

Adelaide 223 4022, Perth 65 4199.



Electronic  
Components  
and Materials

# PHILIPS

153.0198



# Four programs for our baby

by PERRY BROWN

Courtesy Applied Technology Pty Ltd

## GUESSING GAME

LOCATION	CODE	MNEMONICS
0440	07 07	STRT LODI R3 H'07'
0442	3F 04 B6	F2 BSTA UN PRNT
0445	05 00	F1 LODI R1 H'00'
0447	E5 63	F2 COMI R1 H'63'
0449	19 7A	BCTR 'GT' F1
044B	12	SPSU
044C	E4 00	COMI RO H'00'
044E	19 02	BCTR 'GT' F4
0450	D9 75	F3 BIRR R1 F2
0452	E5 00	F4 COMI R1 H'00'
0454	18 7A	BCTR 'LT' F3
0456	77 10	PPSL H'10'
0458	07 00	LODI R3 H'00'
045A	75 10	CPSL H'10'
045C	07 36	LODI R3 H'36'
045E	3F 04 B6	F5 BSTA UN PRNT
0461	07 1F	LODI R3 H'1F'
0463	3F 04 B6	BSTA UN PRNT
0466	3B 3B	BSTR UN INPT
0468	07 09	LODI R3 H'09'
046A	C2	STRZ R2
046B	82	F6 ADDZ R2
046C	FB 7D	BDRR R3 F6
046E	C2	STRZ R2
046F	3B 32	BSTR UN INPT
0471	82	ADDZ R2
0472	77 10	PPSL H'10'
0474	87 01	ADDI R3 H'01'
0476	75 10	CPSL H'10'
0478	07 10	LODI R3 H'10'
047A	E1	COMZ R1
047B	19 61	BCTR 'GT' F5
047D	07 18	LODI R3 H'18'
047F	E1	COMZ R1
0480	1A 5C	BCTR 'LT' F5
0482	07 28	LODI R3 H'28'
0484	3B 30	BSTR UN PRNT
0486	77 10	PPSL H'10'
0488	05 30	LODI R1 H'30'
048A	A7 0A	F7 SUBI R3 H'0A'
048C	E7 00	COMI R3 H'00'
048E	1A 02	BCTR 'LT' F8
0490	D9 78	BIRR R1 F7
0492	87 3A	F8 ADDI R3 H'3A'
0494	01	LODZ R1
0496	3F 02 B4	BSTA UN COUT
0498	77 10	PPSU H'10'
049A	03	LODZ R3
049C	3F 02 B4	BSTA UN COUT
049E	07 02	LODI R3 H'02'
04A0	1F 04 42	BCTA UN FZ
04A2	3F 02 86	INPT BCTA UN CHIN
04A4	E4 30	COMI RO H'30'
04A6	1A 79	BCTR 'LT' INPT
04A8	E4 39	COMI RO H'39'
04AA	19 75	BCTR 'GT' INPT
04AC	C3	STRZ R3
04AE	3F 02 B4	BSTA UN COUT
04B0	03	LODZ R3
04B2	44 0F	ANDI R3 H'0F'
04B4	17	RET UN
04B6	0F 24 BF	PRNT LODA+R2 MSGE
04B8	E4 00	COMI RO H'00'
04BA	14	RETC '='
04BC	3F 02 B4	BSTA UN COUT
04BE	1B 75	BCTR UN PRNT
04B0	00 20 54	MSAG 'NUL SP T
04B4	52 59 53 0D	R Y S CR
04B8	0A 52 45 41	L F R E A
04BC	44 59 3F 00	D Y P NUL
04C0	0D 0A 48 49	CR LF H I
04C4	47 48 21 00	Q H I NUL
04C8	0D 0A 4C 4F	CR LF L O
04CC	57 21 00 0D	W I NUL CR
04D0	0A 47 55 45	L F G U E
04D4	53 53 2D 00	S S - NUL
04D8	0D 0A 59 45	CR LF Y E
04DC	53 21 20 41	S I SP A
04E0	46 54 45 52	F T E R
04E4	20 00 0D 0A	SP NUL CR LF
04E8	4F 4B 20 31	O K SP 1
04EC	2D 39 39 00	- 9 9 NUL

When called, the program will wait until you enter any character. It will then generate a random number between 1 and 99, which you must guess. Starting address is 0440.

## NIM

LOCATION	CODE	MNEMONICS
0440	05 17	STRT LODI R1 H'17'
0442	07 29	LODI R3 H'29'
0444	3F 04 B3	BSTA UN PRNT
0447	1B 0E	BCTR UN F1
0449	07 09	LODI R3 H'09'
044B	3F 04 B3	BSTA UN PRNT
044E	01	LODZ R1
044F	A2	SUBZ R2
0450	C1	STRZ R1
0451	02	LODZ R2
0452	64 30	IORI RO H'30'
0454	3F 02 B4	BSTA UN COUT
0457	07 1E	F1 LODI R3 H'1E'
0459	3F 04 B3	BSTA UN PRNT
045C	01	LODZ R1
045D	C3	STRZ R3
045E	04 30	LODI RO H'30'
0460	A7 0A	F2 SUBI R3 H'0A'
0462	1A 02	BCTR 'LT' F3
0464	D6 7A	BIRR RO F2
0466	87 3A	F3 ADDI R3 H'3A'
0468	3F 02 B4	BSTA UN COUT
046B	03	LODZ R3
046C	3F 02 B4	BSTA UN COUT
046F	E5 01	COMI R1 H'01'
0471	19 07	BCTR 'GT' F4
0473	07 00	LODI R3 H'00'
0475	3B 3C	BSTR UN PRNT
0477	1F 04 40	BCTA UN STRT
047A	07 16	F4 LODI R3 H'16'
047C	3B 35	BSTR UN PRNT
047E	3F 02 86	F5 BSTA UN CHIN
0481	E4 31	COMI RO H'31'
0483	1A 79	BCTR 'LT' F5
0485	E4 33	COMI RO H'33'
0487	19 75	BCTR 'GT' F5
0489	C3	STRZ R3
048A	47 0F	ANDI R3 H'0F'
048C	3F 02 B4	BSTA UN COUT
048F	01	LODZ R1
0490	A3	SUBZ R3
0491	C1	STRZ R1
0492	E5 01	COMI R1 H'01'
0494	19 07	BCTR 'GT' F6
0496	07 3B	LODI R3 H'3B'
0498	3B 19	BSTR UN PRNT
049A	1F 04 40	F6 BCTA UN STRT
049D	44 05	SUBI RO H'05'
049F	1A 0E	BCTR 'LT' F7
04A1	44 03	F8 SUBI RO H'03'
04A3	19 7C	BCTR 'GT' F8
04A5	84 03	ADDI RO H'D3'
04A7	C2	FA STRZ R2
04A8	98 02	BCTR 'LT' F9
04AA	86 01	ADDI R2 H'01'
04AC	1F 04 49	F9 BCTA UN FR
04AF	84 04	F8 ADDI RO H'04'
04B1	1B 74	BCTR UN FA
04B3	0F 24 BA	PRNT LODA+R3 MSGE
04B6	14	RETC '='
04B7	3F 02 B4	BSTA UN COUT
04BA	1B 77	BCTR UN PRNT
04BC	0D 0A 49	MSGE 'CR LF I
04BF	20 57 49 4E	SP W I N
04C3	00 0D 0A 49	NUL CR LF I
04C7	27 4C 4C 20	I L L SP
04CB	54 41 4B 45	T A K E
04CF	20 00 0D 0A	SP NUL CR LF
04D3	4D 4F 56 45	H O V E
04D7	2D 00 0D 0A	- NUL CR LF
04DB	4E 4F 20 4C	M O SP L
04DF	45 46 54 3D	E F T =
04E3	00 0D 0A 0A	NUL CR LF LF
04E7	4C 45 41 56	L E A V
04EB	45 20 31 20	H SP 1 SP
04EF	54 4F 20 57	T O SP W
04F3	49 4E 00 0D	I N NUL CR
04F7	0A 59 4F 55	L F Y O U
04FB	20 57 49 4E	SP W I N
04FF	00	NUL

The game of Nim: starting with 23 you and the program take turns at subtracting a number from 1 to 3. The one that leaves 1 after their move wins. Starting address is 0440.

## MATHS DEMONSTRATION

LOCATION	CODE	MNEMONICS
0440	04 83	STRT LODI RO 'ADDZ R3'
0442	C8 1C	STRR RO MOD
0444	3F 04 B4	BSTA UN SUB 1
0447	3F 02 86	F1 BSTA UN CHIN
044A	E4 2B	COMI RO '+'
044C	18 0E	BCTR '=' PLUS
044E	E4 2D	COMI RU '-'
0450	18 06	BCTR '=' SUBT
0452	E4 2A	COMI RO '='
0454	18 38	BCTR '=' MULT
0456	1B 6F	BCTR UN F1
0458	05 A3	LODI R1 'SUBZ R3'
045A	C9 04	STRR R1 MOD
045C	3F 04 AF	BSTA UN SUB2
045F	02	LODZ R2
0460	82	ADDZ R3
0461	30 00	MOD COMI RO H'00'
0463	5A 09	BCTR 'LT' F2
0465	03	LODZ R3
0466	A2	SUBZ R3
0467	07 2D	LODI R3 '='
0469	CF 04 DE	STRR R3 A1
046C	1B 0B	BCTR UN F3
046E	E4 0A	F2 COMI RO H'0A'
0470	1A 0F	BCTR 'LT' F3
0472	06 31	LODI R2 H'01'
0474	CE 04 DE	STRR R2 A1
0477	A4 0A	SUBI RO H'0A'
0479	64 30	IORI RO H'30'
047B	CC 04 DF	STRR RO A2
047E	1F 04 C9	BCTA UN END
0481	64 30	F3 IORI RO H'30'
0483	CC 04 DE	STRR RO A1
0486	0A 00	LODI RO H'00'
0488	CC 04 DF	STRR RO A2
048B	1F 04 C9	BCTA UN END
048E	3B 1F	BSTR UN SUB2
0490	0A 00	LODI RO H'00'
0492	E6 00	F5 COMI R2 H'00'
0494	18 03	BCTR 'LT' F4
0496	83	ADDZ R3
0499	FA 79	BDRR R2 F5
049B	06 00	F4 LODI R2 H'00'
049D	E4 0A	F6 COMI RO H'0A'
049F	1A 06	BCTR 'LT' F7
04A1	86 01	ADDI R2 H'01'
04A3	44 0A	SUBI RO H'0A'
04A5	1B 76	BCTR UN F6
04A7	66 30	F7 IORI R2 H'30'
04A9	0A 35	STRR R2 A1
04AB	64 30	IORI RO H'30'
04AD	C8 32	STRR RO A2
04AF	1B 1A	BCTR UN END
04B1	3F 02 B4	SUB2 BSTA UN COUT
04B2	03	LODZ R3
04B3	C2	STRZ R2
04B4	3F 02 86	SUB1 BSTA UN CHIN
04B5	75 0A	CPSL H'0A'
04B6	E4 30	COMI RO H'30'
04B7	3A 77	BCTR 'LT' SUB1
04B8	E4 39	COMI RO H'39'
04B9	39 73	BCTR 'GT' SUB1
04CA	C3	STRZ R3
04CB	3F 02 B4	BSTA UN COUT
04CC	47 0F	ANDI R3 H'0F'
04CD	C0	STRZ R3
04CE	17	RETC UN
04CF	05 00	END LODI R1 H'00'
04D0	0D 24 DC	F8 LODA R1+ MSGE
04D1	E4 00	COMI RO H'00'
04D2	18 05	BCTR '=' PA
04D3	3F 02 B4	BSTA UN COUT
04D4	1B 74	BCTR UN F8
04D5	3F 00 8A	FA BSTA UN CR LF
04D6	1F 04 4D	BCTA UN STRT
04D7	3D	MSAR '='
04D8	00	A1
04D9	00	A2
04DA	00	A3

This program provides the functions of a simple 3-function calculator. It will multiply, add or subtract two single digit decimal numbers. Normal plus, minus and equals signs are used, with an asterisk symbol for multiplication. Starts at 0440.



## MESSAGE EDITOR

LOCATION	CODE	MEMONICS
0440	3F 00 8A	F1 BSTA UN CRLF
0443	04 3E	LODI R0 'GT'
0445	3F 02 B4	BSTA UN COUNT
0448	3F 02 86	BSTA UN CHIN
044B	01	STRZ R1
044C	3F 02 B4	BSTA UN COUNT
044F	3F 00 8A	BSTA UN CRLF
0452	E5 54	COMI R1 'T'
0454	18 0A	BCTR 'M' FT
0456	E5 52	COMI R1 'R'
0458	18 39	BCTR 'M' FR
045A	E5 43	COMI R1 'C'
045C	18 2D	BCTR 'M' FC
045E	1B 60	BCTR UN F1
0460	05 00	F2 LODI R1 H'00'
0462	3F 02 86	F2 BSTA UN CHIN
0465	CD 64 B4	STRA (REL)R1 TEXT
0468	E4 1B	COMI R0 'ESC'
046A	18 54	BCTR 'M' F1
046C	E4 7F	COMI R0 'DEL'
046E	98 06	BCTR 'M' FP
0470	A5 02	SUBI R1 H'02'
0472	07 00	LODI R3 M1
0474	3B 17	BSTR UN PRINT
0476	3F 02 B4	FP BSTA UN COUNT
0479	E5 4E	COMI R1 LIMIT
047B	1A 0C	BCTR 'LT' F2
047D	07 04	LODI R3 M2
047F	3B 0C	BCTR UN PRINT
0481	04 1B	LODI R0 'ESC'
0483	CD 64 B4	STRA (REL)R1 TEXT
0486	1F 04 40	BCTR UN F1
0489	D9 37	F2 BSTR R1 F2
048B	07 0A	FC LODI R3 H'0A'
048D	06 00	PRINT LODI R2 H'00'
048F	CA 17	STRZ R2 F3
0491	1B 06	BCTR UN F4
0493	06 70	FR LODI R2 H'70'
0495	07 0A	LODI R3 H'0A'
0497	CA 0F	STRZ R2 F3
0499	0F 24 A9	F4 LODA +R3
049C	E4 1B	COMI R0 'ESC'
049E	18 05	BCTR 'M' F5
04A0	3F 02 B4	BSTA UN COUNT
04A3	1B 74	BCTR UN F4
04A5	07 0A	LODI R3 H'0A'
04A7	1B 00	BCTR UN F4 **
04A9	17	RETC UN
04AA	08	M1 'BS'
04AB	20	'SP'
04AC	08	'BS'
04AD	1B	'ESC'
04AE	0D	M2 'CR'
04AF	0A	'LF'
04B0	5B	'('
04B1	46	'P'
04B2	5D	' '
04B3	1B	'ESC'
04B4	00	TEXT

\*THIS SETS MAX TEXT LENGTH

\*\*F3 IS SECOND BYTE OF INSTR.

Upon being called, this program will give a prompt character, and await a command character. Command T allows a message to be entered, C allows a stored message to be checked, and R allows it to be repeated until the CPU is reset. In text input mode, the Del character acts as a destructive backspace for correcting errors. To return to command mode, type an ESC.

If the message being stored is too long for the buffer, an F will be displayed. Starts at 0440.

# 2650 HOME COMPUTER SELECTION GAME

DO YOU  
WANT TO BUILD  
A POWERFUL,  
LOW COST  
HOME COMPUTER

START HERE

## LOW COST TV TERMINAL

A MUST TO GET THE MOST OUT OF YOUR  
SYSTEM — INCLUDES AUTOMATIC SCROLLING,  
CURSOR, FULL EDIT FACILITIES

ETI 632 VDU PLUG IN VERSION ..... \$180.00  
632 ECONOMY VERSION..... \$135.00

## CASSETTE INTERFACE

For Low Cost  
Bulk Storage

CT750 (Assembled) \$37.50

Radio Electronics low cost  
kit with test tape and  
instructions \$22.50

LIMITED  
QUANTITY  
ONLY

## MEMORY

4K STATIC RAM KIT ..... \$100.00  
1K RAM STICKS ..... \$ 25.50

★ YOU WIN !! ★

YOU NOW HAVE A MOST  
COST EFFECTIVE HOME  
COMPUTER SYSTEM

2650 HOME STUDY  
PROGRAMMING COURSE  
CASSETTES AND WORKSHOP  
NOTES

Available soon

## SELECT A BASIC SYSTEM

- A. EA BABY 2650 \$75.00
- B. KT 9500 COMPLETE \$199.00
- C. EA 2650/KT9500  
CONVERSION \$142.00

## POWER SUPPLY DECISION

+5, -12V

- A. 1A LCPS KIT \$15.00
- B. AT 512 ASSEMBLED  
UNIT -12 \$27.50
- C. AT 1250 HI-CURRENT \$47.50

## SOFTWARE GAMES PACK

TAPE WITH LISTINGS \$12.50

ASTRO TREK, HANGMAN, NIM,  
POKER MACHINE, MASTERMIND ETC

## 2650 USERS GROUP

INITIAL MEMBERSHIP INCLUDES  
PROFESSIONAL DOCUMENTATION  
PACKAGE CONTAINING A MOST  
COMPREHENSIVE SOFTWARE  
LISTINGS INCLUDING

TEXT EDITOR  
ASSEMBLER  
BLOCK MOVE  
ASTRO TREK  
BAUDOT ROUTINES  
VARIOUS GAMES  
SORTING ROUTINES  
MATHEMATICS PACK-  
AGES ETC, ETC \$40.00

ALSO INCLUDES UP DATE  
SERVICE

NEED MORE INFO send  
\$1.00 and return address  
for full details



**APPLIED  
TECHNOLOGY  
PTY. LTD.**

APPLIED TECHNOLOGY P/L  
109-111 Hunter St., Hornsby 2077.  
Phone: 476 4758, 476 3759  
Hours: Mon — Fri. 9 — 5, Sat All Day.

# A low cost video display unit

Here is a new design for a low cost video display unit, capable of displaying data from a microcomputer on a standard TV receiver or monitor. It displays 16 lines of 32 characters and offers both flashing cursor and a destructive backspace facility. All timing is derived from a crystal oscillator, and no setting up is required.

by **MICHAEL O'NEILL**

Physics Department,  
Newcastle University

This Video Display Unit (VDU) was designed primarily for the microprocessor system user, who requires a video terminal of minimum complexity to enable him to communicate with his system. Therefore many of the unnecessary features of commercial style VDU's were abandoned in order to provide a cheap but effective video terminal for such applications.

Sixteen lines of 32 characters was selected as the screen format which allows for adequate display of program steps. With continuous roll-up facility, the user can see at least his last 16 lines of information. The cursor, indicating the position of the next character is fix-

ed permanently on the bottom line (line 16). Carriage return and line feed (non-print characters) are decoded and these are normally all that would be required for a basic unit. However, a back space control function has been included mainly for the benefit of those who might use such a unit as a TV typewriter. This control allows editing of the bottom line before a line feed is given. Back space actually types a space in the location of the cursor after moving it back one character position.

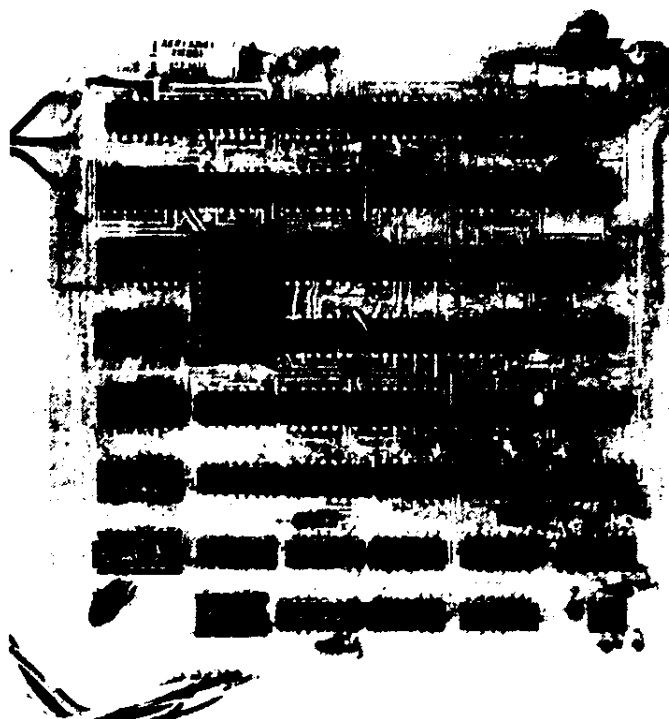
The VDU uses all standard readily available TTL IC chips, except for six CMOS memory chips and the character generator chip.

The method of actually displaying a

character on a TV screen will not be described in detail here, as reference to the issue of EA for January 1977 should make this clear. The VDU described here uses the same character generator IC described in the earlier article (i.e., the 2513), and hence allows for the display of the full 64-character subset of ASCII known as "6-bit ASCII". This is the same character set displayed on most teleprinters.

A 4.7MHz crystal oscillator provides all of the clock pulses for the VDU. As can be seen from the block and circuit diagrams, this base frequency is divided down to produce the horizontal and vertical sync pulses required by the TV set. The 4.7MHz signal is also used to clock the output shift register used to convert the parallel "row data" from the character generator into the serial data required as video information by the TV display.

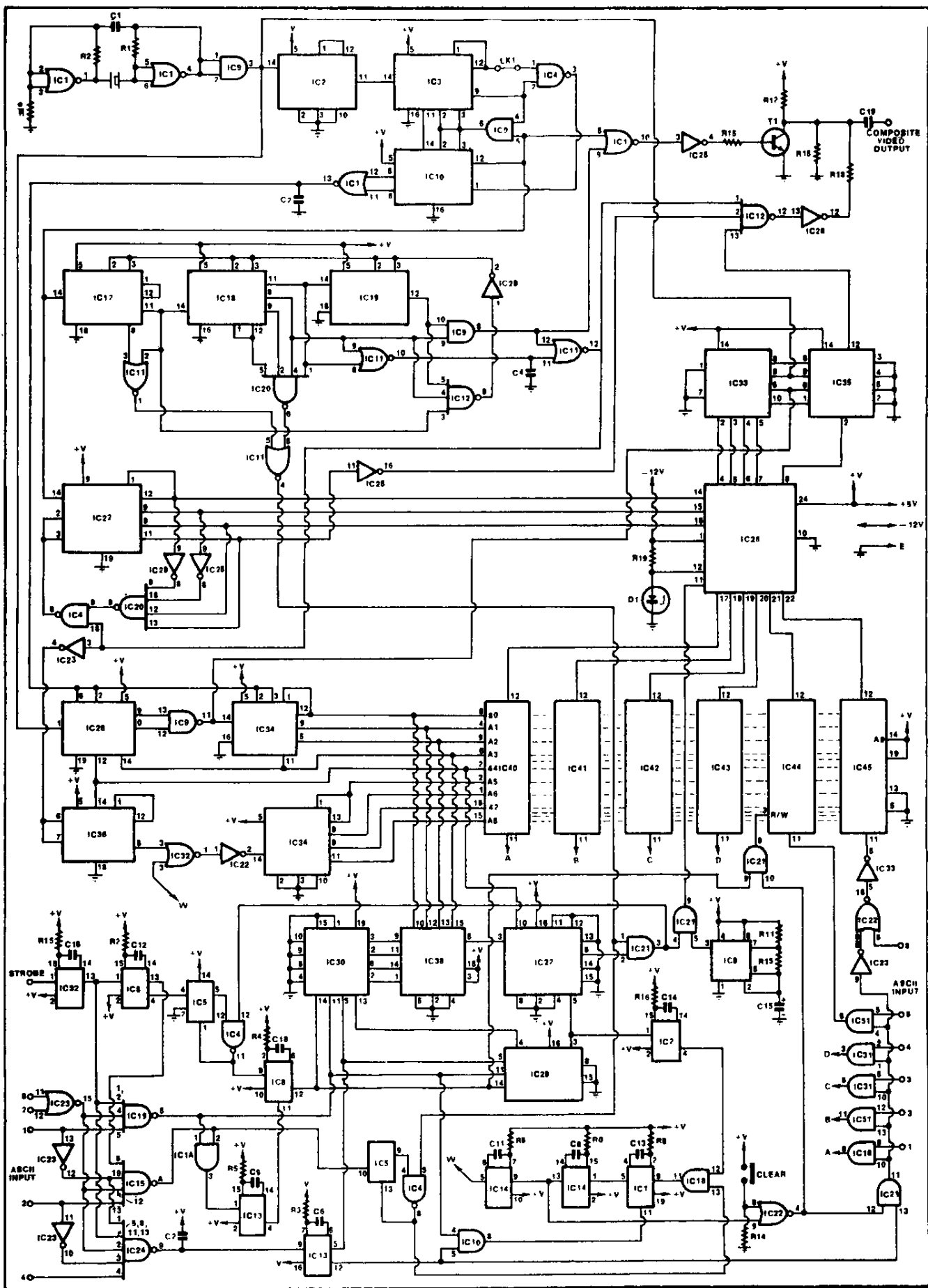
Incidentally it has been found that a 4.43MHz crystal of the type used in the subcarrier oscillator of colour TV receivers may be used instead of the nominal 4.7MHz crystal. This can be worthwhile, as the 4.43MHz crystal is generally cheaper and easier to obtain. Naturally when the lower frequency



## SPECIFICATION

VDU displays the 6-bit ASCII character set, in 16 lines of 32 characters. All timing derived from a crystal-locked oscillator; no setting up required. Continuous line scrolling of display. Maximum input data rate 50 characters/sec. Destructive back space facility for editing. Flashing cursor indicates next character position. Uses standard TTL ICs for low cost.

At left is the assembled PC board. Note that the version shown here uses a 100pf capacitor paralleled by a 30pf trimmer in place of the crystal.



## Video display unit

crystal is used, both of the TV sync pulse frequencies are lower also, but most TV sets seem to be able to lock onto them quite easily. As the vertical frequency becomes 45.5Hz instead of 50Hz, some sets may produce a small amount of horizontal wavering or "snaking", particularly if there is some 50Hz ripple getting into the vertical oscillator from the receiver's power supply.

If such an effect is experienced and found annoying, then a 100pF capacitor with a 30pF trimmer in parallel may be substituted for the crystal if a 4.7MHz crystal is not available. The trimmer capacitor can be varied until the TV set locks onto the VDU sync pulses. Further trimming may be required to obtain a steady display.

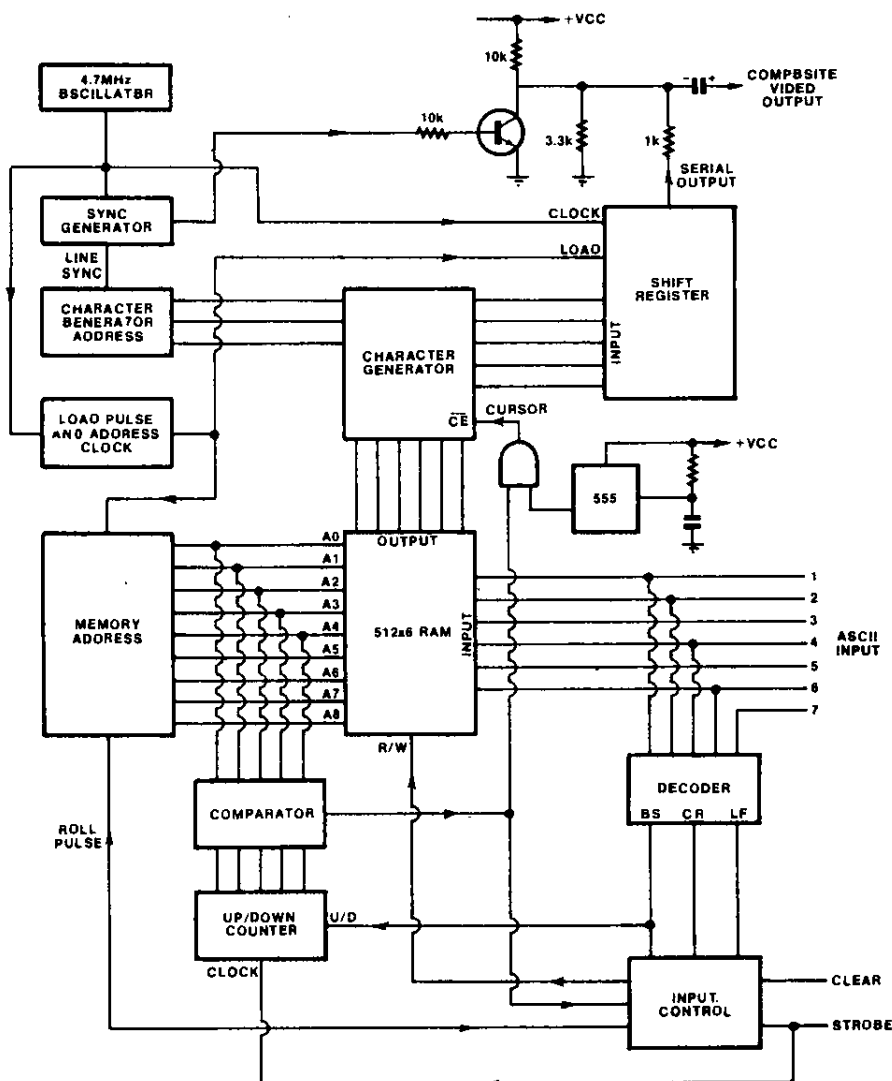
Note that if this capacitor is used, a 220 ohm resistor is required as an addition between pins 2 and 3 (joined together) on IC1 and ground. This is indicated on the circuit diagram as R\* and can be soldered onto the board, vertically, from the appropriate side of R2 and the outer ground line.

A further chain of frequency dividers generates the line address information for the character generator and the load pulses for the shift register. One load pulse occurs for every six clock pulses given to the shift register, thus loading it with the required five bits of data for a character row and also giving a single "dot" space between characters. Since this load pulse occurs for each character across the screen (i.e. 32 times for each horizontal TV scan) it becomes the ideal clock pulse for the memory address.

Nine address lines are required to address the memory, which holds each character to be displayed on the screen in its ASCII code. The memory is re-addressed each frame and therefore a character remains in its particular location in memory until changed by an external control signal. The memory consists of six 2102, 1k x 1 RAMs, six being required to hold the six bit ASCII code. This provides 1024 6 bit words, but only 512 are used.

The outputs of the memories are connected directly to the character generator. The memories are normally held in the read mode and each time the address changes, the outputs from the memories change to provide a six bit ASCII code for the character generator.

As just mentioned, an external control signal is required to change data held in memory. To do this we must have a written command, together with an indication as to where in memory its contents are going to change. Memory location indication is achieved by com-



Above shows the block diagram, while at right is the component overlay pattern.

parators and a set of counters that duplicates the memory address. This extra set of counters are advanced one count by the input strobe pulse, which indicates that a new character is being entered either from a keyboard or a computer. The comparator gives an output when the memory address equals the count on the duplicated set of counters, and this output is used to gate the ASCII input into the correct location in memory.

Because of this gating technique, a character can only be written into memory every frame, which immediately indicates a baud rate limitation of 500 baud. Since this VDU was designed for microprocessors, this modest baud rate should not be a problem as the VDU will operate at the 110, 150 or 300 baud rates used by most debug ROMs in microcomputers.

If the output of the comparators is fed to the CE-bar input on the character generator chip, it disables the chip for that particular location, and therefore a single bar is generated on the screen. This occurs instead of

generating a character and therefore a cursor appears. Since the cursor appears permanently on the 16th line, only five of the nine address lines need to be compared, thus controlling the 32 positions along the line. The blinking effect of the cursor is achieved by gating the control signal with a low frequency astable multivibrator; a 555 timer has been used for this purpose.

At this point it should be clear that we now have a "page" of information displayed on the screen with a cursor indicating the next character position. Let's now take a look at how the scrolling of lines is achieved.

The memory address counters can be divided into two parts. The first five address lines control the 32 characters across each of the 16 lines, while the 16 lines themselves are controlled by the last four address lines. If at any time an extra clock pulse is given to this last address counter it would add an extra count and thus change the character line position as they appear on the screen. If the pulse is applied to this counter during the time that there is no

# PARTS LIST

## INTEGRATED CIRCUITS

IC1 7402	IC16 7408	IC31 7408
IC2 7493	IC17 7493	IC32 74123
IC3 7493	IC18 7493	IC33 7495
IC4 7400	IC19 7493	IC34 7493
IC5 7474	IC20 7420	IC35 2513
IC6 74123	IC21 7408	IC36 7492
IC7 74123	IC22 7402	IC37 7485
IC8 555	IC23 7404	IC38 7485
IC9 7408	IC24 7430	IC39 7493
IC10 7493	IC25 7495	IC40 2102
IC11 7402	IC26 7492	IC41 2102
IC12 7410	IC27 7493	IC42 2102
IC13 74123	IC28 7404	IC43 2102
IC14 74123	IC29 74191	IC44 2102
IC15 7420	IC30 74191	IC45 2102

## RESISTORS

R1 470 ohms	R11 10k
R2 470 ohms	R12 120k
R3 22k	R13 10k
R4 6.8k	R14 470 ohms
R5 22k	R15 10k
R6 6.8k	R16 3.3k
R7 6.8k	R17 10k
R8 39k	R18 1k
R9 22k	R19 680 ohms
R10 6.8k	R* 220 ohms

## CAPACITORS

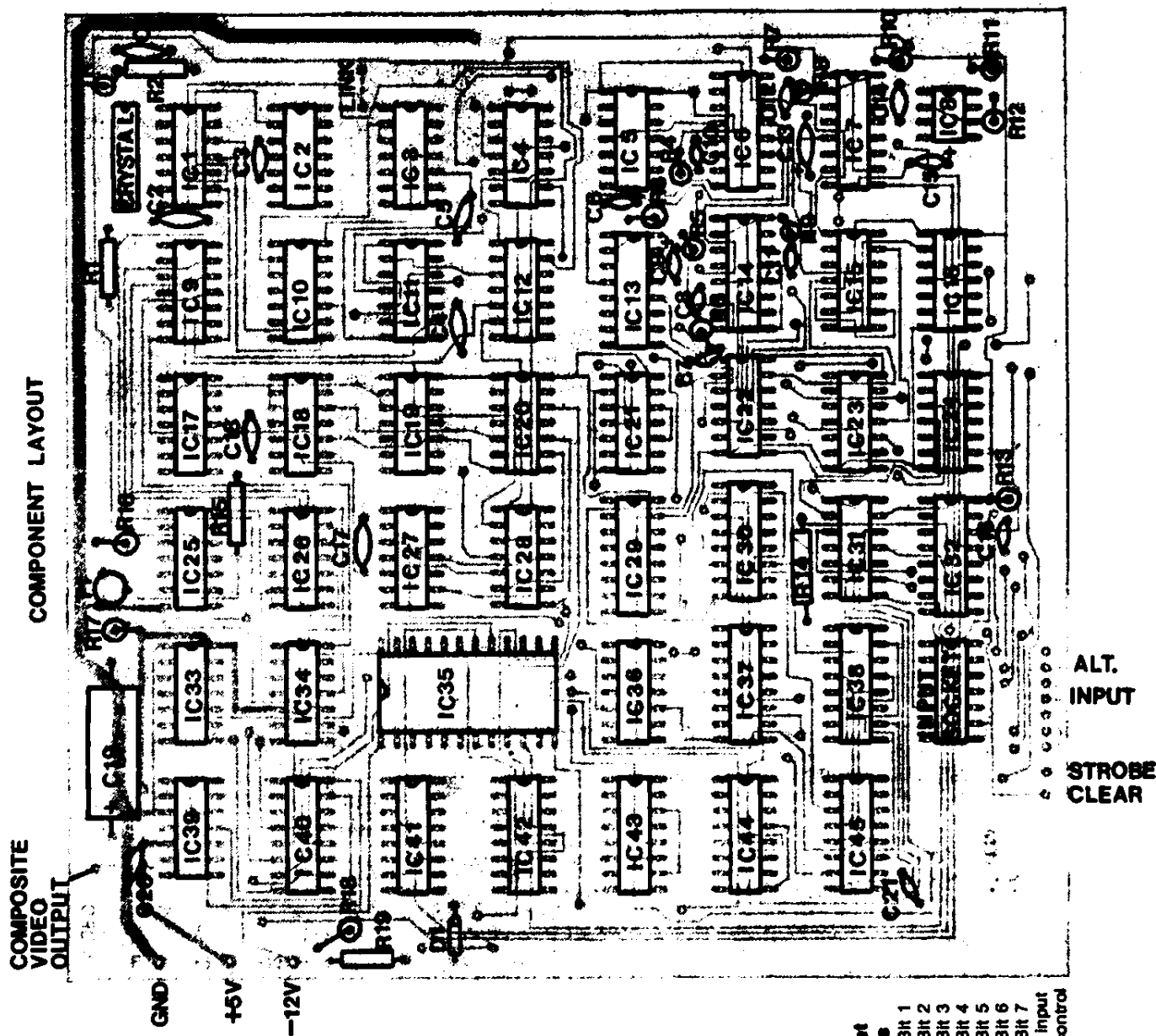
C1 .01uF	C12 330pF
C2 330pF	C13 .33uF tant.
C3 1uF	C14 .022uF
C4 330pF	C15 4.7uF tant.
C5 1uF	C16 .027uF
C6 3.3uF tant.	C17 1uF
C7 .001uF	C18 1uF
C8 .022uF	C19 47uF 25V electro.
C9 3.3uF tant.	C20 1uF
C10 22pF	C21 1uF
C11 .002uF	

CRYSTAL 4.7MHz

or 100pF capacitor and 30pF trimmer

D1 BZX79C5V1

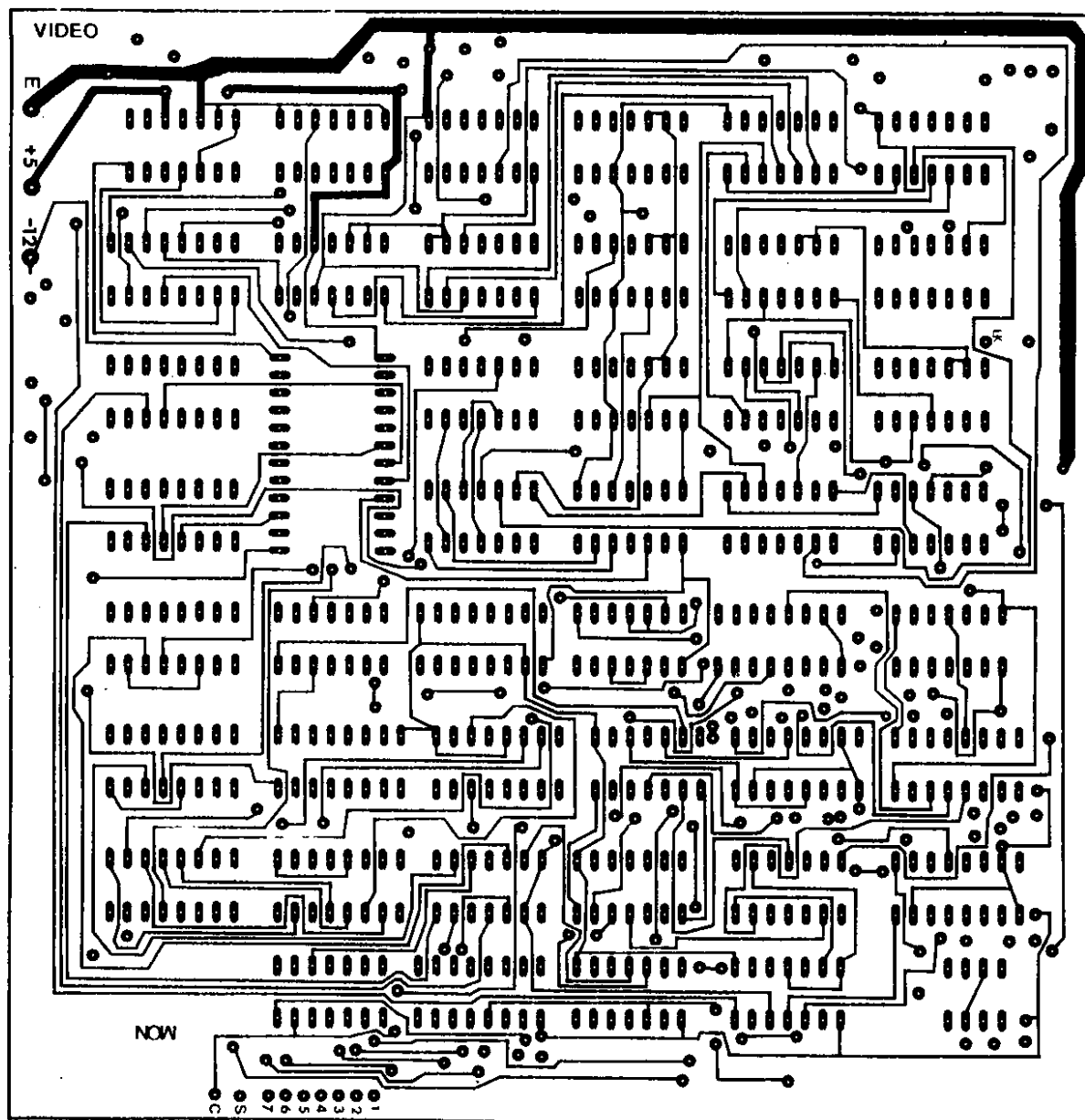
T1 BC109



## Input Socket Connections

- Pin 1 ASCII Bit 1
- Pin 2 ASCII Bit 2
- Pin 3 ASCII Bit 3
- Pin 4 ASCII Bit 4
- Pin 5 ASCII Bit 5
- Pin 6 ASCII Bit 6
- Pin 7 ASCII Bit 7
- Pin 8 Strobe input
- Pin 9 Clear control

## Low cost video display unit



Actual size reproduction of the PC pattern on the component side of the board.

display on the screen (i.e. the time between frames) then the next time a frame appears on the screen it will start at one extra character line due to this extra count. This extra clock pulse is generated at the end of a line, or when line feed is detected, and gives the scrolling effect.

When roll-up does occur another pulse is also generated which applies the ASCII code for a space to the memories and a write command is given at the same time. This immediately gives a clear line on line 16, to type onto after the previous line is rolled up.

A decoder is used to detect when carriage return, line feed or back space information is given to the VDU. The

control bit in the ASCII code — bit 7, is used for this purpose.

The video information from the shift register is fed to the output of transistor T1 via a 1k resistor, and is mixed with the inverted sync pulses which are applied to the base of the transistor. The 10k and 3.3k resistors provide the correct 1:3 ratio for sync and video information. This composite video is then output via an isolating capacitor and is suitable for applying to any video amplifier employed in standard TV sets.

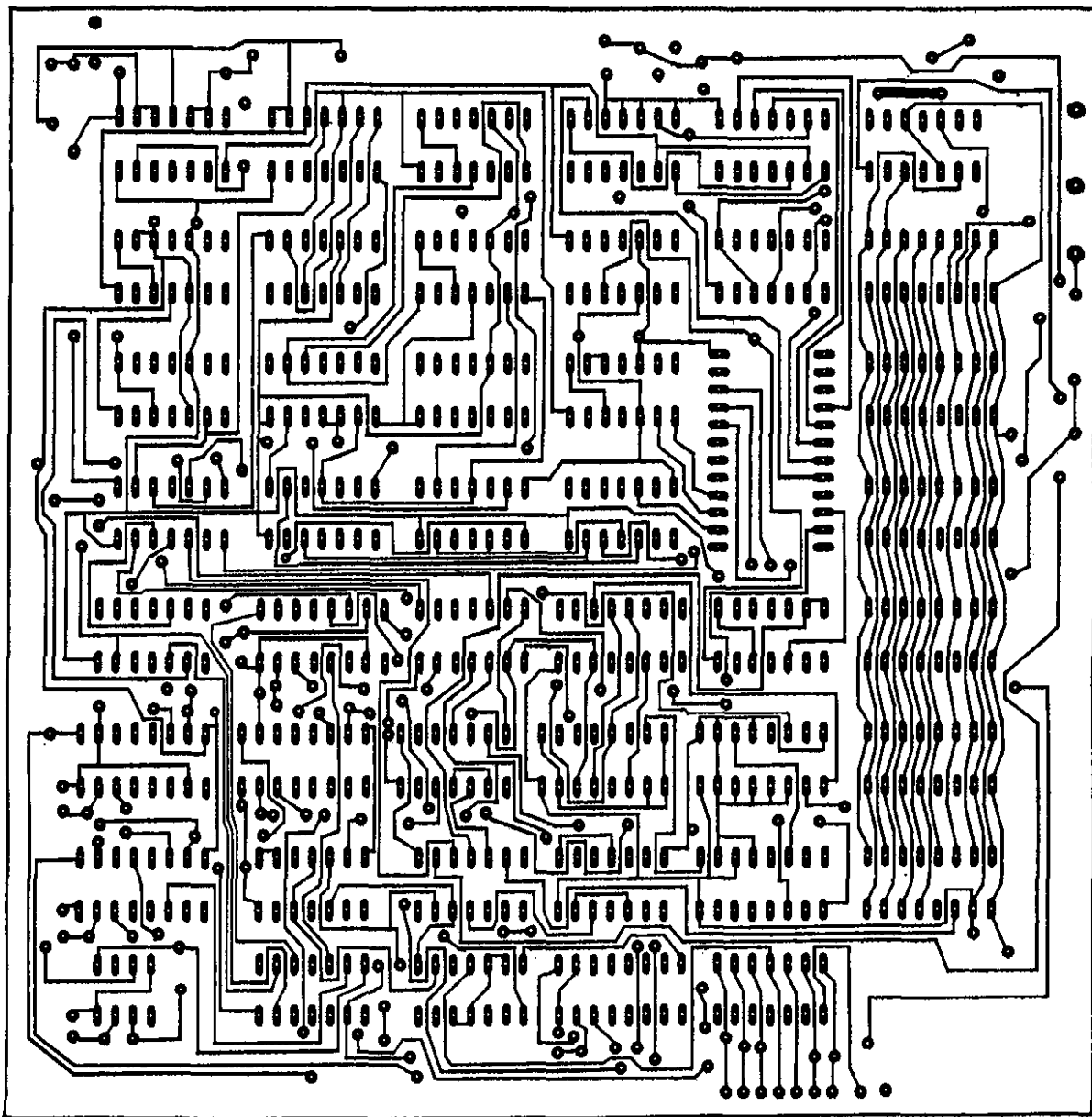
Experience has shown that the video output from the VDU is suitable for applying to the grid or the base (depending on whether valve or solid state) of the video driver in a TV set

without any alteration or disconnection of any components.

When checking for this input, one should ensure that the take-off for the sync separator is after this stage of amplification in the TV receiver.

There is absolutely no setting up required with the VDU. Random characters should appear on the TV screen as soon as power is switched on. To enable a clear screen when first turned on, a clear input has been provided on the PC board. It requires a switch to the +5V rail, or a logic "1" applied to it. This can be obtained from an unused key on the terminal's keyboard giving manual clearing, or alternatively by means of a capacitor to

## Low cost video display unit



*The PC pattern for the reverse side of the board, again shown actual size.*

the +5V supply rail, to give automatic clearing on power-up. A 47 $\mu$ F tantalum should work.

A link, LK, has been provided on the PC board to provide an option regarding horizontal positioning of the VDU display. With the link out, the video information is generated in the centre of the period between horizontal sync pulses, giving a display which should be centred on most TV sets. If, however, it is found that the display is not in the centre of your TV screen, this link can be inserted and the whole picture will be shifted about three character widths to the right of the screen.

The printed circuit board for the

VDU measures 155 x 160 mm and has an input socket facility where the required input data lines can be entered via a 14 pin DIP connector, using flat ribbon cable. This makes for a very neat connection. However, for those wishing to keep costs down, the same inputs are available at the edge of the PC board where wires can be soldered directly to the copper. The strobe input is triggered by a negative edge; if this is not available, an inverter on this line would be required.

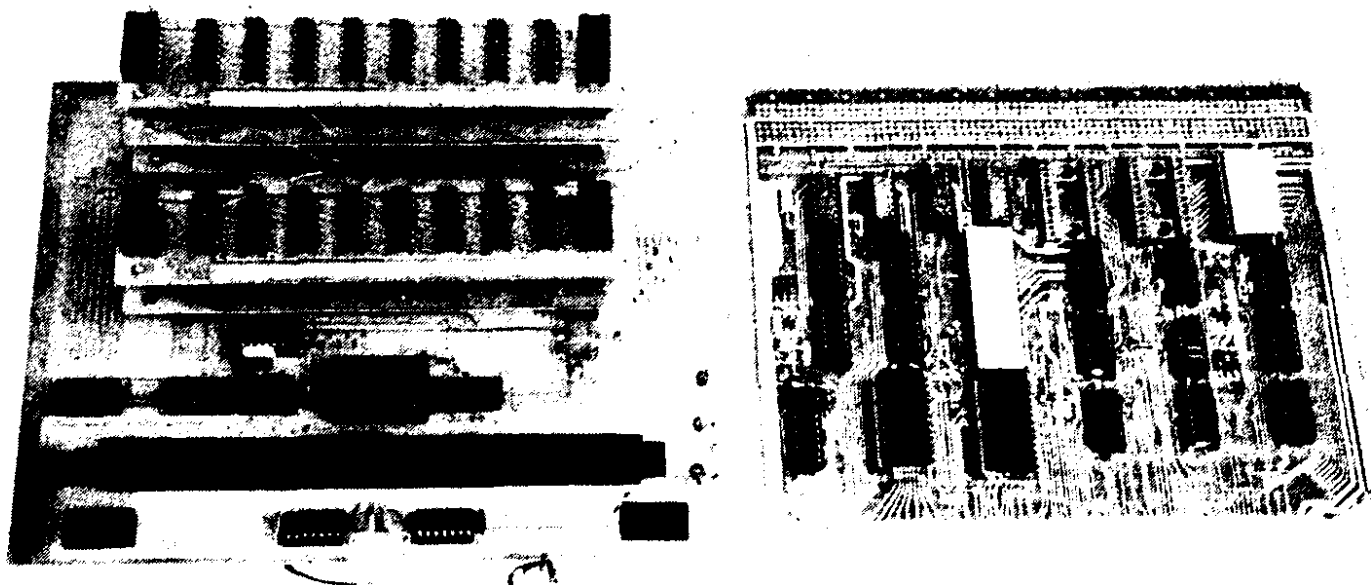
Power supply requirements are +5 Volts at 1.2 Amps and -12 Volts at around 40mA. The higher +5V supply current is required because of the TTL chips used. Three terminal regulators

rated for 1.5 Amps are adequate for this voltage supply.

A UART has not been included on the PC board because the VDU was considered to be a separate self-contained control system which accepts parallel data only, and if serial data is required by a microcomputer system then an external device such as a UART should be added. Parallel data is also acceptable to some microprocessors and makes for easier programming.

**Editor's Note:** For those who do wish to add serial interfacing and a keyboard, to produce a complete self-contained terminal, we hope to supply the necessary information shortly. ②





# Easy expansion kit for 2650 microcomputers

Many microcomputer enthusiasts have shown interest in building up medium-scale systems based on the Signetics 2650 microprocessor. This can be done quite easily and at surprisingly moderate cost, by combining the Signetics KT9500 evaluation kit with the "RAM-stick" and motherboard system which has been developed by the local firm Applied Technology.

by JAMIESON ROWE

Computer hobbyists in Australia are currently showing a lot of interest in systems based on the Signetics 2650 microprocessor. I believe one reason for this was EA's "baby" 2650 system, which I described in the March 1977 issue. This provided a really simple and low cost way of getting the 2650 "up and running", and allowed many hobbyists to become familiar with the device and its powerful minicomputer-like instruction set.

Of course the "baby" system was very small. Although it offered the same "PIPBUG" monitor program as the larger 2650 evaluation kits, resident in a 1k-byte ROM, it provided only a modest 256 bytes of RAM for user programs. And having been designed for economy rather than ease of expansion, it was not readily expanded into a larger system.

For this reason I suggested in the original article that those who were already fairly sure they would be progressing to a larger 2650 system might be better advised to start with one of the Signetics evaluation kits,

such as the PC1500 or the assemble-it-yourself KT9500.

As it happens, however, those who elected to start with the baby system can still change over to the KT9500 fairly easily — particularly if they followed our advice and used sockets for the microprocessor and monitor ROM chips rather than solder them directly into the PC board.

Applied Technology Pty Ltd has conversion kits available, so that you can upgrade from the baby system to the KT9500 at minimum cost. The conversion kit provides the 9500 PC board together with all of the required parts, apart from the 2650 microprocessor, the 2608 ROM with PIPBUG, and the two 2112 RAM chips.

With the KT9500, you have a much better starting place for an expanded system. Along with the PIPBUG ROM and 512 bytes of user RAM, there is full address decoding and fully buffered data and address bus lines. Also provided are two bidirectional 8-bit input/output ports, as well as serial input/output ports for a teleprinter,

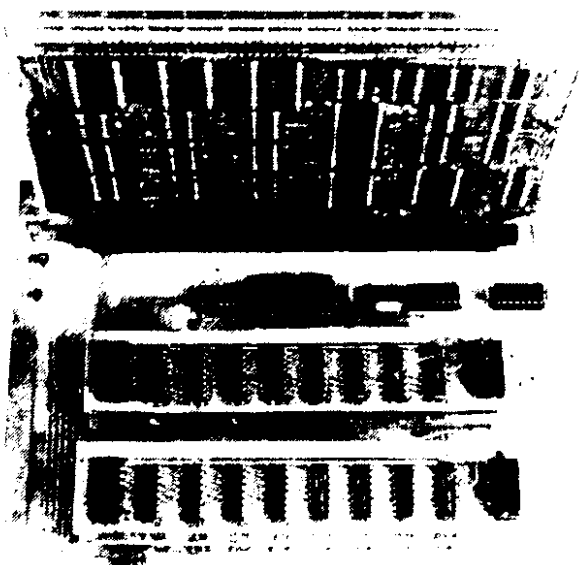
video terminal or similar device. The complete system is mounted on a PC board measuring 175 x 213 mm, which plugs into an accompanying 100-way edge connector.

Needless to say even though the KT9500 already offers enlarged capabilities, most enthusiasts find that they want to begin expanding it not long after they have it up and running. Probably the most common urge is to expand the memory, so that larger programs can be developed and run; the other urge is to replace the dual-monostable RC-timed clock oscillator with a more stable crystal clock.

To help you expand the KT9500 along these lines, Applied Technology has developed a "mother board" expansion kit which utilises their "RAM-stick" memory modules. As explained in our December 1977 issue (page 96), the AT RAM sticks are small PCB modules designed to be stackable by means of DIL sockets. Each stick provides 1k-bytes of low power static RAM, allowing an enthusiast to build up his system's memory in convenient and affordable increments.

The motherboard expansion kit assembles to form a PCB measuring 174 x 228 mm. The 100-way edge connector socket which comes with the KT9500 mounts directly on this PCB, so that the two boards now become an L-shaped assembly.

Adjacent to the main socket on the motherboard are six ICs, two of which



The picture on the facing page shows the Signetics KT9500 at right, with the Applied Technology motherboard and "piggyback" RAM sticks at left. The picture at left shows the two when assembled together. Up to 15 RAM sticks may be used.

are used to implement a crystal clock oscillator. This uses a 4MHz crystal, with division to the 1MHz required by the PIPBUG monitor and its serial communications routines. The remaining four ICs are used for additional address decoding and data bus buffering.

The address decoding circuitry uses a 74LS154 device to decode address bits 10, 11, 12 and 13. The sixteen decoder outputs thus become enable lines for 16 contiguous memory blocks of 1k-bytes each — so that they can be used to select up to 15 RAM sticks along with the PIPBUG ROM on the KT9500. The ROM must now be driven by the new decoder, and to enable this to be done a copper track must be cut on the KT9500 PCB, and replaced with a wire link to an unused edge connector pad. The on-board RAM chips are not used.

The motherboard is provided with four undedicated 16-pin DIL sockets along the front. These may be used for connection to the 8-bit input/output ports on the KT9500, or for any other desired purpose.

The motherboard PCB is double sided, although for economy it does not have plated-through holes. The constructor is thus faced with the rather daunting prospect of soldering in some 116 through-board wire links; however while doing this you can be cheered by the thought that you are saving money!

As it happens the through-board links are the major part of the job in assembling the kit, in any case. Apart from the links there are only six ICs, twelve bypass capacitors, two resistors and the crystal. Plus the 100-way connector and the four 16-pin DIL sockets for the RAM sticks, of course. So overall the assembly shouldn't be unduly tedious or time-consuming.

Using the motherboard it is thus quite easy to provide the KT9500 with a crystal clock, and to expand its RAM by 1k-byte increments up to 15k. You can then expand the system still further, if you wish, by adding a second motherboard with up to 16 further RAM sticks.

Incidentally Applied Technology is producing a metal case suitable for housing the KT9500/motherboard assembly, together with power supplies and even a floppy disc if you plan to go that far. It should be available by the time you read this.

Prices for the various items described above are as follows, with all prices inclusive of tax. A complete kit for the KT9500 is \$199, with the conversion kit for the baby system costing \$142. The motherboard kit costs \$35, while the 4MHz crystal costs \$7.95. Wired and tested RAM sticks cost \$25.50 each, but you can buy the RAM stick PCBs separately for \$6 each.

One of the things that is making the 2650 microprocessor increasingly popular with hobbyists is the growing library of support software. Much of the software has been generated by hobbyists themselves, many of whom started with our baby 2650 system.

Just about all of the software that has been generated to date is available to members of the 2650 Users' Group, so that it can be very worthwhile to join. The group is associated with Applied Technology, and further information is available from them at 109-111 Hunter Street, Hornsby, NSW 2077 (telephone 02-476 4758, 476 3759. Initial membership costs \$40, for which you get a documentation package with listings of many useful programs.

These include an assembler, a text editor, block move and search routines, hexadecimal input and listing routines, a disassembler, a reassembler, a tape verifier, maths routines, and many games programs including "Astro-Trek" and a Lunar Lander. Many of these programs are also available on cassette tapes, for a modest extra fee.

Next month we hope to present a few sample programs from the growing library of 2650 software, to whet your appetite. Who knows — they may spur you not only to join the Users' Group and get the rest of the library, but to write some programs of your own! ☺

# MICROPROCESSORS

## NEW RELEASE

## LOW COST V.D.U.

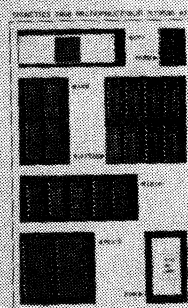
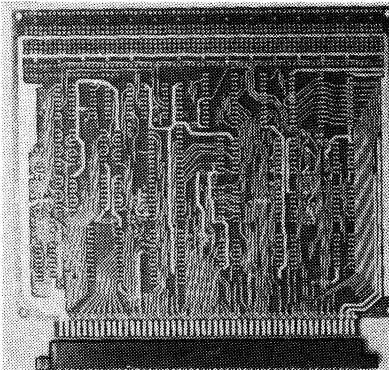
(E.A. FEB/MARCH 1978)  
Based on a clever design by Michael O'Neill of Newcastle Uni., this compact module is an ideal terminal for microprocessor users. The heart of the terminal is the E.A.100 V.D.U. as described in E.A. February, 1978. Exclusive features of our kit include —

- ☆ Top quality P.C.B. with plated through holes.
- ☆ Step by step Assembly Manual complete with waveforms and detailed circuit description.
- ☆ 4.43 MC. Xtal and trimmer cap supplied.
- ☆ Sockets for memories and character generator (to simplify setting up).
- ☆ Low power drain — uses low power Schottky devices, not standard TTL.
- ☆ Full service backup — details with kit.

E.A. 100 V.D.U. (Complete kit) **\$99.50**

### OPTIONS

KB04 PROFESSIONAL KEYBOARD	\$59.50
ENCODER/UART (See E.A. March)	\$32.00
MODULATOR/POWER SUPPLY	\$22.50



## KBO4 Professional Keyboard



## NEW RELEASE

We have now available a superior quality keyboard with UNIVERSAL ENCODING. This exclusive feature makes the keyboard ideal for software scanning or use with any keyboard encoder. It is ideal for the E.A. 100 V.D.U. and eliminates the tedious switch to switch wiring associated with other unencoded keyboards.

The KB04 is laid out in ASR33 format and includes two user defined keys.

A matching number pad KB05 is also available, as well as cursor control set KB06 and spare key switches (KB10).

KB04 UNIVERSAL KEYBOARD	\$59.50
KB05 NUMBER PAD	\$16.50
KB06 CURSOR CONTROL	\$7.50
KB10 MATCHING BLANK SWITCHES	\$2.00

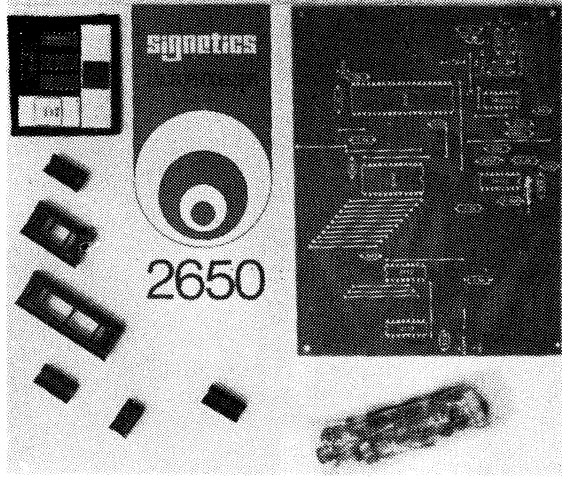
## 2650 Micro Computer

Now that you have the low cost terminal, you should consider the 2650 for your own microprocessor system. The 2650 is easy to learn to program, simple to use and features a powerful instruction set and a rapidly growing wealth of software support.

Using the kits detailed below you can readily expand your 2650 as your requirements and budget permit. The end result is probably the most cost effective home computer available in Australia to-day.

All systems are supplied with the incredibly effective PIPBUG operating system which handles all serial communication with the 2650, enables you to examine and modify address locations, set the registers, set breakpoints and also include a powerful routine that loads and dumps programs using a standard cassette tape.

BABY 2650 — STARTER KIT	\$75.00
B2650/KT9500 CONVERSION KIT	\$142.00
KT9500 FULLY BUFFERED KIT	\$199.00
KT9500 MOTHER BOARD with COMPONENT KIT (2650 RSMB)	\$35.00
RAM STICKS 1K x 8 MEMORY MODULES	\$25.50
2650 USERS GROUP	\$40.00



POSTAGE \$2.50 CERTIFIED PER ORDER

POSTAL ADDRESS P.O. Box 355, Hornsby, 2077

SHOWROOM 109-111 Hunter St., Hornsby 2077

PHONE 476 4758 — 476 3759



(9-5 Monday to Sat)

# SIGNETICS 8X300

Signetics has recently released an evaluation kit for its new 8X300 bipolar microprocessor. In this article we give a brief summary of the 8X300 chip itself, and of the evaluation kit.

**by DAVID EDWARDS**

The 8X300 has been designed to be a fast microprocessor controller, and because of this differs considerably from conventional NMOS microprocessors that we have considered in the past. Perhaps the major difference is that it is implemented with bipolar Schottky technology, and can fetch, decode and execute an instruction in only 250' ns.

The device is supplied in a 50-pin DIL ceramic package, and runs from a single 5V supply rail. An external pass transistor is required to complete an on-chip voltage regulator, which supplies 3V to selected areas of the chip. This helps to maintain the total current drain of the chip at less than 450mA.

Clock requirements are met by connecting a crystal directly to two pins. Alternatively, out of phase signals from

an external clock generator can be used. The remaining pins are divided into four functional groups, as detailed below.

The first thirteen pins connect to the instruction address lines, and allow up to 8192 words of program to be directly addressed. The next sixteen pins are the instruction word lines, allowing sixteen bit instructions to be passed to the processor.

Another eight pins are used for data memory and I/O purposes. Designated as the interface-vector (IV) bus, these allow data to pass from and to the processor. The remaining pins are used for IV bus control, and halt and reset functions.

The chip includes full instruction-decoding logic that interprets the particular class of instruction, such as input/output or arithmetic and logic, and

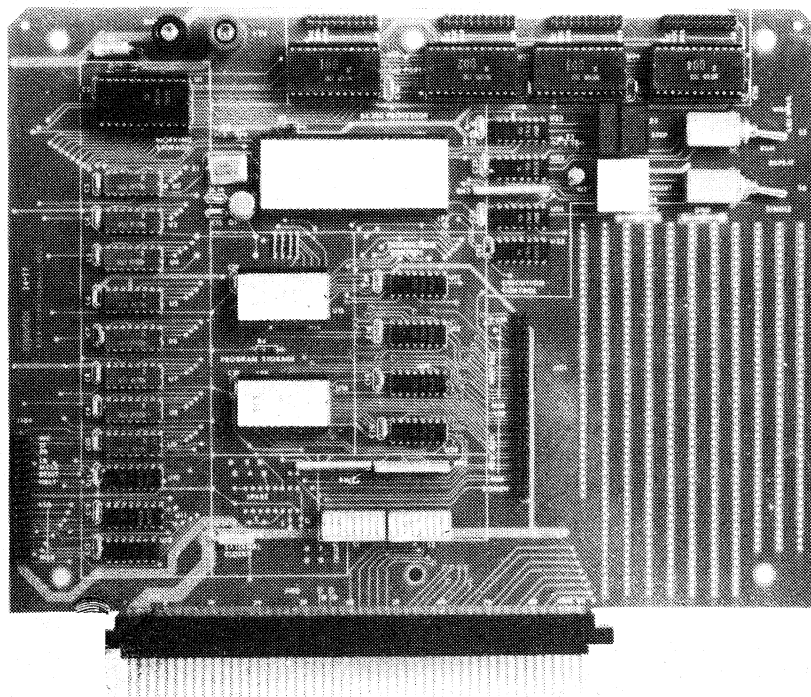
performs the indicated operation. The decoding and control logic supplies all internal signals for the processor, as well as signals on the control lines for directing the data input and output.

The processor also contains its own program counter which is automatically incremented upon execution of the instruction. The counter may also be left unchanged or loaded with a new value. Control of the current address is provided by the address register and may be derived completely or partially from the program counter, from the instruction data lines (AR0 through AR4), or from the output of the arithmetic/logic unit (lines AR5 through AR12). Because of this flexible instruction-address scheme, the order of execution may be altered by instructions or under conditions determined from selected data.

The processor manipulates 8-bit data bytes. Internal data is stored in 8-bit read/write registers—R1 through R6, R11, and an auxiliary register. The auxiliary register holds one of the operands used in two-operand instructions, such as ADD or AND, and a single-bit overflow register stores the carry-over bit from additional operations.

Interfacing with external circuitry is through an 8-bit bus called the interface-vector bus and consisting of lines IV0 through IV7. The bus carries both address and data information, and the accompanying data-I/O control lines tell the external circuitry which of the two types of information is on the bus. These lines include write- and select-control, right- and left-bank-signal, and master clock lines.

Since the interface-vector bus carries addresses as well as data, I/O ports on the external circuits must be enabled before data transfer can take place. This is usually accomplished by placing an address on the bus under program control and then activating the select-control line, which indicates that a valid address is on the bus. When presented with an address, each of the possible



*The evaluation kit includes 256 bytes of RAM and 512 words of PROM.*

512 I/O ports (two banks, each of 256 addresses) either enables itself upon identifying the address as its own or disables itself if the addresses do not match.

Within the processor, the interface-vector bytes are addressed in a unique fashion. Each byte has an 8-bit field-programmable address. When a given address is selected, the byte is automatically designated, and the 8X300 can then communicate with the I/O device. Moreover, once enabled, the addresses remain so until the processor changes them. This direct addressing feature is especially convenient if a few ports are to be accessed frequently. However if the time required for this operation is an imposition on the user, instruction memory can be extended so that the selection of ports is automatic upon instruction fetch.

The interface-vector bus is partitioned into two banks, allowing the 8X300 to select ports dynamically. The processor uses the left-bank (LB) and right-bank (RB) data-control lines as master enables for the I/O ports, as shown in the typical interconnect scheme of Fig. 1. Any two I/O ports can be active at the same time provided they are on opposite banks, and the ports recognize address, data, and controls only when enabled by the bank signal to which each is connected. Bank partitioning can thus be considered a ninth address bit that is alterable by the processor within an instruction, and it is this additional bit that permits direct addressing of 512, or  $2^9$ , I/O ports.

In a general data operation between two I/O ports, first an address is presented to one bank that enables an I/O port and disables all others on the bank. Next, another address is presented to the opposite bank, effecting a similar selection there. Then the operation between the two takes place.

Each 8X300 operation is executed in one instruction cycle (250ns), which is divided into four quarter cycles. The instruction address for an operation is presented at the processor output during the third quarter of the previous instruction cycle, and the program memory returns the instruction to the processor during the first quarter cycle.

In terms of processing data, the instruction cycle may be viewed as having two halves, an input and an output phase. During the first half of the instruction cycle, data is brought into the processor and stored in an interface-vector latch. Storage is completed during the first quarter cycle, and in the next quarter cycle the data is processed through the ALU. In the second half cycle, the output data is presented to the bus and finally clocked into the designated I/O port.

Bank selection during the input and output phases is independent. Thus data may be received from the right bank, processed, and then deposited in

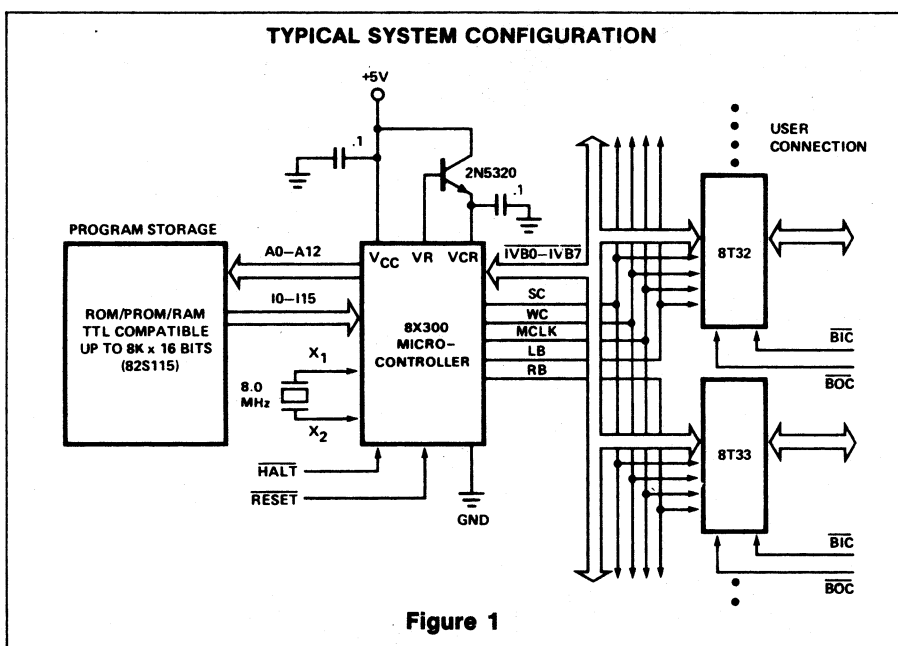


Figure 1

the left bank or vice versa, or may even be sent to and from the same bank. Bank selection during instruction cycles is specified by the instruction.

Each sixteen bit instruction is divided into one of eight possible classes. The MOVE instruction allows the contents of selected registers to be exchanged, or placed on the IV bus, or vice-versa. The ADD, AND and XOR instructions are similar, except that with these instructions the contents of the auxiliary register are combined with the source register before the MOVE part of the instruction is executed.

The XEC instruction allows a selected instruction at a different address to be executed without incrementing the program counter.

The NZT instruction allows a conditional branch to be implemented, while the JMP instruction implements an unconditional branch.

The remaining instruction class, XMIT, allows a binary pattern specified in the instruction to be placed in a specified register or on the IV bus. It is similar to a load-immediate instruction.

As you can see from Fig. 1, the main peripheral chip required to implement a typical working configuration, apart from ROM and RAM memory, is the 8T32 dual port register. This is an 8-bit bidirectional data register, which is accessible via either a microprocessor port (normally connected to the IV bus), or a user port.

A unique feature of the 8T32 is the way in which it is addressed. Each device has a field programmable 8-bit address, which is used to enable the microprocessor port when that address is present on the IV bus. A control signal (select control) is used to distinguish valid addresses from data.

Enabled ports remain open until another valid address is presented on the control line. Two 8T32 devices

which have been selected simultaneously can be differentiated from one another by means of the LB-bar and RB-bar lines, which separate the IV bus in two banks.

The evaluation kit for the 8X300 consists of a single large printed circuit board, measuring 280 x 210mm. It is fitted with an edge connector and matching socket on one edge. Included with the 8X300 chip are four 8T32s for external interface, 256 bytes of RAM for working data storage, and 512 words of PROM program storage.

Part of the PROM is programmed with I/O control, RAM control and RAM integrity diagnostic programs, with the remaining space being left free for user programs. Access is available to all address, instruction and IV buses as well as all controls and signals of the 8X300 itself. An area of the board is provided so that additional circuitry can be mounted using wire wrap techniques.

Controls are also provided for diagnostic and instructional purposes by allowing various operating modes, such as single stepping, instruction jamming and repeated instruction jamming. In these latter cases, the jammed instruction is selected by means of board mounted DIL switches.

An 8X300 programming course is also available. This consists of a large folder of written material, and is accompanied by 10 pre-recorded cassette tapes which interact with and explain the written material.

In conclusion, the 8X300 chip and its association evaluation kit are both rather specialised and will probably be of most interest to professional control equipment designers, rather than hobbyists. Further information can be obtained from Philips Electronic Components and Materials, 67 Mars Road, Lane Cove NSW 2066.

**Special offer for EA readers:**

# Low cost record of useful 2650 software

Here is some good news for those using small microcomputer systems based on the Signetics 2650 microprocessor. Electronics Australia and Philips Electronic Components and Materials, in conjunction with the 2650 Users' Group, have produced a low cost 175mm 33 1/3rpm record of useful 2650 system software. You can load the software into your system via any standard cassette interface.

**by JAMIESON ROWE**

Most small microcomputer systems based on the Signetics 2650 microprocessor use the monitor/debug program 'PIPBUG', resident in a ROM (read-only memory), to control program entry, manipulation and execution. And compared with many similar monitor/debug programs supplied with small microcomputer systems, PIPBUG is very good. It allows you to dump programs onto paper tape or cassette and reload them into memory, and to run them in controlled fashion with up to two breakpoints.

However like most small monitor/debug programs, PIPBUG has its limitations. After you have used it for a while, these become fairly apparent. You soon find yourself hankering for a faster and more convenient way of feeding long programs in, examining them when they have been fed in, moving parts of them around in memory, checking the accuracy of dumps, dumping and reloading, and so on.

As it happens, many of the utility programs required to do these things have already been produced, by people who have been working with small 2650 systems for a while. So there's no need for newcomers to "reinvent the wheel".

To help those who are just starting to get under way with their 2650 system, we have gathered together a group of these utility programs which we think are likely to be of most interest and value. With the generous support of Philips Electronic Components and Materials, and the co-operation of the 2650 Users' Group, we have recorded the resulting "software package" on a low cost 175mm 33-1/3rpm disc. This can be played on any standard record player, and fed into your 2650 system

via a standard cassette interface such as the one we described in the April 1977 issue (File number 2/CC/19).

The programs in the package include routines for feeding in programs faster, listing them more efficiently, moving them around in memory, searching them for certain instructions, verifying dumps, measuring the length of programs in dumped form, disassembling them for analysis, dumping them and reloading at higher speed than with PIPBUG, and producing dumps which automatically begin execution when they are loaded. There are also two short game programs, for amusement and system demonstrations.

All of the programs recorded on the disc have been dumped from a 2650 system using PIPBUG, so that they are in the Signetics "Absolute Object Format", and hence suitable for loading into other systems under PIPBUG control. The system from which they were dumped has a total of 4k (4096) bytes of RAM in addition to the 1k PIPBUG ROM, with the RAM occupying the hexadecimal address range 0400-13FF. Some of the programs currently occupy memory locations near the top of that range.

As many small 2650 systems are likely to have at least this much RAM, most of the programs should be usable as they are. However if your system has a smaller memory, you should still be able to use many of the programs. Quite a few of them are either relocatable without any changes at all, or require only a few minor changes. Others are already located down at the bottom of RAM memory space, and should be directly usable.

The programs have been recorded on the disc using the 2-tone "audio FSK" technique, with binary 1 and

"mark" represented by a tone of 2400Hz, and binary 0 and "space" represented by 1200Hz. These are the same tones used in standard microcomputer cassette interfaces, based on the so-called "Kansas City Standard" originated by the American magazine Byte. Hence you should be able to feed the programs from the disc into your system simply by connecting a standard record player up to your system's cassette interface, in place of the cassette tape recorder.

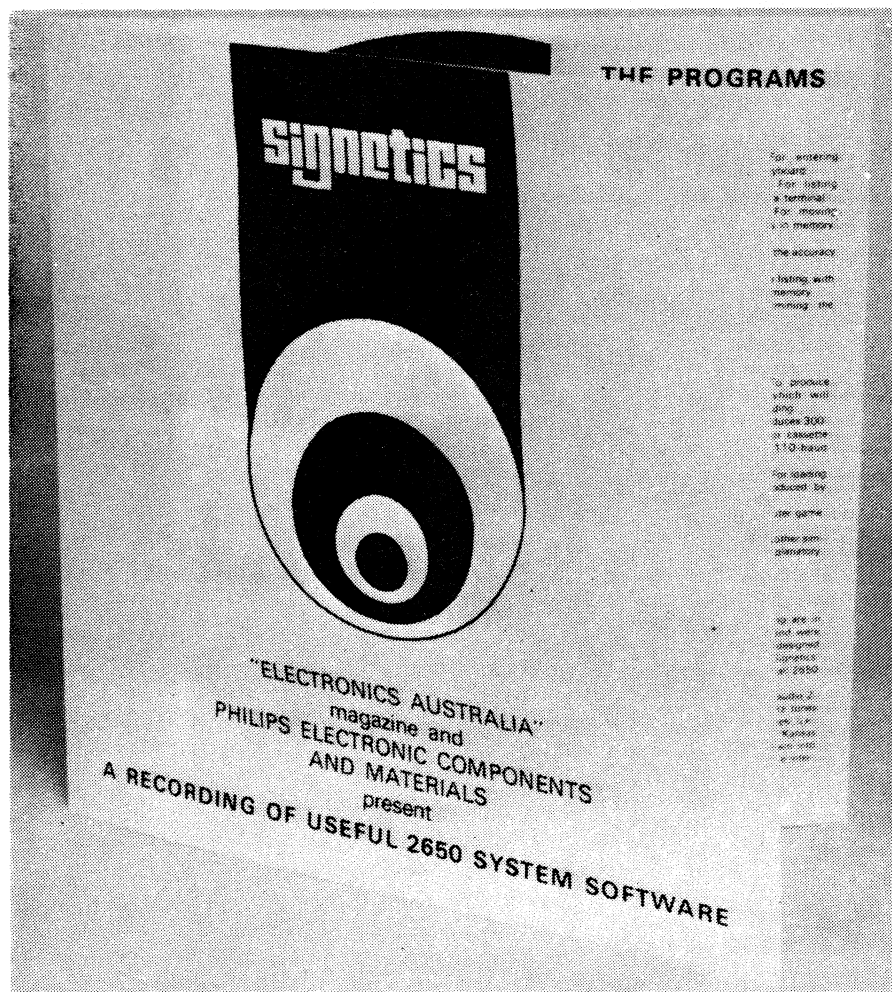
If you experience any trouble loading them into your system in this way, it will probably be because your cassette interface is not set for exactly the standard frequencies. A judicious adjustment of the interface may therefore be required, by trial and error, until loading takes place correctly. This will be a simple procedure if you are using the cassette interface described in the April 1977 issue, as you will only need to adjust the 4.7k "clock adjust" preset pot a little one way or the other.

**As a special offer to EA readers, we are making the 2650. Software Package Recording available at the nominal price of \$2.50, or \$3.00 posted anywhere within Australia. However your remittance should be accompanied by the order coupon given in this article, unless you live in a State where this requirement is illegal — in which case a letter giving the same information may be sent instead.**

**But note that this offer is strictly limited. Only 1000 discs have been produced, and when these have gone the offer must close. So be early if you don't want to miss out!**

As you can see from the photograph, the 2650 Software Package Recording comes inside a matching protective sleeve. On the sleeve is printed brief information on each of the various programs on the disc, and their use. However in order to let you evaluate their potential value to you in advance, the remainder of this article gives a somewhat expanded description. Also given are program sizes and relocatability.





Supplied in an informative sleeve, the record plays on a standard player. It provides nine handy items of 2650 software, plus two games.

### 1. HEX INPUT ROUTINE

This simple program allows either programs or data to be fed into your system in hexadecimal code from a terminal keyboard, more speedily and more conveniently than with the PIPBUG input routine. The data or instruction bytes are fed in as lines of any length, each line beginning with the address in which its first data byte is to be stored. The address and each data byte must be terminated by any convenient non-hex character, such as a space or comma.

Thus by typing:  
440s3Fs82s69sCDs84s7Fr  
where "s" is a space, and "r" is a carriage return, the input routine will load 3F into location 440, 82 into 441, 69 into 442 and so on. Note that the last data byte on the line may be terminated by the carriage return character; a space is not necessary. The program automatically provides a line feed, also.

When typing in both the address and the data bytes, no leading zeroes are necessary. Thus an address typed as "440" is automatically interpreted as 0440, while a data byte with a value of 02 may be entered simply by typing

"2s". Zero bytes may be entered by simply typing a terminator character, such as a space.

As the program automatically enters only the last four digits before the terminator, in the case of an address, or the last two digits in the case of a data byte, errors discovered before typing the terminator may easily be corrected. Simply continue typing, to make the last four or two digits correct. Thus typing:

44F0440s6BCs

will enter C8 into location 0440. But note that when correcting errors in this way, you must type in any leading zeroes as well.

Any number of lines may be entered, as long as each line begins with its appropriate initial address. The addresses of each line need not follow those of the line before, nor precede those of the next line; all lines are treated independently. This allows convenient correction of lines, and entering of multiple programs.

To escape from the program and return to PIPBUG, either type a Control-G (BELL character) or press the system reset button. It may be necessary to type control-G twice.

As recorded on the disc, the hex entry routine occupies memory locations 1250 — 162D. However it is relocatable and may be moved anywhere in page 0, that is anywhere from 0440 to 1FFF (PIPBUG itself occupies 0000 — 03FF). It also contains no scratchpad locations, making it suitable for storage in a ROM if desired. It uses PIPBUG subroutines STRT, CHIN and COUT. Call by typing G1250r. This program was written by the author.

### 2. HEX LISTING ROUTINE

This program enables you to list a program or data stored in your system's memory on a terminal, in hexadecimal code, more conveniently than with PIPBUG. The listing is done in lines, with each line beginning with a 4-digit address and followed by up to either 16 or 8 two-digit groups representing the data bytes, separated by spaces. The memory range to be listed is given to the program as part of its calling protocol; when called the program lists the memory contents in the specified range, then returns control automatically to PIPBUG. It must therefore be called separately for each listing.

As recorded on the disc, the listing program occupies memory locations 1200 — 1248 inclusive. However it is relocatable, and may be moved anywhere in page 0. It may also be stored in a ROM if desired. Call by typing G1200sAAAA sBBBBr, where AAAA is the start and BBBB is the finish addresses of the range to be listed.

At present the program is arranged to list in lines of up to 16 data byte groups, so that lines will have up to 53 characters. If the terminal or printer you are using can only handle lines of 32 characters or less, you can alter the program to list in 8-byte groups by changing the instruction byte in location 1244 from "0F" to "07".

The hex listing routine uses PIPBUG subroutines GNUM, STRT, CRLF, BOUT and COUT. It was written by the author.

### 3. BLOCK MOVE & SEARCH

The block move routine allows you to move the contents of the locations in any designated memory range either up or down in memory. It may thus be used to move complete programs or data, or to move part of a program for insertion or deletion of instructions. The destination and source ranges may overlap, so that moves of as little as one byte are permitted in either direction. Note, however that the program uses indexing and will not move data correctly where either the source or destination ranges flow over 2650 page boundaries. However the source and destination range may lie in separate pages.

As supplied the block move routine occupies memory locations 1100 — 1183. However it is relocatable and may be moved anywhere in page 0. It uses PIPBUG subroutines STRT and GNUM.

## 2650 SOFTWARE PACKAGE RECORDING

It is called by typing G1100sAAAA sBBBB sCCCCr, where AAAA and BBBB are the start and finish respectively of the present memory range occupied by the block to be moved (i.e., the source range), and CCCC is the start of the memory range to which it is to be moved (i.e., the destination range).

After moving the data, the block move routine automatically returns control to PIPBUG. The routine was written by Ian Binnie.

The accompanying block search routine is designed to search through a designated memory range for a specified pattern in two adjacent locations. Wherever the pattern is found, the routine prints out the address of the second byte of the pattern. It may therefore be used to find specific instructions in a program, or data in a table. It can search any desired memory range, even a range which flows over a 2650 page boundary.

The block search routine is designed to be used in conjunction with the block move routine, and this is why the two are combined on the record. However the two are quite independent, and may be separated if desired. As supplied the block search routine occupies 1190 — 11D7, but it is relocatable and may be moved anywhere in page 0.

To call the block search routine, type G1190sAAAA sBBBB sXXXX, where AAAA and BBBB are the start and finish respectively of the memory range to be searched, and XXXX is the two-byte pattern to be found. The routine will print out the locations at which it is found, and then return control to PIPBUG. The block search routine uses PIPBUG subroutines STRT, GNUM, CRLF and BOUT. It was written by Craig Barratt.

### 4. TAPE VERIFIER

After you have dumped a program from your system's memory onto paper tape or cassette using PIPBUG, this verifier program lets you check that the tape or cassette has a faithful copy. It does this by reading the tape or cassette, and comparing it with the original still residing in the system memory. If there are any errors, the verifier program will type out an appropriate message. Otherwise it will type out "TAPE OK".

The verifier checks for both address and data BCC (block control character) errors on the tape or cassette, as well as for data byte errors. Currently the verifier occupies memory locations 1360 — 13F3 inclusive. However it may be moved to any desired part of page 0 by modifying the contents of the instruction bytes currently in addresses 13B6 and 13B7. The five least significant

bits of the byte in the first location and the full byte in the second must correspond to the address of the byte SIX BYTES after the second of the two bytes, for correct printout of the verifier messages.

Thus currently these bytes are 37 and BD, corresponding to address 13BD. If the verifier were moved to occupy 760-7F3, you would thus need to change the contents of 7B6 and 7B7 to 27 and BD respectively. If it were moved to occupy 500-593, the contents of 556 and 557 would need to be changed to 25 and 5D respectively. Note that the sixth least significant bit of the first of the two bytes is always set; this is for correct indexing.

To use the verifier, simply call it by typing G1360r. Then feed in the tape or cassette, as if you were loading it. Note, however, that for correct operation the original program on the tape or cassette must still be resident in the system memory. The verifier will either type out a message as soon as it finds an error, or will give the "TAPE OK" signal at the end of the tape. After giving a message the verifier returns control back to PIPBUG.

The verifier uses PIPBUG subroutines CRLF, CHIN, BIN and COUT. It was written by the author.

### 5. DISASSEMBLER

This program may be used to examine a program or part of a program in your system's memory, and produce both a hexadecimal listing and a reconstruction of the program in mnemonic or assembly language. This allows convenient analysis of programs, and is also of value in tracking down subtle logic errors, errors in program entry and errors in calculating relative addresses and PC-relative branches.

Not all of the codes in the 2650 instruction set are translated into mnemonic form by the disassembler; some infrequently used codes are ignored. However all commonly used codes are translated, and absolute ad-

resses are calculated for relative addressing instructions. This allows very convenient program analysis. However please note that the program does not calculate the absolute address correctly for relative indirect addressing instructions which are "forward referencing" — i.e., those which reference higher addresses. It does calculate the correct address for those which are backward referencing.

The disassembler listing is 31 characters wide, making it suitable for use with almost every kind of terminal and printer. It occupies the memory range 0F00 — 10B2, and is not easily moved.

To use the disassembler, call it by typing GF00sAAAA sBBBBr, where AAAA and BBBB are the start and finish of the range in memory occupied by the program or section of program to be disassembled. For long programs, the disassembler will pause after listing about 64 lines to allow manual form feeding. To continue the listing, type any character on the terminal keyboard. Control is returned to PIPBUG at the end of the listing.

The disassembler uses PIPBUG subroutines STRT, GNUM, BOUT, AGAP, CRLF, CHIN, COUT and FORM. It was written by Ian Binnie, with modifications by the author.

### 6. TAPE MEASURE

If you acquire a program on paper tape or cassette in Signetics Absolute Object Format, it is usually easy enough to feed it into your system and try it out. However in order to list it or disassemble it for analysis, one needs to know its length or the range it occupies in memory. This program is designed to read programs stored on paper tapes or cassettes, and print out the memory range of each block. It prints out this information at the end of the tape or cassette, as a small table having one line per block.

The program occupies the range 440 — 4FE, and is not easily relocated. It also requires RAM buffer area above 4FE, for storage of block start and finish addresses during reading. Four bytes of storage are required for each block on the tape to be measured. To use the program, simply call it by typing G440r,



## 2650 SOFTWARE PACKAGE RECORDING

then feed in the tape or cassette to be measured.

Please note that the program lists the ends of blocks as one location in memory higher than their true position, so that the block ends listed should be decremented to find the true ends.

The tape measure program uses PIPBUG subroutines CHIN, BIN, STRT, COUT and BOUT. After printing out the block information at the end of the tape or cassette, it returns control automatically to PIPBUG. It was written by the author.

### 7. DUMP FOR AUTO-START

This routine duplicates the function of the dump routine in PIPBUG, except that it allows you to produce program tapes or cassettes which begin executing automatically as soon as they have been loaded into your system using the PIPBUG load routine.

The routine currently occupies the range 0E60 — 0EF7, but may be moved anywhere in page 0 by changing the contents of the last two bytes. These currently contain the branch address 0E7C, and if the routine is moved they must be changed to contain the corresponding address.

To use the routine, type GE60sAAAAAsBBBBBsCCCC, where AAAA and BBBB are the start and finish of the memory range to be dumped, and CCCC is the address at which automatic starting of execution is to occur upon loading. Then turn on the tape punch or set the cassette recorder for recording, and finally type a carriage return. The routine will return control to PIPBUG after performing the dump.

The routine uses PIPBUG subroutines STRT, CRLF, COUT, GAP, and BOUT. It was written by the author.

### 8. 300 BAUD BINARY DUMP

This program is designed to dump programs onto cassette tape, in binary format and at 300 baud, so that they may be reloaded into your system considerably faster than with the 110-baud Absolute Object Format used by PIPBUG. This gives roughly a six times reduction in loading time, for the programs themselves.

The program provides two main options. Programs may be dumped alone, or preceded by a bootstrap loader. If preceded by the bootstrap loader, binary cassettes may effectively be loaded using the normal PIPBUG load routine. If dumped without the bootstrap, binary cassettes must be loaded using the following binary loader.

For dumping programs with the bootstrap preceding, the following binary loader must be resident in memory, because it is used as the

bootstrap source.

Dumping programs with the bootstrap loader preceding them does increase the loading time, tending to reduce the advantage over normal PIPBUG dumping and loading. However it saves having to load in the binary loader in advance. And the increase in loading time is really only significant for very short programs; even programs as short as 256 bytes still load in little more than half the normal time (39 seconds compared with 68 seconds). For large programs the loading time either with or without the bootstrap is drastically reduced: an 8k memory dump can be reloaded in 5½ minutes, compared with over 30 minutes with PIPBUG.

A further option provided is for the dumped program itself to be set for automatic execution after being loaded.

The binary dump routine occupies 1200 — 12FF and cannot easily be relocated. Its starting address for dumps preceded by the bootstrap is 1204; for dumps without the bootstrap start at 1223. Call by G1204sAAAAAsBBBBBr (or G1223sAAAAAsBBBBBr) for non auto-start of the dumped program, or G1204sAAAAAsBBBBBsCCCCr (or G1223sAAAAAsBBBBBsCCCCr) for auto-starting, where AAAA and BBBB are the start and finish of the program being dumped, and CCCC is the address for auto starting.

The routine uses PIPBUG subroutines STRT, GNUM and CBCC. It was written by Ian Binnie.

### 9. 300 BAUD BINARY LOADER

This routine is designed to load programs into memory from 300-baud cassette recordings made using the preceding dump routine, when the cassettes do not have the loader already present as a bootstrap. It is also used by the dump routine as a source for the

bootstrap. It occupies 440 — 497, and cannot easily be relocated. Call by G440r. Written by Ian Binnie.

### 10. NIM GAME

A simple version of the traditional computer game of strategy. When called by typing G440r, it announces itself and explains how to play the game. It occupies 440 — 588. The version presented has been adapted by the author from a program written by Perry Brown.

### 11. NUMBER GUESSING GAME

Another simple game of strategy, for amusement and diversion. Like the Nim game, it announces itself and explains how to play. It occupies 440-59F, and is called by typing G440r. The version presented here has been adapted by the author from a program written by Perry Brown.

Programs 3, 5, 8 and 9 are presented by permission of the 2650 Users' Group and Applied Technology Pty Ltd, and we thank them for their courtesy in allowing us to do so. Further information on these programs is available to members of the Users' Group. If you are interested in joining the group, its address is 109-111 Hunter Street, Hornsby, NSW 2077. Initial membership costs \$40, for which you get a documentation package with hexadecimal listings of many other useful programs.

Incidentally, we aren't able to supply hexadecimal or source listings of the programs on the record. However this should be no problem, because you can produce hex listings and mnemonic listings of them for yourself, using the hex listing routine and the disassembler program on the disc itself! Both of these programs will happily process themselves along with all of the others, too — so that you can make the hex listing routine list itself, and the disassembler disassemble itself...

In short, we think you'll find the 2650 Software Package Recording very handy, and good value at the price. If you agree, why not fill in the order coupon below and send it in with your remittance?

## ORDER FORM FOR EA-PHILIPS 175mm DISC OF 2650 SOFTWARE

To obtain your special offer record of useful 2650 system software, complete this form and send it with a cheque or money order to Electronics Australia, PO Box 163, Beaconsfield, NSW 2014. Note that only 1000 records are available, and when these are exhausted the offer will close. Records are \$3.00 each posted anywhere in Australia.

NAME .....

ADDRESS .....

..... POSTCODE .....

I enclose \$ , being payment for ..... discs

# INFORMATION CENTRE

## NOTES & ERRATA

**2650 SOFTWARE RECORD** (April 1978):  
In the section on page 81 describing the Hex Input Routine, the memory address range currently occupied by the routine should read from 1250 to 12BD hexadecimal. Also in the section on page 83 describing the Tape Verifier, the current content of location 13B6 should read 33, not 37 as shown.



**Uses latest 4K-bit RAM chips:**

# New, expandable 2650 mini system

The Signetics 2650 microprocessor has become quite popular among computer hobby enthusiasts, spurred on by our "baby" 2650 system described in the March 1977 issue. This and the recent release of 1024 x 4 bit RAMs has prompted us to redesign the circuit, with this article and the unit described herein as the result.

**by DAVID EDWARDS**

Our previous approach was to present a design for a printed circuit board unit only, which reduced costs to a minimum. With this project however, we are also describing a case and power supply, so that the unit becomes a complete stand-alone mini computer. Of course, if desired the PCB can be used by itself, as before.

We estimate that the complete unit will retail for around \$115.00, which is very reasonable considering the features of the unit.

What are the features of the unit? Well, it has a debug and monitor program resident in a 1k ROM, a standard 20mA asynchronous communication link, a minimum of 1k of RAM, full memory decoding, provision for memory and I/O expansion, and an on-board power supply.

All this is contained on a single-sided PCB, measuring only 218 x 81mm. The only other components required are a reset switch and a power transformer and associated hardware.

At the heart of the circuit is the 2650 MPU chip itself. This is an 8-bit device, with an instruction set of 75 instructions, and having eight different addressing modes. It is fabricated using low threshold ion implantation, and is an N-channel silicon gate device operating from a single 5V supply, with all inputs and outputs TTL compatible.

A 74123 dual monostable is used to generate the single phase 1MHz clock required. A trimpot is used to set the correct operating frequency, which can be adjusted without the use of special test equipment.

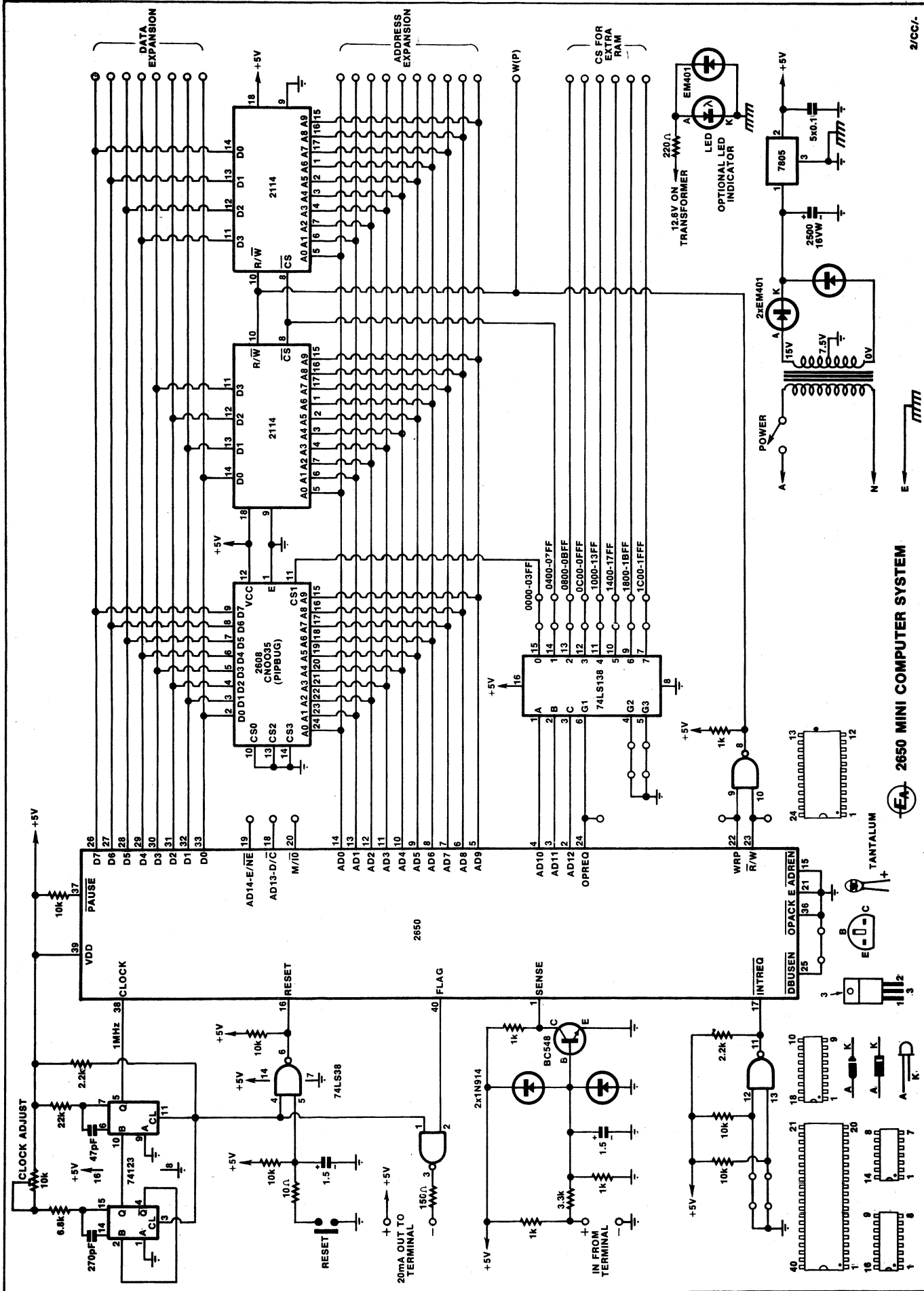
The debug/monitor program, code

named "Pipbug", is resident in a 2608 ROM. Pipbug recognises seven basic commands, each of which consists of an alphabetic character, any required numerical parameters, and a terminating return. The parameters are given as hexadecimal characters, with leading zeros unnecessary.

The seven commands and their functions are as follows:

- A — See and alter memory;
- B — Set breakpoint (2 permitted);
- C — Clear breakpoint;
- D — Dump memory to tape;
- G — Go to address, run;
- L — Load memory from tape;
- S — See and alter registers.

*Only two of a maximum of eight RAM chips are shown in the circuit diagram at right. The rest are wired similarly.*



## New 2650 system

The D command may be used to dump out onto paper or magnetic tape any desired range of memory locations, with leader, checksum and trailer to facilitate reloading. Both the A and S commands may be auto-incremented, by terminating with a line feed instead of a carriage return.

Pipbug is explained further in Signetics Application Memo SS50, which you should receive with the 2608 ROM. It also includes a listing of Pipbug, which among other things lets you make use of some of its utility sub-routines such as the serial input and output routines "CHIN" and "COUT".

Only two ICs are required to implement the basic 1k of RAM provided, thanks to the new 2114 memory devices. These are 4096-bit static memories, organised as 1024 4-bit words. Access time is 650ns or better, and all inputs and outputs are TTL compatible. At the time of writing, only devices made by Synertek are available, and these are coded SY2114.

We understand that in the near future similar devices will be available from National Semiconductor (coded MM2114) and Signetics (coded 2614). At present, the Synertek devices are available from Radio Despatch Service Pty Ltd, of 869 George Street, NSW 2000, and also from Dick Smith Electronics stores and dealers.

Memory address block decoding is performed by the 74LS138 device. The A, B and C inputs, operating on address lines AD10, AD11 and AD12 produce output signals which effectively divide

*This close-up photograph of the board should aid in placing the components on the PCB.*

## List of component parts

### SEMICONDUCTORS

- 1 2650 MPU chip
- 1 2608 CN0035 ROM (Pipbug)
- 2 2114 1024 x 4 static RAMs
- 1 74LS38 quad open collector gate
- 1 74123 dual Schmitt trigger
- 1 74LS138 decoder
- 1 7805, LM340T-5.0 5V regulator
- 1 BC548 or similar NPN transistor
- 2 1N914 or similar silicon diodes
- 2 EM401 or similar silicon diodes

### CAPACITORS

- 1 2500uF 16VW PCB mounting electrolytic
- 2 1.5uF tantalum electrolytics
- 5 0.1uF polyester
- 1 270pF polystyrene
- 1 47pF polystyrene

### RESISTORS (all 1/4W)

- 1 10k trimpot (5mm lead spacing)
- 1 22k, 5 10k, 1 6.8k, 1 3.3k, 2 2.2k, 4 1k, 1 150 ohm, 1 10 ohm

### MISCELLANEOUS

- 1 40 pin DIL socket

- 1 24 pin DIL socket
- 2 18 pin DIL sockets
- 3 PCB standoffs (9.5mm)
- 1 SPDT miniature toggle switch
- 1 SP miniature momentary contact switch
- 1 transformer, 240V to 15VCT @ 1A.
- DSE 2155, A&R 2155 or similar
- 1 PCB, coded 78up5, 218 x 81mm
- 1 case, 284 x 93mm (see text)
- 1 output connector (see text)
- 4 rubber feet

- 1 mains cord, mains plug, grommet, cord clamp and terminal block
- 2 aluminium brackets (see text)

Machine screws and nuts, PCB pins, solder, tinned copper wire, hookup wire, rainbow cable

NOTE: Resistor wattage ratings and capacitor voltage ratings are those used for our prototype. Components with higher ratings may generally be used provided they are physically compatible.

the memory space into eight 1k blocks, each of which is uniquely decoded within the 2650's memory address "pages" of 8k.

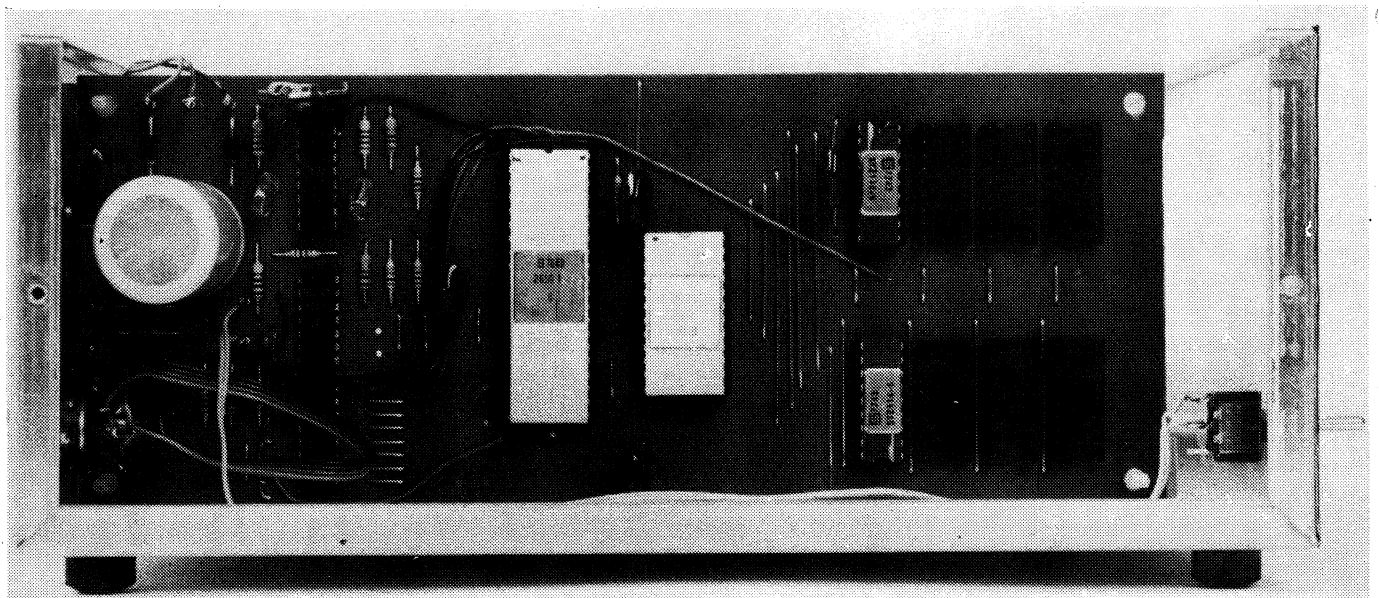
The first 1k block, from 0000 to 03FF, is assigned to the ROM, with the next block, 0400 to 07FF, assigned to the first 1k of RAM. The first 63 locations of this block are used by Pipbug as temporary storage locations, so that user memory commences at 0440.

A 74LS 38 quad open collector gate is used to perform the remaining housekeeping functions. One element is used to combine the R-bar/W and WRP signals, to form the W-bar (P) signal, which is then used to drive the R/W-bar lines of the RAMs.

A second element is used to buffer and invert the signal from the reset switch, allowing a cheap and readily available normally-open type switch to be used.

The third element is used as a 20mA current sink for the teleprinter (TTY) output signal. It was for this reason that an open collector type gate was used, necessitating the three pullup resistors on the other element outputs. The current level is set by the 150 ohm resistor, and is sufficient to operate the current loop input of an ASCII TTY or video data terminal.

The remaining gate element is not strictly required, but since it was available, we have used it to provide







## New 2650 system

photosensitive aluminium, and we hope that commercial versions will be made available in due course.

We assembled the LED indicator and its support components on a small piece of tagstrip, mounted immediately below the LED BEZEL. Only one wire is required from the tagstrip to the transformer, the earth connection being made via the chassis and the tab of the power supply regulator.

Commence construction of the PCB by fitting the uninsulated links. All soldering should be done with a small pencil shaped bit, and with a minimum of solder. Be careful to avoid solder bridges.

We recommend that sockets be used for the CPU chip, the ROM chip, and the RAMs if you are at all unsure of your soldering ability with these MOS devices. Or if you wish to add extra RAM at a later date. Fit the sockets to the board at this stage, before any other components are fitted.

Now fit all the passive components, followed by the TTL ICs. Circuit board pins for the external connections should also be fitted at this point, if they are required. Mount the regulator IC, and then locate and drill the mounting hole for it in the rear panel.

The next step is to fit the eight insulated links to the board. Use rainbow cable, and join the lettered points together, routing the wires between the components, so that a neat finish is obtained. Once this has been done, mount the board in the chassis, and complete the connections to the outer connector, the reset switch and the power transformer.

Now visually check the completed board for misplaced or misoriented components, and for dry joints and solder bridges. Bolt the regulator IC to the chassis, using a little heatsink compound for improved heat transfer. Do not insulate the mounting tab from the chassis.

Now monitor the 5V rail, and switch on. If the supply does not rise immediately to 5V, switch off, and trace and rectify the fault. If all is well, adjust the clock preset so that the signal at pin 5 of the 74123 is 1MHz. If you have no means of measuring this frequency, leave this adjustment till later.

Now switch off, and insert the MPU chip, the ROM chip and one pair of RAMs. Note that the lower RAM is inverted with respect to all the other ICs. Then connect a suitable TTY or video terminal, and switch it on.

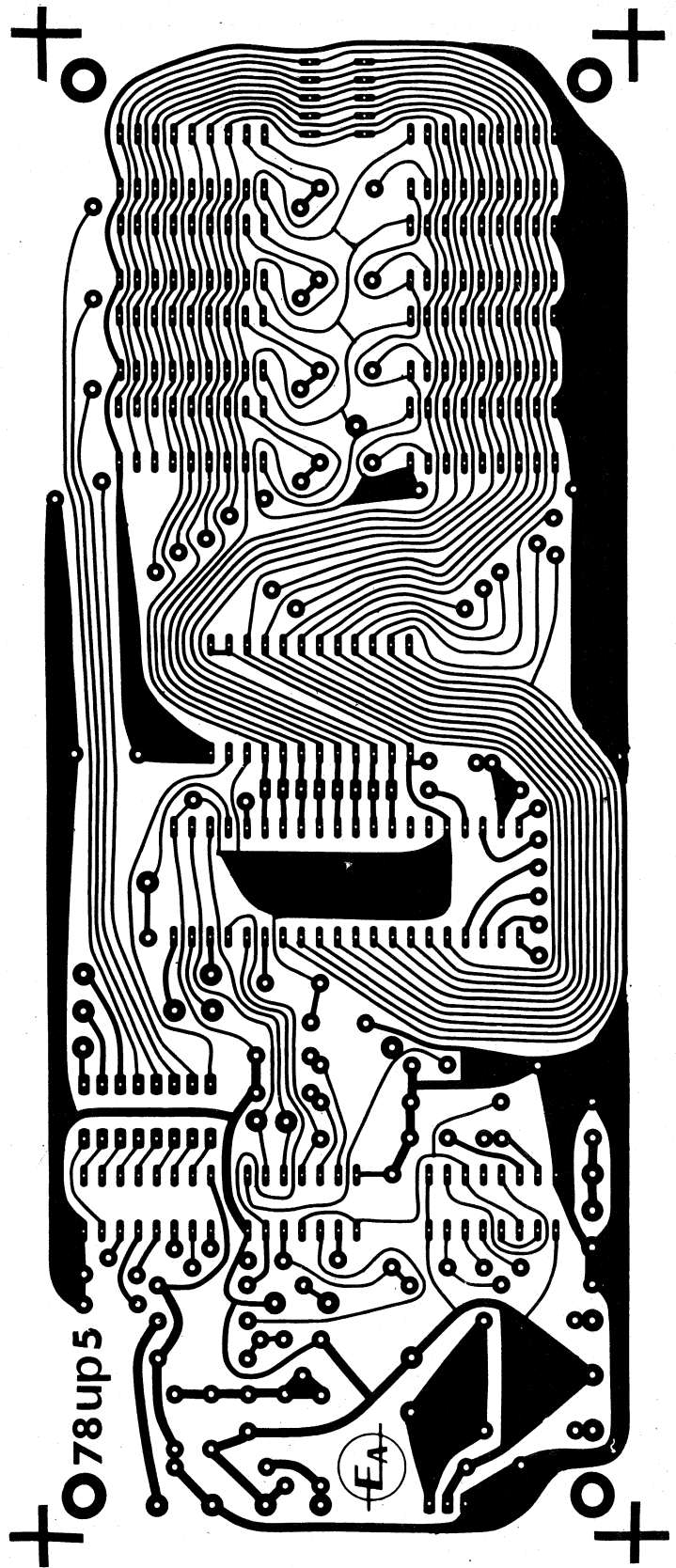
Switch the computer on, and then press the reset switch. You should be rewarded by a carriage return, a line feed and an asterisk (\*). If you get something garbled, adjust the clock frequency while repeatedly pressing the reset switch till an asterisk appears. Once this happens, set the preset to the middle of the region wherein an asterisk can be obtained.

Now check out the Pipbug commands, and verify that you can load data into memory. If you have purchased the 2650 Software Package Recording offered in the April 1978 issue, then you will be able to load and run some of the smaller programs, such as "Nim" and "Number Guessing".

Note that if you can afford the full 4k of memory, you will be able to run all of the programs, with room left over for your own programs as well. And if you haven't ordered your record yet, do it now, as stocks are strictly limited.

Finally some comments concerning the expansion capabilities of the board. Most of the MPU pins required for expansion have been made available on the board. Some of them are grounded via links for normal operation, so that these links will have to be removed for expansion purposes.

Eight links have been provided at the 74LS138 outputs so that the memory configuration can be altered if



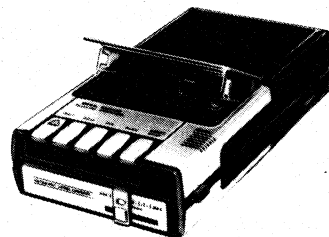
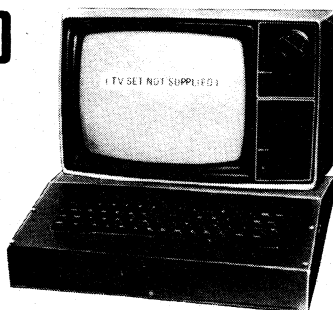
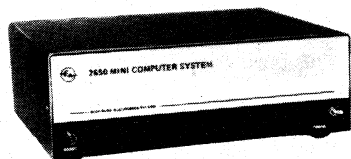
This is an actual size reproduction of the PCB pattern.

required. These are arranged in a DIL pattern, so that a socket and programming plug can be used if desired.

The address and data lines, as well as the W-bar (P) signal, have been made available, so that memory expansion and input/output capabilities can be provided. It is our intention to present a second article in the near future, showing how to expand the memory up to at least 7k of RAM, and provide two non-extended I/O ports.

# MINI-COMPUTER BREAKTHROUGH!

**COMPLETE SYSTEM \$369.20**

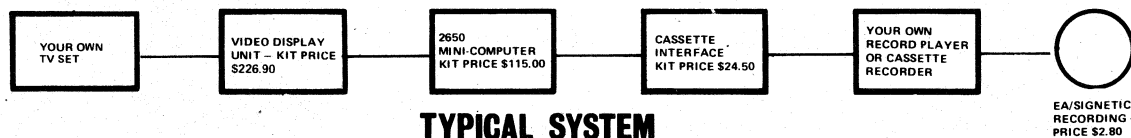


Here it is at last — the first mini-computer kit system for the electronics enthusiast who knows nothing (or a lot) about computers.

If you're one of the people who think that computer technology is beyond you, this is the system for you.

Just imagine it — after building it, you use your standard record player to 'input' the program with the EA/Signetics record, and you're using your computer right away. You don't need to know any complicated machine 'languages'. You communicate with it in English!

Incredible bargain — you'd pay over \$800 for a similar built-up system!



## 2650 MINI-COMPUTER

Fantastic new 2650 Mini Computer (see May '78 EA). Complete kit includes all electronic parts, PC board and power supply plus case, Marvi-plate lid and deluxe brushed aluminium front panel.

Complete kit for above: Cat K-3447 .. .. . \$115.00\*

PCB (78up5), fibreglass, only: Cat H-8341 .. .. . \$5.00

2650 microprocessor chip only: Cat Z-9201 .. .. . \$28.50

2608/CN0035 pip bug 8k ROM only: Cat Z-9309 .. .. . \$19.75

## VIDEO DISPLAY UNIT

Incredibly low cost Video Display Unit uses your own TV set as the monitor. See EA, February & May '78 for details.

Basic Video Display Kit: Cat K-3460 .. .. . \$97.50 \*

Video Modulator Kit for above: Cat K-3462 .. .. . \$4.50 \*

ASCII Keyboard Encoder Kit for above: Cat K-3464 .. \$39.50 \*

UART IC (S1883/MM5303N/TMS6011: Cat Z-9204 .. \$5.90 \*

Keyboard Console Metalwork: Cat H-3130 .. .. . \$24.50 \*

Keyboard (fully assembled): Cat X-1180 .. .. . \$55.00 \*

## CASSETTE TAPES

AC/DC cassette recorder ideal for this system.

Cassette recorder Cat A-4092 .. .. . \$39.95

Cassette tapes: C60 LN — Cat C-3350 .. .. . \$1.50

C90 LN — Cat C-3352 .. .. . \$2.00

## CASSETTE INTERFACE

Enables your cassette recorder or record player to interface with mini-computers such as the 2650. Kit includes PC board and all components except power transformer. The complete PC board assembly will fit inside your 2650 mini-computer case, the 2650's transformer providing the AC power. We believe this kit to be the best available on the market to suit the 2650 system. Other kits require long setting-up time and the final result is not nearly as good as with this one.

Complete kit (as above) Cat K-3465 .. .. . \$24.50 \*

PCB only (Cat H-8331) .. .. . \$3.75

## PAPER TAPE READER KIT

See page 33 or our new catalogue for full details. Ideal for use with the 2650 mini-computer. Kit includes all electronic components, handsome black anodised aluminium case, ribbon interface cable and complete assembly and interface instructions, schematics and software.

Tape Reader Cat K-3466 .. .. . \$95.00

## SOFTWARE RECORDING

This is a 33-1/3 RPM recording of useful 2650 system software. By using it on your record player or dubbing it onto a cassette and using it in your cassette interface system you can program the 2650 directly for exciting new programs and competitive games. It contains 11 programs you can run.

Record Cat B-6300 .. .. . \$2.80 \*

**COMPONENTS MARKED \* ARE REQUIRED TO BUILD THE SYSTEM SHOWN IN THE BLOCK DIAGRAM ABOVE. THEY ADD UP TO ONLY \$369.20!**

# DICK SMITH ELECTRONICS



SYDNEY:  
125 York St.  
City. Ph 29 1126.

147 Hume Hwy.  
Chullora. Ph: 642 8922.  
We've moved!

SYDNEY:  
162 Pacific Hwy.  
Gore Hill. Ph 439 5311  
Ample parking at door.

SYDNEY:  
30 Grose St.  
Parramatta. Ph 683 1133  
1st floor — friendly store!

MELBOURNE:  
399 Lonsdale St.  
City. Ph 67 9834  
New: right in town!

MELBOURNE:  
656 Bridge Rd.  
Richmond. Ph 42 1614  
Easy access: huge stock.

BRISBANE:  
166 Logan Rd.  
Buranda. Ph 391 6233  
Opens 8.30AM

ADELAIDE:  
203 Wright St.  
City. Ph 212 1962  
Now Open. See us!

MAIL ORDER DEPARTMENT: PO Box 747, Crows Nest, NSW 2065. Phone 439-5311. Post & Pack extra.

WE HAVE DEALERS RIGHT ACROSS AUSTRALIA — THERE'S ONE NEAR YOU!

SHOP HOURS  
Mon-Fri 9AM - 5:30PM  
Sat. 9AM - 12 noon  
(Brisbane: 1 hour earlier)

**bankcard**  
welcome here

Order value	P&P charge
\$5 - \$9.99	\$1.00
10 - 24.99	\$2.00
25 - 49.99	\$3.00
50 - 99.99	\$4.00
\$100 or more	\$5.50



# INFORMATION CENTRE

found without exception a mass of unintelligible jargon. Is there any book which has a list of codes together with their binary equivalents and an explanation in readable English of what they are supposed to do? If so I would be very grateful if you could let me know. (R. H., Alphington, Vic.)

● We agree that there do not seem to be any easily read up-to-date books on machine language programming, which make no assumptions regarding the background knowledge of the reader. Most people have had to plough their way through many different books and articles, to gain even a modest understanding. And whether it would be a proposition for anyone to write such a book now is perhaps in doubt, as machine language may not be in wide use for much longer. Incidentally, there is no standardised machine instruction code used by all of the various microprocessors; virtually all of them use different codes.

---

## Notes & Errata

**MICROCOMPUTERS:** I would like to learn something about computers and I was thinking about getting the 2650 system described in your May 1978 issue. However I don't think I will because wherever I look I come up against an impenetrable wall of unintelligible jargon. I have looked in half a dozen books, pamphlets and introductions to various systems, "structured so that the user not familiar with computers can learn to generate code with a minimum of effort..." but have

**LOW COST VIDEO DISPLAY UNIT** (February 1978, File No 2/CC/23): In the circuit diagram on page 63, R9 and R6 should be interchanged, along with C11 and C8. The parts list and overlay diagram on page 65 are correct. Note also that the polarity of C19 is incorrectly marked on the overlay diagram.

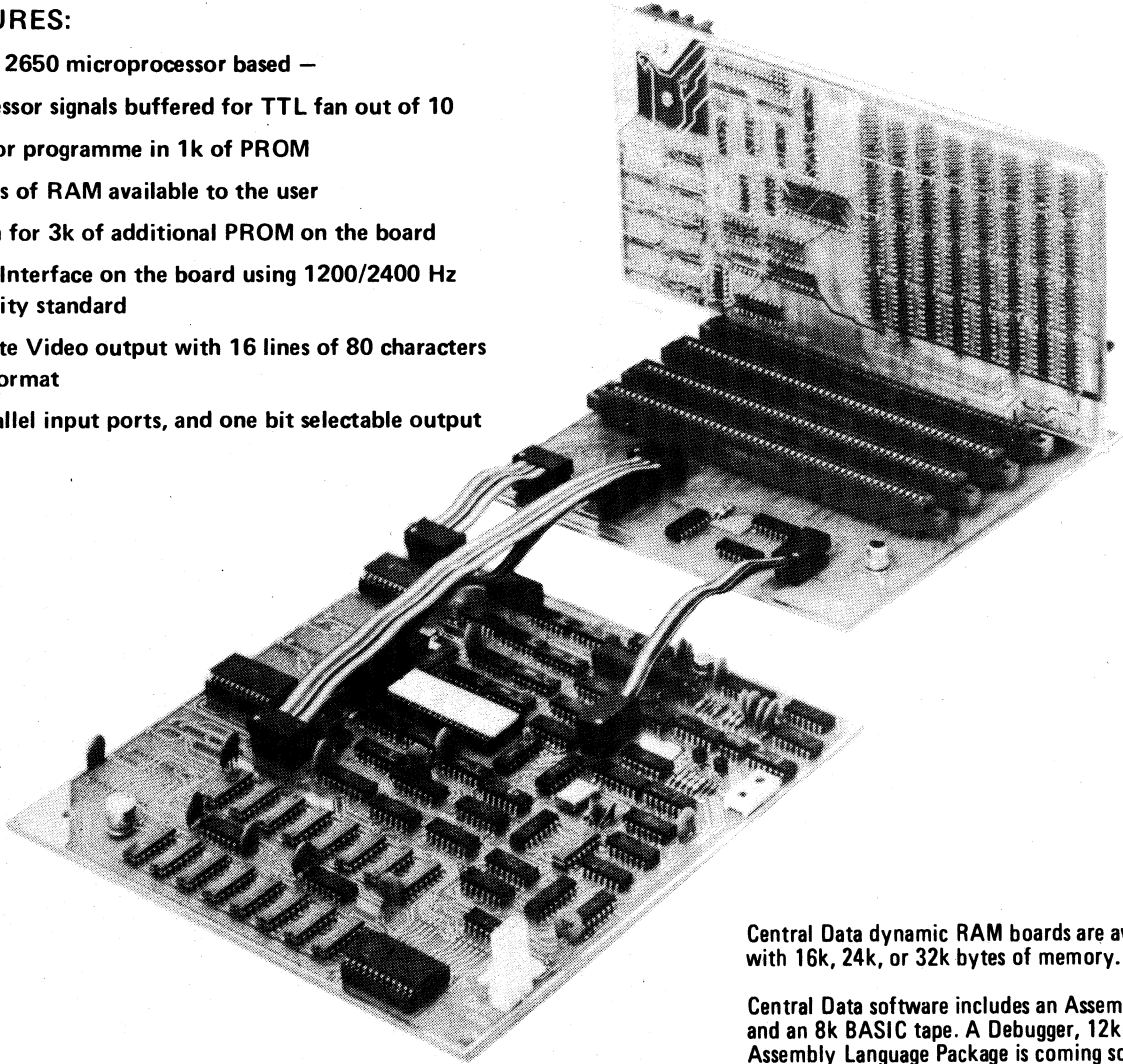
---

# Central Data

## Get into computing the economical, expandable way

### FEATURES:

- \* Signetics 2650 microprocessor based —
- \* All processor signals buffered for TTL fan out of 10
- \* Supervisor programme in 1k of PROM
- \* 730 bytes of RAM available to the user
- \* Provision for 3k of additional PROM on the board
- \* Cassette Interface on the board using 1200/2400 Hz Kansas City standard
- \* Composite Video output with 16 lines of 80 characters display format
- \* Two parallel input ports, and one bit selectable output port



Central Data dynamic RAM boards are available with 16k, 24k, or 32k bytes of memory.

Central Data software includes an Assembler/Editor and an 8k BASIC tape. A Debugger, 12k BASIC and Assembly Language Package is coming soon.

Other hardware now available and on the way includes Central Data Computer Mainframe with Power Supply, ASCII keyboard with solid-state low-profile keyswitches and +5 volt operation and Floppy Disc controller with one, two or three drives.

The Central Data System Board CDYSBDA facilitates the writing of programmes in Hexademical with only the addition of a TV monitor, ASCII Keyboard and power supply.

The System Board can be expanded by connecting the S-100 Board CDS100BDA. This allows you to plug in any S-100 static memory board with an access time of less than 500ns or the Central Data dynamic RAM boards CDXXKBDA.

For general and specific information:

**TECNICO ELECTRONICS**



Premier Street, Marrickville,  
N.S.W. 2204. Tel. 55 0411.

2 High Street, Northcote,  
Vic. 3070. Tel. 489 9322.



***Add full buffering and parallel IO ports:***

# How to expand your 2650 system

Since the publication of our 2650 mini computer design in the May 1978 issue, we have had many requests for information on expanding this system. Here is the first of two articles in response to these requests. It tells how to add full address and data bus buffering, memory page decoding and four parallel input-output ports.

**by DAVID EDWARDS**

Before discussing how the 2650 Mini Computer can be expanded, let us first spend some time discussing the reasons for expansion. The basic design, as presented in the May 1978 issue, is limited in memory size to 5K total (1K ROM and 4K RAM), due to the bus driving capacity of the processor.

Of course, this is quite a respectable amount of RAM, and allows quite large machine language programs to be run. However, when one considers handling large amounts of data or text, or running a BASIC interpreter, one finds that it is not quite enough. TCT BASIC, for instance, which was reviewed in the September 1978 issue, requires 4K of RAM for the interpreter, another 1K for use as scratchpad memory, plus whatever RAM is required for user

program storage.

Thus, a definite need for extra RAM exists. The 2650 CPU can address a maximum of 32K of RAM, and now that 2114 RAM chips are more readily available, a memory of this size is quite feasible. In fact, to implement an 8K RAM board requires only 16 2114 chips, and these can easily be accommodated on a single board.

The second main limitation of the original design is that the only external communication with the CPU is via the 20mA teleprinter interface. This limits not only the data transfer rate, but the number of devices which can be connected to the computer.

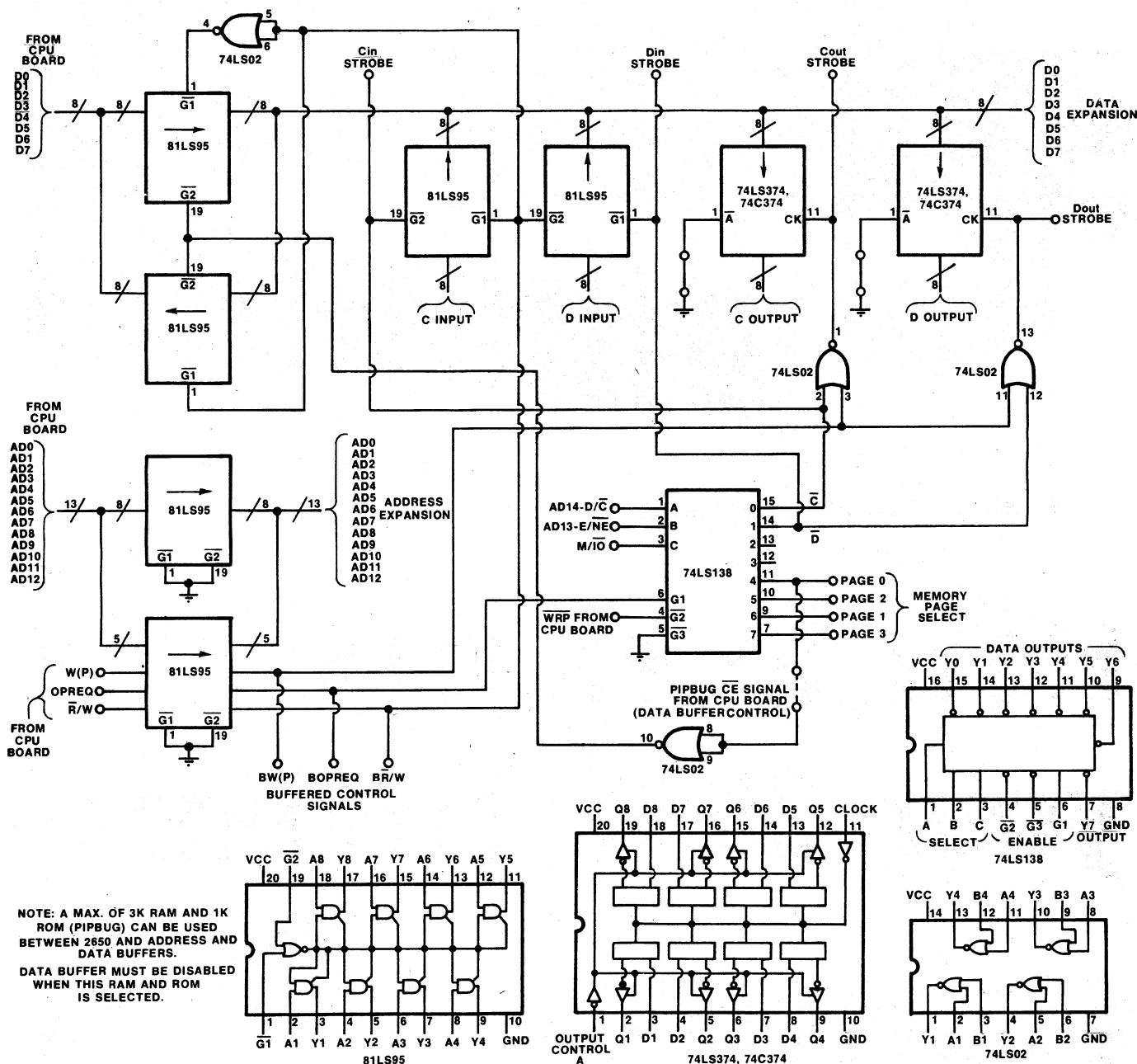
Of course, peripheral devices can be configured as memory, but this is not the approach which was taken by the

designers of the 2650 CPU chip. Instead, they chose to implement peripheral communications via dedicated input/output ports.

Of particular interest to a small system, such as we are interested in, are the "non-extended" I/O ports. These are accessed via four special instructions, REDD, REDC, WRTD and WRTC, which are all singlebyte instructions.

The non-extended I/O ports can be implemented as either two bi-directional ports, or as two separate input ports and two separate output ports. We have chosen the latter approach, as we felt it to be more flexible for a small system.

Provision of these output ports will enable the computer to communicate with high speed devices in parallel



2/CC/-

rather than serial format. Other uses include digital-to-analog and analog-to-digital conversion, as well as control of devices such as motors and lights, and the input of the system data.

Expansion of the 2650 Mini Computer thus involves the provision of I/O ports, and additional RAM. In order to provide these, it is necessary to buffer the address and data lines, so that overloading does not occur. We have designed two new circuit boards, one concerned with buffering the address and data lines, and providing the I/O ports, and the other concerned only with the provision of extra RAM.

Each board has the same dimensions and mounting holes as the main CPU board and is intended to mount in the case in the same way as the CPU board.

No power supply components are included on either board, and connections between the boards are made using rainbow cable.

The expansion-board, coded 78up9, contains the address and data bus buffering, the I/O ports, and memory "page" decoding. The RAM board, coded 78up10, contains 16 2114 RAM chips, as well as optional address and data buffers.

The expansion board may be powered from the existing power supply, while the RAM board requires an additional power supply. Details of the RAM board, and the required power supply will be given in the following article. In this article we will give full constructional details of the expansion board.

Turning now to the circuit diagram, we can discuss the expansion board in more detail. National Semiconductor 81LS95 octal Tri-state buffers have been used to implement the data and address buffers, as well as the two input ports. The output ports are implemented with 74LS374 or 74C374 Tri-state octal latches. Page selection and I/O port selection is achieved using a 74LS138 one of eight decoder.

The 81LS95 device is most probably unfamiliar to most readers, so a brief digression concerning it will no doubt clear up a few doubts. This device is packaged in a 20 pin DIL package, and has eight non-inverting buffers. Two control inputs are provided, G1-bar and G2-bar. If either or both of these inputs is at a logic 1 (high) level, the

# How to expand your 2650 Minicomputer

outputs of the buffers are placed in a high impedance or Tri-state mode.

This facility is used to enable several devices to be wired in parallel on the one bus. Only one device drives the bus at any time, with all other devices switched into the Tri-state mode. This should become clearer if we now turn to Fig. 1, a simplified representation of the bi-directional data bus.

When the CPU is transmitting data to the RAM or the I/O device, the R-bar/W line is high. This disables the output of buffer B, and places it in the high impedance state. Buffer A, however, is enabled, because its output enable signal is low. So the data placed on the bus by the CPU is transmitted to the RAM. The output of buffer B does not load the bus, because it is in the high impedance state.

Similarly, when the RAM or I/O device is sending data to the CPU, buffer A is disabled, and buffer B transmits the information.

Returning to the main circuit diagram, we can now discuss the address and I/O decoding. The control signals for the non-extended I/O ports are multiplexed together with the two high Jer 2650 address lines, AD13 and AD14. A control signal, M/IO-bar, is provided to enable them to be separated. These three signals are applied to the A, B and C inputs of the 74LS138.

Control input G1 is driven by the buffered OPREQ signal, while input G2-bar is driven by the inverted WRP signal. This latter signal is obtained from the CPU board after the modifications detailed in the box accompanying this article have been carried out.

The "O" output from the 74LS138 then becomes the C port select line, while the "2" output becomes the D port select line. These signals are used to enable the output buffers of the 81LS95s used for the input ports, so that information at the port inputs can be transmitted to the CPU.

The same signals are also gated with the buffered W (P) line, and are used to clock the two output port latches. This clocks the valid output data on the data bus into the appropriate output latch. The data is then available for peripheral devices until fresh data is clocked in. The output enable lines of the latches

have been earthed via links, so that they can be strobed externally if necessary.

We have specified two pin-compatible ICs for the output port latches. The low power Schottky devices (74LS374) were our first choice, but there may be supply problems with these. However, the CMOS devices can

be used, and are available.

The four high-order outputs of the 74LS138 decoder form the page-select lines. These go low whenever an address within the relevant page is selected, and are used to select the appropriate 8k block of memory.

A spare gate from the 74LS02 used for decoding the output port control

*This diagram shows the way in which the bidirectional data buffer operates. Only one of the eight separate buffers is shown.*

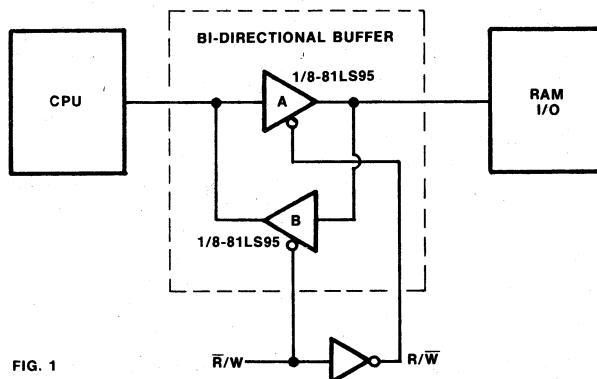


FIG. 1

## Modifications to CPU board

A possible timing conflict exists in the data bus of the 2650 Mini Computer System, described in the May 1978 issue. We have had no reports of this conflict causing operating faults, but recommend that the following modifications be carried out to eliminate this possibility.

The modifications require the use of an extra logic inversion, so this is provided by utilising the gate previously used as the interrupt request buffer. This avoids the need for an extra gate package, although it does mean that buffered interrupts are not available.

The alterations involve re-routing the WRP signal from the 2650 so that it is not gated with the R-bar/W signal, but instead is used to gate the 74LS138 address decoder. An inverter is required to do this, as only active low inputs are available as unused inputs on the 74LS138.

The modifications required, and the order in which they are best carried out, are listed below:

1. remove the links earthing pins 12 and 13 of the 74LS38 quad buffer.
2. remove the 10k resistor connected to pin 12 of the 74LS38.
3. remove the link earthing pin 4 of the 74LS138 address decoder.
4. cut the track leading to pin 9 of the 74LS38, and join pins 9 and 10 of the 74LS38 together with a small piece of tinned copper wire (used component lead).
5. cut the track leading to pin 11 of the 74LS38 at pin 11.
6. add a 1k resistor on the copper side of the board between pin 11 and pin 14 of the 74LS38.
7. connect pin 11 of the 74LS38 to pin 4 of the 74LS138.
8. connect pin 12 of the 74LS38 and pin 22 of the 2650. Pin 22 is available near pin 9 of the 74LS38.
9. check that steps 1 to 8 have been carried out correctly, and that there are no short circuits or solder bridges between IC pins.

We have also had a small number of reports of ringing on the clock input to the 2650, causing faulty operation. This ringing, if present, can be eliminated by connecting a 22pF ceramic capacitor between pins 5 and 8 of the 74123 device.

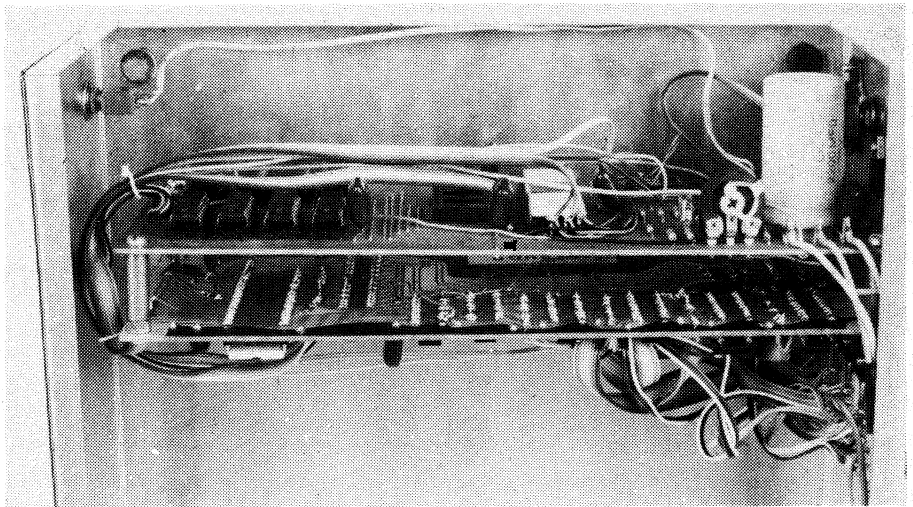
Finally, we would like to note that in theory both of the problems mentioned above may occur with the "baby" 2650 system described in the March 1977 issue, although there have been no reports of trouble.

Note also that the 1.5uF capacitor in the sense line should be reduced to 0.47uF for operation at 300 baud.

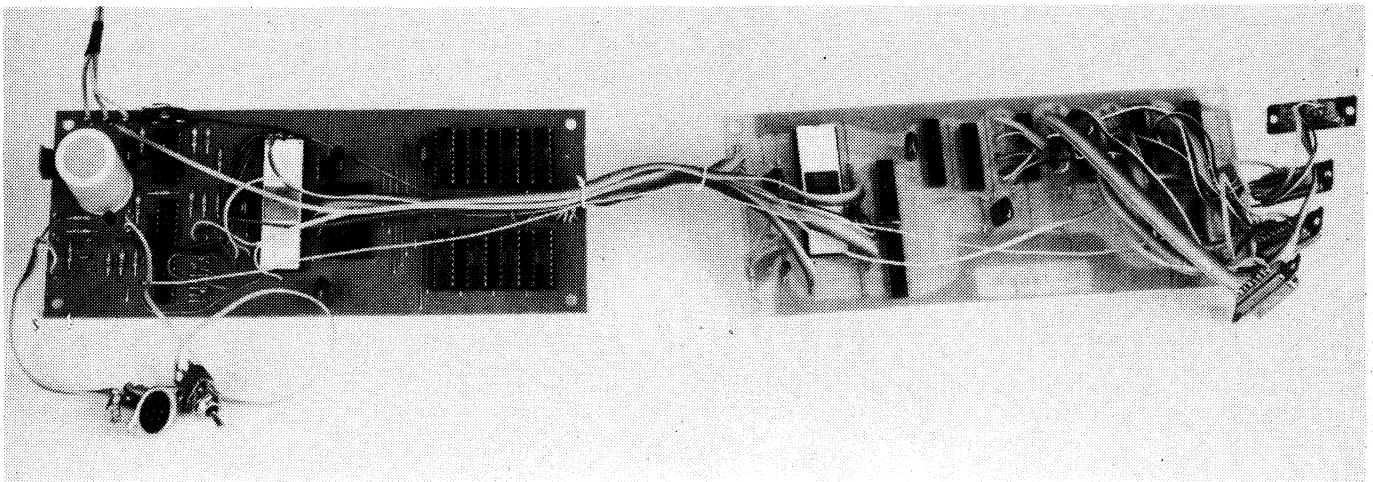
## PARTS LIST

- 6 81LS95 octal Tri-state buffers
  - 2 74LS374 or 74C374 octal Tri-state latches
  - 1 74LS138 decoder
  - 1 74LS02 quad NOR gate
  - 1 PCB, coded 78up9, 218 x 81mm
  - 8 0.1uF polyester capacitors
  - 4 15-way chassis mounting sockets and plugs to suit
- Solder, tinned copper wire, rainbow cable, machine screws and nuts, tapped spacers, PCB pins

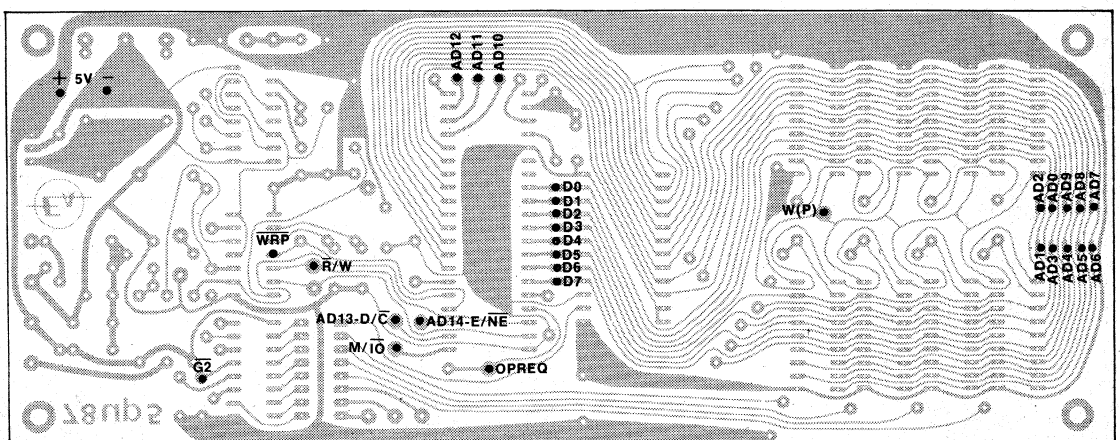
**NOTE:** Resistor wattage ratings and capacitor voltage ratings are those used for our prototype. Components with higher ratings may generally be used provided they are physically compatible.



These two photographs show how the two boards are assembled in the case, and how they appear when removed. Arrange the boards as shown in the lower photograph while completing the interconnections.



**RIGHT:** Use this overlay diagram of the CPU board to find the appropriate connection points. You may also need to refer to the overlay diagram on page 57 of the May 1978 issue.



signals is used as an inverter so that the data buffer can be disabled whenever page 0 is selected. This is required to prevent a conflict, as the Pipbug ROM and existing RAMs are on the CPU side of the buffer. Without this disabling, the data buffer and RAM or ROM would both try to drive the data bus

whenever a read operation in page 0 was performed.

The page 0 signal is coupled via a link, so that a different configuration can be achieved if desired. The reasons for this will become clearer in the next article.

Note that strictly only 3K of RAM can

be inserted on the CPU board when the expansion board is connected, due to bus loading limitations. This presumes that all devices present their maximum specified load. However in practice we found it possible to insert the fourth pair of RAMs, and still have correct operation.



# How to expand your 2650 Minicomputer

Before commencing construction, we recommend that you carry out the modifications detailed in the box. The following constructional hints assume that this has been done.

The circuitry is all contained on a single-sided board, coded 78up9, measuring 218 x 81mm. Note that the board has provision for a 40-pin IC (a possible future addition), but this is not used at present.

As you can see in the photographs, the new board mounts parallel to the CPU board. To do this, we removed the nylon standoffs used initially, and replaced them with tapped spacers screwed either side of the existing mounting bracket using a short length of threaded rod.

By arranging the cabling between the two boards in a suitable way, it is possible to remove both boards as a unit from the case, with a minimum of unsoldering. This is made easier if the TTY socket and the I/O sockets are mounted from the inside of the case, so they can be removed by simply unscrewing them, without unsoldering.

While this mounting method is a little awkward, and not conducive to extended servicing, it does leave enough space for at least two additional boards, and it is economical (motherboard systems have a high initial cost).

The existing transformer and power supply on the CPU board have enough reserve capacity to run the additional board as well, so no mods are required in this area. The power supply rails from the new board are simply connected in parallel with those on the CPU board.

Commence construction by fitting the I/O sockets, and by arranging the mechanical supports for the board. Make sure that the support pillars do not short to any of the tracks on the board (use insulated pillars if necessary). Then remove the board from the case, and commence to fit the components.

A low wattage, chisel pointed iron will be required, as well as some fine resin filled solder. A good light will also be necessary.

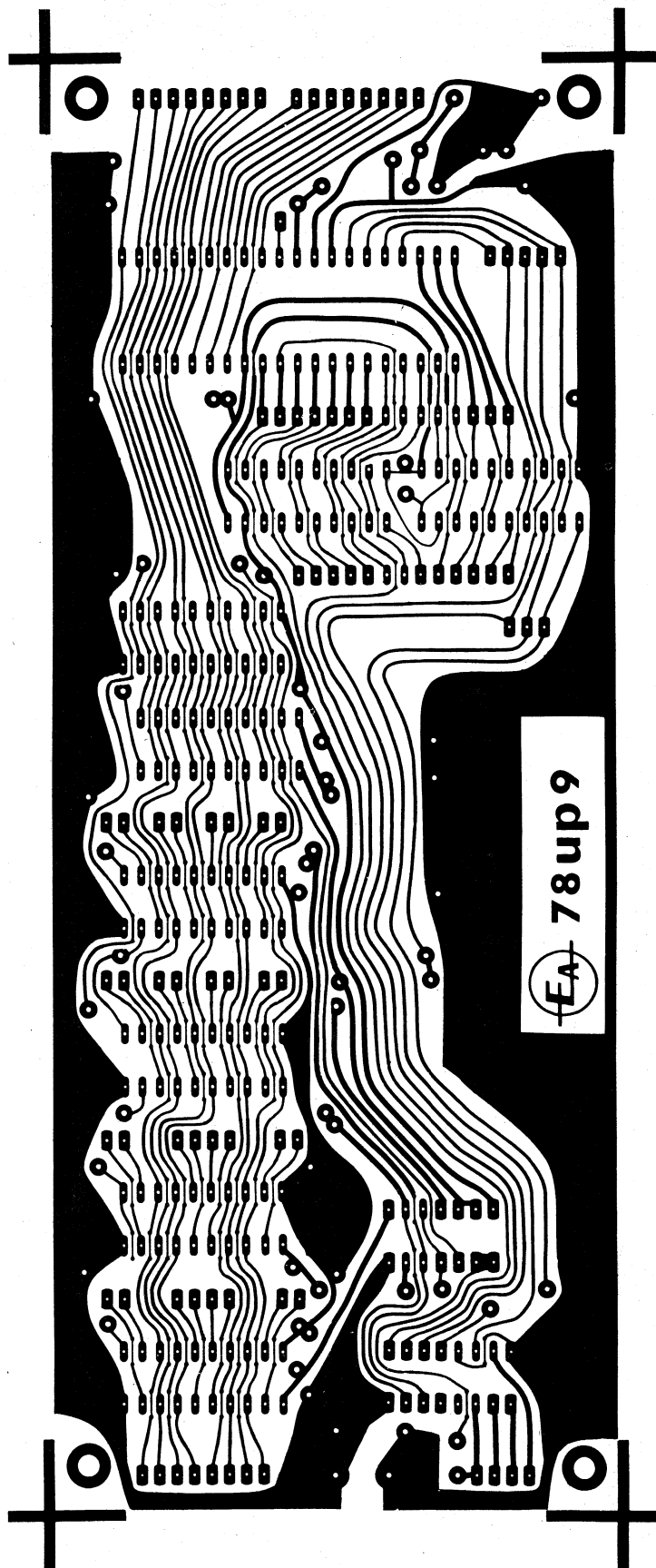
Fit all the passive components and links first. Leave the bypass capacitor between the 81LS95 address buffers out at this stage, and fit it only after the ICs have been inserted. Sockets are not required for the TTL ICs (ie, all but the 74C374's, if used) provided you are careful with your soldering iron.

Once all components have been fitted to the board, inspect it very carefully to check for solder bridges, wrongly oriented ICs or other similar faults. We do not recommend PCB pins for most of the inputs and outputs to the board, as there is not sufficient room for them.

Now remove the CPU board from the case, and place it in position on your work bench next to the expansion board, with the two "front ends" adjacent to each other. Using suitable lengths of rainbow cable, and using the two overlay diagrams as a guide, make the required interconnections. Remember to leave enough length so that the two boards can be folded copper side to copper side, and then assembled in the case.

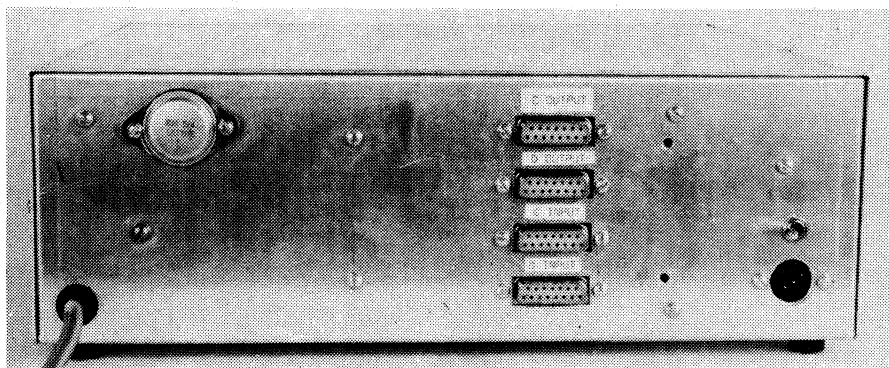
Take your time when making the interconnections, as any mistakes will be difficult to correct later, as well as being difficult to trace. It will be easier if you make use of the colour code of the rainbow cable, and make say the DO line back, the D1 line brown and so on.

Insert the link to disable the data buffer whenever page 0 is selected, and connect the page 0 line to the G2-bar input of the 74LS138 on the CPU board.



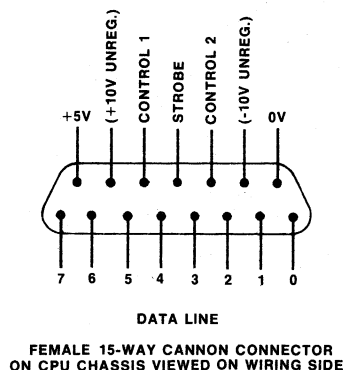
Here is a full sized reproduction of the PCB artwork. Commercial boards should be also available.

# How to expand your 2650 Minicomputer

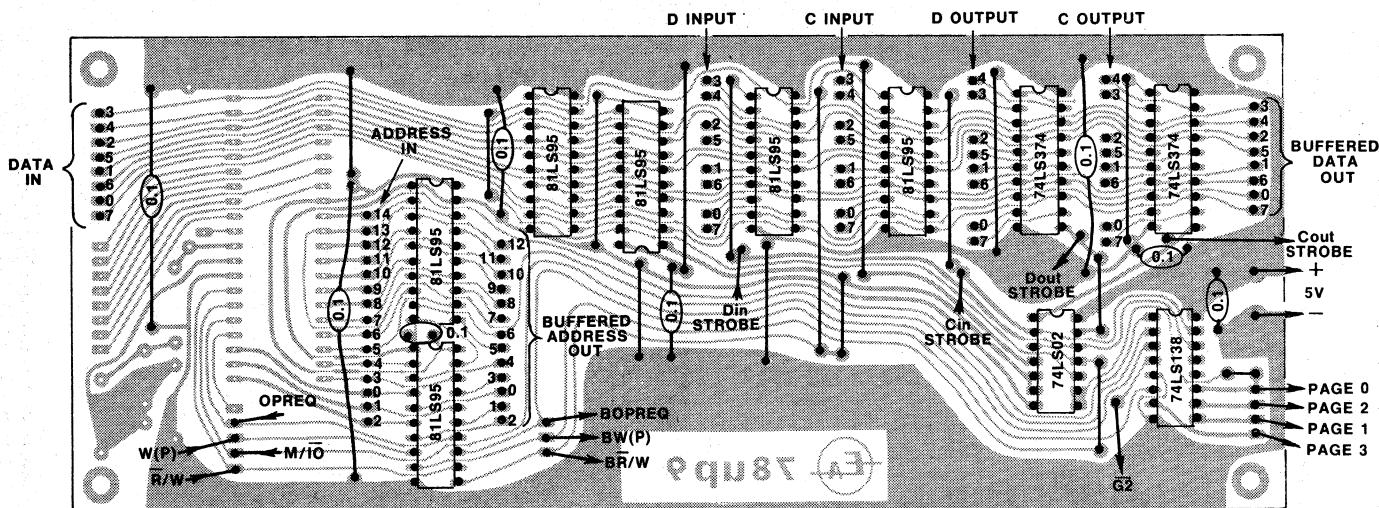


ABOVE: The position of the IO sockets can be scaled from this photograph. Mount them from the inside of the chassis.

BELOW: The overlay diagram is used both to place the components on the PCB, and to make the connections between the PCBs.



ABOVE: This is the suggested wiring diagram for the 15way Cannon connectors we used for the IO sockets.



The wire leading from pin 11 of the 74LS38 (the WRP-bar signal), must be extended to the G2-bar input of the 74LS138 on the expansion board.

The two power supply rails are available near the three terminal regulator. Make sure that you do not get the polarity wrong.

Once you have finished and checked all the interconnections between the two boards, you can wire up the I/O sockets. We used 15-way Cannon D Subminiature plugs and sockets. The connection scheme we used is shown in the accompanying diagram.

The eight pins in a row are used for the data connection, with the remaining seven pins used for control and power supply signals. The +5V and 0V signals are the main supply rails, while the +10V and -10V signals can be derived from the power transformer using a bridge rectifier and two electrolytic capacitors.

The strobe signal is obtained from the expansion board, while the control signals can be selected from the other

ports as desired. For example, to run the OP-80A paper tape reader, a ninth data input is required, so this can become one of the control signals.

Once all the wiring is completed, and has been checked thoroughly, testing can commence. This can be done before the board assembly is replaced in the case. Fit a heatsink to the regulator tab, and reconnect the three wires from the transformer.

Monitor the +5V rail, and switch on. If it does not rise immediately to the correct value, switch off and trace and rectify the fault. Assuming all this is well, connect up your terminal, and check that Pipbug and whatever RAM is fitted is working correctly.

The output ports can be tested by using a small routine of instructions to write data to them and checking that it appears on the appropriate output pin with a multimeter. You can use the Pipbug BIN routine to get data bytes from your terminal, the BOUT routine to echo them, and the WRTC or WRTD instructions to transfer them to the out-

put ports.

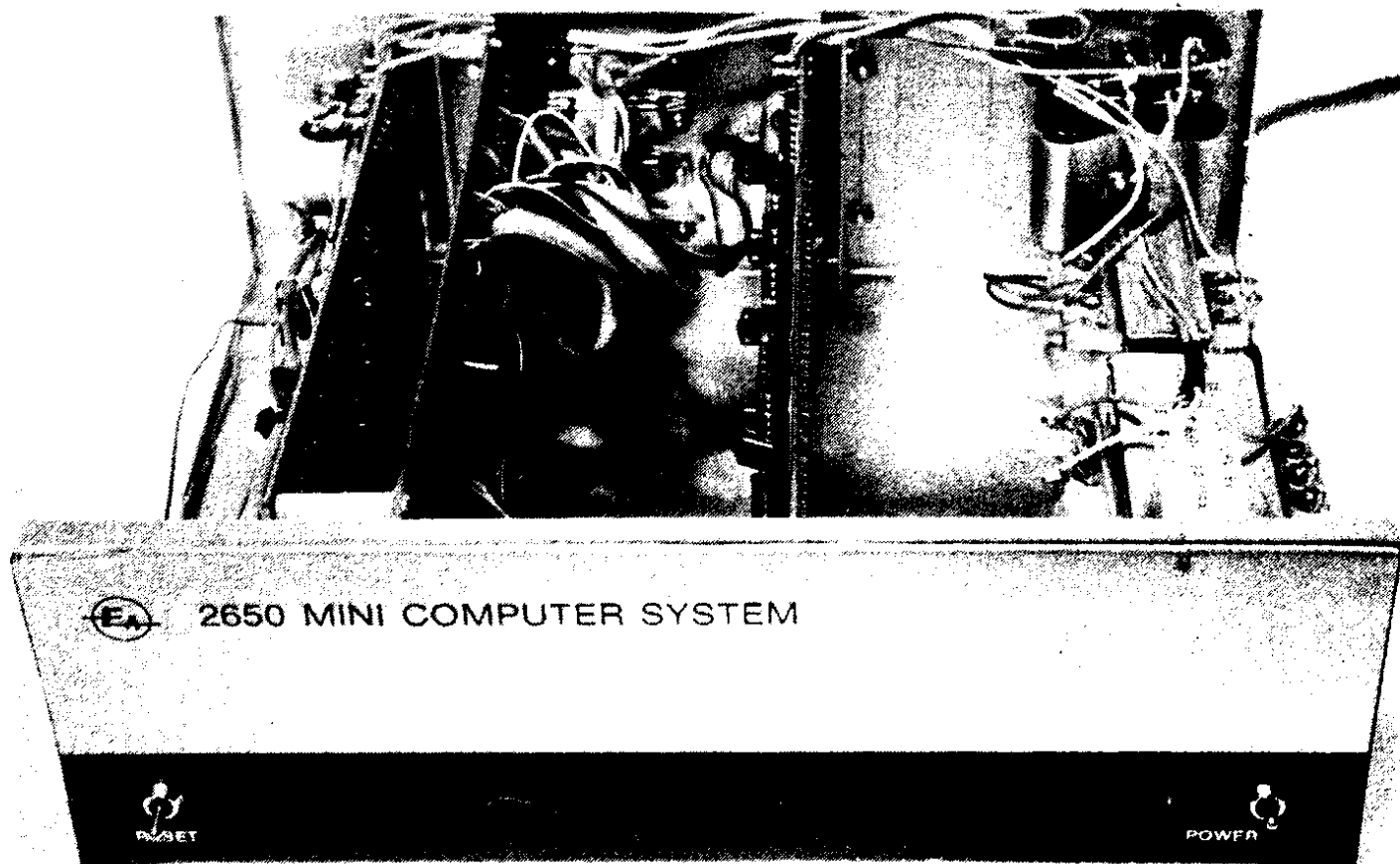
Similarly the input ports can be tested by reading them via REDC and REDD instructions, and using the BOUT routine to print the byte obtained on the terminal. With nothing connected to the inputs, you should get hexadecimal FF from both ports.

Now use a clip lead to ground one input pin at a time, and check that the appropriate byte is displayed. For instance, shorting the bit 0 data input should change the display from FF to FE.

Once you are satisfied that all facets of the unit are operating correctly, the board assembly can be fitted to the case. Use of a magnetised long blade screwdriver will be found helpful in this operation, if you are using steel screws.

In the next article, we will give details of the 8K RAM board and the additional power supply components required with it. This additional supply will also provide the +10V and -10V unregulated voltages mentioned previously.





# Extra RAM for the 2650

In this second article giving expansion details for the 2650 Mini Computer System, we give details of an 8K RAM board based on 2114 static RAM chips. Optional address and data buffering is provided, as well as full address decoding.

by **DAVID EDWARDS**

The 2114 static RAM chip, which forms the heart of the memory system, is only a relatively new device. These are 4096-bit devices, organised as 1024 4-bit words. Access time is 650ns or better, and all inputs and outputs are TTL compatible. They are packaged in 18-pin DIL form, and require only a single 5V supply.

Specified maximum power supply current is 100mA, with typical devices drawing about 80mA. This implies an 8K array would require a supply current in the vicinity of 1.5A, and that the dissipation in the RAMs would be about 7.5 watts.

The new board described in this article holds a maximum of 16 2114 chips, as well as five buffer and housekeeping chips. It is intended primarily for use with 2650-based systems, but can be

adapted for use with other microprocessors. Optional address and data buffers have been provided, as well as on-board address decoding.

Turning now to the circuit diagram, we can discuss the circuit in more detail. The optional data buffer is provided by two 81LS95 octal Tri-state buffers, wired as a bi-directional buffer. The direction of the buffer is controlled by the read/write line.

Low cost 74LS04 hex inverters are used as the address buffers, with two spare inverters used to buffer the read/write line. These inverters also provide the required control signals for the data buffer.

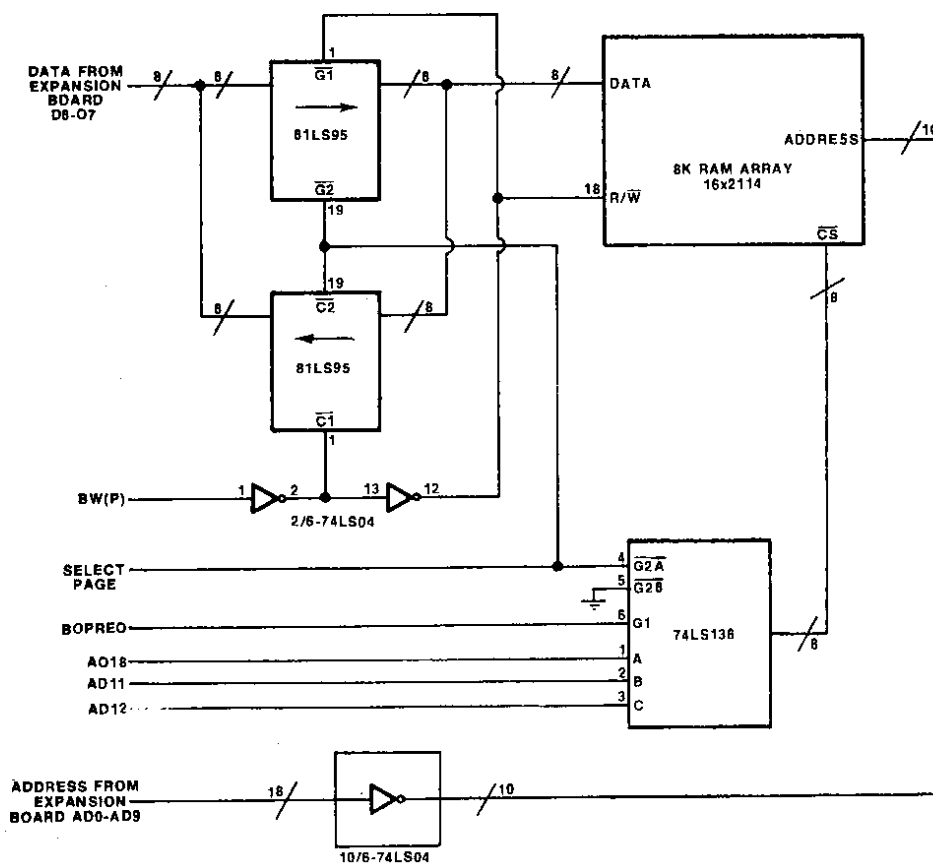
The RAMs are connected in pairs to form 1K blocks, with half of the data lines going to each chip. The two chip enable lines for each pair are con-

nected together, giving a total of eight active-low chip enables.

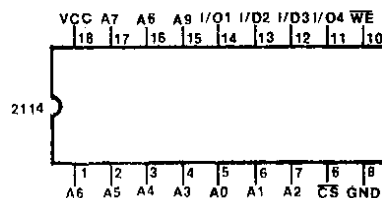
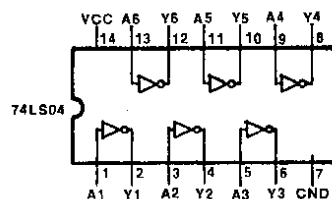
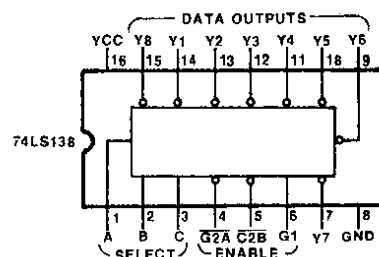
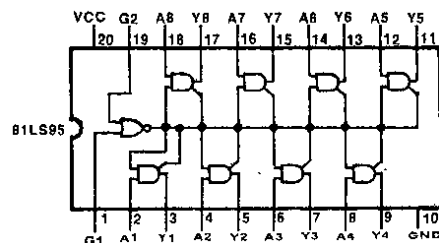
These are controlled by the 74LS138 decoder, which decodes address lines 10, 11 and 12. The 74LS138 is gated on and off by the page select signal and the buffered operation request (BOPREQ) signal. The page select line is generated on the expansion board, and is derived from address lines AD13 and AD14. It is also used to disable the data buffer when the page concerned is not selected.

The buffers on the expansion board described in the October 1978 issue are capable of driving at least one of these RAM boards without the use of the additional buffers. In this case, all that is required to support the RAM chips is the 74LS138 decoder.

Since the address lines are loaded



**2650 RAM BOARD**  
2/CC/-



more than the data lines (each data line connects to 8 2114s only, while the address lines connect to all 16), and since the cost of two 74LS04s is negligible compared to the RAM cost, it is probably worthwhile buffering the address lines.

On economic grounds, the extra cost of the data buffers can probably be justified also. Use of the buffers has the advantage that the system access time is not degraded by the extra bus capacitance that otherwise occurs.

Note that we have not specified in either the circuit diagram or the PCB overlay diagram the order in which the data lines and the ten lowest address lines should be connected. This is because it is immaterial which way they are connected.

All that is important is that each address should define a unique memory location, so that data is not lost or destroyed when reads and writes are performed. It is for this reason also that the logic inversion occurring in the address buffer is allowed. All the inversion does is shuffle the actual RAM storage locations about, without actually losing any of them.

It is important, however, that AD10, AD11 and AD12 be connected in the correct order, so that the memory chips

can be fitted in pairs. This allows the memory to be filled in increments of 1K, so that the cost of the RAM chips can be spread over time.

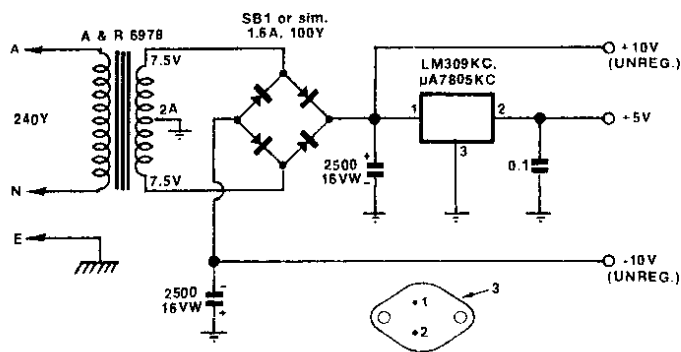
Before discussing the constructional details, let us first return to the power supply requirements. The maximum theoretical current requirement for a fully populated board is  $16 \times 100\text{mA} + 2 \times 26\text{mA}$  (81LS95)  $+ 2 \times 4.5\text{mA}$  (74LS04)  $+ 11\text{mA}$  (74LS138) = 1673mA. Obviously, if it is intended to run several RAM boards, the best approach is to use an external 5V supply, such as the Minibrite described in the November 1977 issue.

If only one RAM board is to be

driven, things become a little more difficult, however. The maximum current required is in excess of that which can be obtained from a single three terminal regulator. However, in practical cases, the actual current will be somewhat less than this.

If fact, the measured current consumption of the prototype board, fitted with full buffering and 14 2114 chips was only 720mA. So use of a standard three-terminal regulator should be possible in most practical cases.

The suggested power supply we have shown is based on the use of a TO-3 encapsulated three terminal regulator, on the basis that heatsinking of these



devices is easier. In our particular case, construction was also simplified.

We used an additional 15V 2A transformer and an encapsulated bridge rectifier, along with two 2500uF 16VW electrolytic capacitors. The bridge rectifier is not strictly required, as the centre tap of the transformer is earthed. However, it allows a negative supply rail to be developed, and this can be used to power devices attached to the non-extended I/O ports, as detailed in the October 1978 issue.

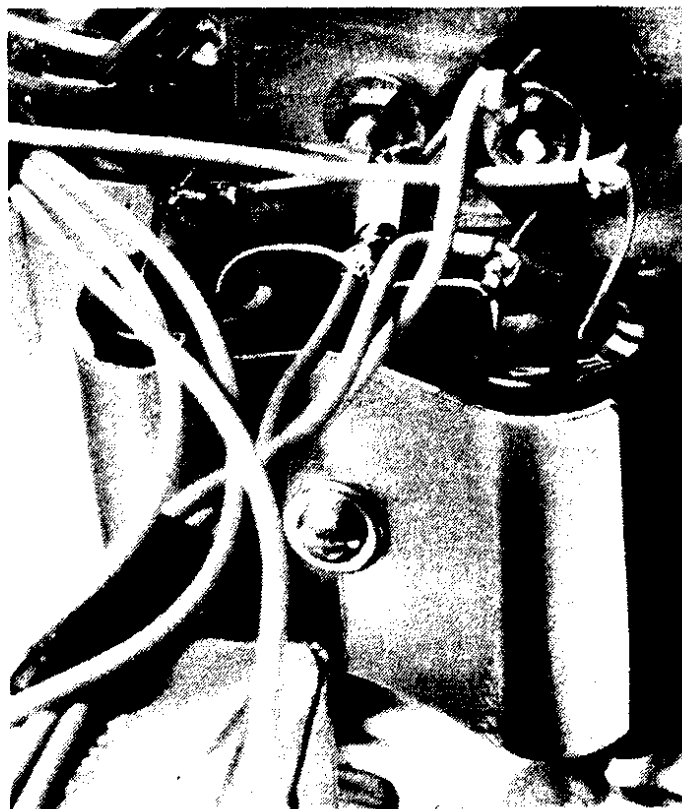
A second advantage of the encapsulated bridge is that it simplifies the mounting arrangements, as it can be bolted directly to the chassis. As you can see in the photographs, we mounted the additional transformer between the front panel and the original transformer.

The capacitors are clamped to the bottom right hand corner of the back panel (use PC mounting tubes), with the bridge and regulator mounted above them. The wiring can be completed by suitably bending the component leads. Use solder lugs for the chassis connections to the regulator.

Construction of the RAM board should be quite easy. The PCB is coded 78up10, and measures 218 x 81mm.

Fit all the links first, and then the RAM sockets. We recommend sockets for the 2114s as this allows them to be removed easily, or added in stages. Then fit the bypass capacitors, and finally the TTL ICs. Sockets are not required for these.

*At the right is a full size reproduction of the PCB pattern, which may be copied or traced. Commercial PCBs should be available in due course. Use the photograph below as a guide while wiring the power supply.*



When the board is complete, check it carefully for solder bridges and dry joints. The next step is to complete the connections between the RAM board and the expansion board. There are two different configurations which can be achieved, however.

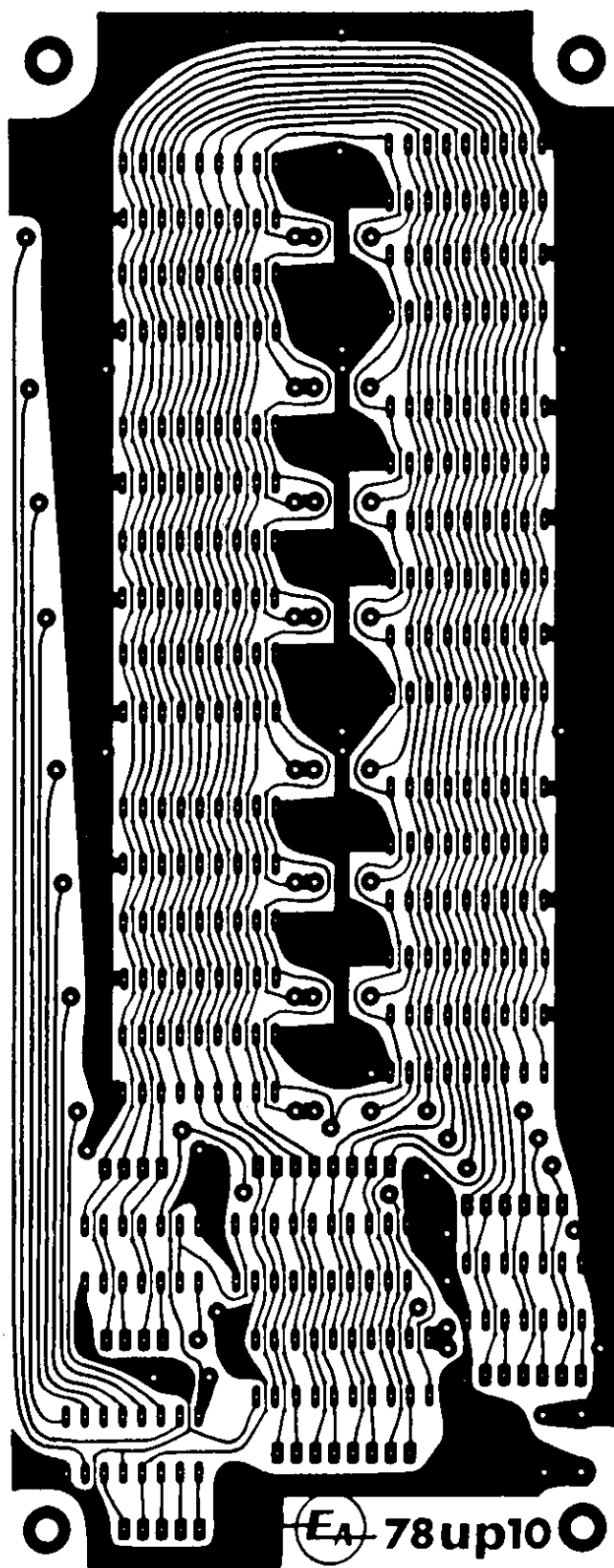
If you wish to maximise the amount of RAM, this can best be achieved by making the new RAM board page 1 of

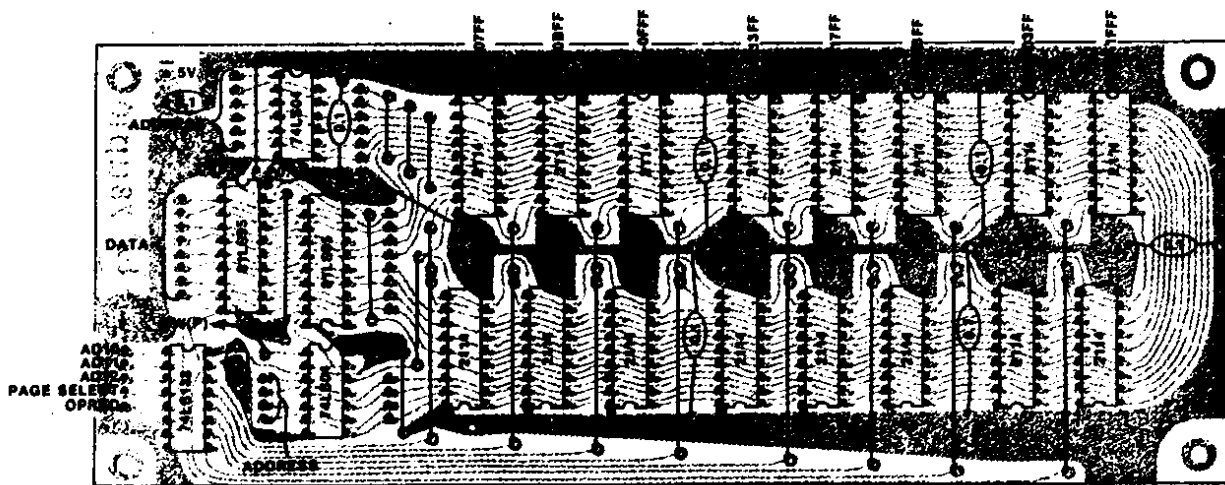
the memory. This will then allow at least 3K of RAM to be retained on the main CPU board. It does mean, however, that the memory will not be continuous, as page 0 cannot then be completely filled with memory.

If continuous memory is desired, then the best approach is to remove all RAM from the CPU board, and make the new RAM board exist at page 0. In

order to retain Pipbug, it will then be necessary to leave 1K of RAM vacant at locations 0000 to 03FF.

In order to prevent bus conflicts, it will then be necessary to disable the data buffer on the expansion board only when Pipbug is selected, rather than when page 0 is selected. This can be achieved by connecting the chip enable signal for Pipbug to the buffer,





## Parts List

- 16 2114 static RAM chips
- 1 74LS138 one-of-eight decoder
- 9 0.1uF polyester capacitors
- 1 PCB, coded 7Bup10, 218 x 81mm

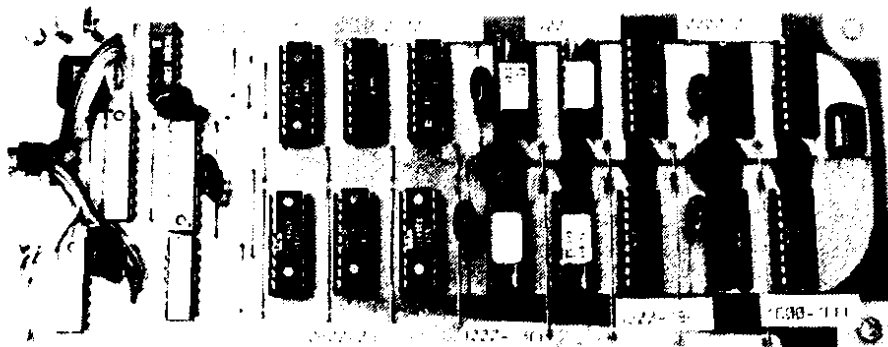
### OPTIONAL PARTS REQUIRED FOR BUFFERING

- 2 81LS95 octal Tri-state buffers
- 2 74LS04 hex inverters

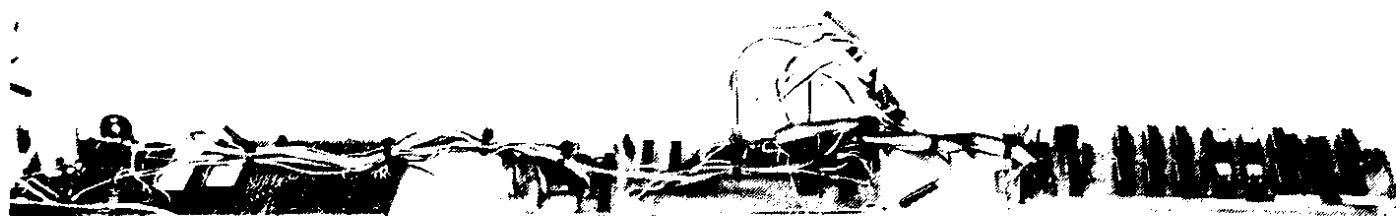
Solder, tinned copper wire, rainbow cable, mounting hardware, PCB pins  
 NOTE: Resistor wattage ratings and capacitor voltage ratings are those used for our prototype. Components with higher ratings may generally be used provided they are physically compatible.

ABOVE: Use this overlay diagram both to assemble the components onto the PCB, and to find particular 1K memory blocks.

BELOW: As you can see in this photograph, we actually labelled the RAM blocks with their addresses.



BELOW: The way in which the three PCBs can be removed from the case for servicing and the way in which the cables between them are arranged can be seen in this view.



rather than the page 0 select signal. This can be done by opening the link on the expansion board, and running a wire from the 0 output of the 74LS138 on the CPU board to the opened link.

In all other respects, wiring of the RAM board is identical for both cases, and is quite straight forward. Simply connect the appropriate control, data and address leads between the expansion board and the RAM board. If you do not use the address and data buffers, use the second set of address and data inputs.

Once construction is complete, test

the board before inserting the 2114s. Do this by applying power, while monitoring the supply rails. If they do not rise immediately to 5V, switch off and trace and rectify the fault. Once all is correct, plug in one pair of 2114s, and switch on, again monitoring the supply rail.

Test the RAM, and the address and data bus buffers, if fitted, by loading and running a small program from this area of RAM. Assuming all is OK, you can insert the remaining 2114s. These can be likewise tested by running programs known to be OK.

2114 RAM chips are available from Radio Despatch Service, of 869 George Street, Sydney, Dick Smith stores, Applied Technology of 109 Hunter Street Hornsby NSW, and from Pennywise Peripherals, of 19 Suemar Street, Mulgrave Victoria.

Please note that under normal conditions, the 2114 chips do dissipate significant amounts of heat, and become warm to the touch. For this reason, it is advisable to operate the unit in a well ventilated environment, to reduce the overall temperature rise of the case.

**New game programs for your 2650 computer:**

# Music player, Rotate and Conway's "Life"

Here are three very interesting programs for microcomputer systems based on the Signetics 2650, developed by a father and son team from Hobart. With them, you will be able to program and play your own computer music, play John Conway's game "Life", and test your mental endurance with the letter rearrangement game "Rotate".

**by PETER and HUGH CAMPBELL**

19 Brushy Creek Road, Lenah Valley Tasmania 7008.

The "Music" program occupies locations X'4A0 to X'5D3, and uses Pipbug routines. It contains absolute addresses, and is not easily relocated. The music is generated at the flag output of the 2650, and some form of audio transducer is required. This can simply be an audio amplifier and speaker, connected via a suitable attenuator, to the buffered flag output of the CPU.

Monotonic musical notes are generated by pulsing the flag output at suitable rates, with the program "reading" the music from a section of memory. The timing of the music is determined by a time value called "UNIT", which is an even number of up to 15 bits, such that X'5160 is about 1/32 of a second.

Each note is specified by two bytes. The first byte represents the number of UNITS that the note will last: X'01 gives a duration of 1 UNIT, while X'00 gives 256 UNITS, or 8 seconds with a UNIT value of X'5160.

The second byte is split into three fields. The most significant bit, bit 7, indicates either a note (0) or a rest (1). The next three bits, bits 4, 5 and 6, specify the octave. 111 represents the top octave, while 000 represents the lowest. In practice, the three lowest octaves are not usable, giving a range of only five octaves.

The remaining four bits in the second byte represent the note within the octave. The first note in any octave is E, represented by X'0, while the last note

is D sharp, represented by X'B.

For rests, bits 6 to 0 are not used, so all rests become X'80.

It is best to start and end all programs (tunes!) with X'80 80, a long rest, to separate the music from the noises Pipbug makes while communicating with the terminal. To signify the end of a tune, insert X'02FF after the long rest.

Fig. 1 is a hexadecimal listing of the program, as well as two tunes. "Yankee Doodle" occupies locations X'5D4 to X'6B7, and requires a unit value of X'2800, while "Bach" occupies locations X'6B8 to X'7A3, and requires a unit value of X'7000.

To run the program, type: G68C (address of first note) (value of unit) cr. The last two parameters are optional. If they are not given, the program will use the previous values. Thus to play "Yankee Doodle" type: G58C 5D4 2800cr; and for "Bach" type: G58C 6B8 7000cr.

The second program presented here is called "Rotate". The computer generates a 4 x 4 array of the first 16 letters of the alphabet, arranged in a random order. The object of the game is to rearrange the array into the following form:

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

The array can only be rearranged by rotating blocks of four letters

clockwise. The block to be rotated is specified by the letter in its top left hand corner. It is invalid to try to rotate by calling letters on either the bottom row or the right hand column of the array.

If a mistake is made, it can be corrected once between valid rotations. Any two adjacent letters can be exchanged, with the proviso that only one exchange is permitted. When the required pattern has been achieved, or when the game is aborted, the program will print out the number of moves used.

The program occupies locations X'440 to X'5C7, and uses routines from Pipbug. To run the program, type G440cr, and the computer will respond with "PRESS ANY KEY". Once this has been done, a random pattern will be generated and printed, and the prompt message "ROTATE:" given.

A sample game is shown in Fig. 2. If you wish to rotate a particular block, type the letter in the top left hand corner of that block. If you wish to cancel a move, type carriage return, and the program will respond with "CANCEL", and then reprint the last but one block.

If you wish to exchange two adjacent letters, type X. The program will respond with "EXCHANGE:", and expect you to type in the two desired letters. It will supply the comma separating the letters. If you cannot solve a particular pattern, type Z, and this will abort the game.

An average pattern, with only one exchange permitted, should take between 25 and 30 moves. Early attempts may take more. A hexadecimal listing of the complete program is given in Fig. 3.

The third and final program is the game of "Life", which is now well-known in computer circles. Life was originated by American computer programmer John Conway, and details of it were first published in the October 1970 issue of "Scientific American", in Martin Gardner's column

# LIFE: HEX LISTING

```

0000 76 40 75 F9 04 F9 08 27 12 1A 7D 3E 25 12 1A 78
0010 3E 1F 3E 1C 12 9A 04 05 59 1E 07 3E 13 12 1A 04
0020 05 F9 C9 0E 18 32 04 01 3E 0F 04 0A 3E 0B 17 59
0030 3E 00 0E 7H 14 14 F9 7D 17 77 10 05 08 3F 71 3E
0040 6F 74 40 3E 6B 50 1A 04 74 40 1E 03 76 40 14 F9
0050 72 76 40 3E 5E 75 10 17 1E 1C 77 18 05 00 06 08
0060 12 1A 7D 14 3E 4C 3E 48 12 1A E0 51 1A 78 3E 40
0070 45 7F 01 75 18 17 05 01 3F 0E 66 04 3A 3F 0C 39
0080 3F 0C 5A 1A 50 1C 0C E8 E4 47 1C 0C 8A E4 4E 98
0090 6F 3F 0C 39 05 46 2C 0C 4E E8 59 7B 05 FF 3F 0C
00A0 26 06 08 07 00 L3 3F 0C 5A E4 20 18 22 E4 4F 18
00B0 22 E4 0A 18 10 E4 0L 98 6D 3F 0C 39 85 01 45 FC
00C0 A5 01 1E 5D L3 FA 7E 03 0C 2E F4 65 03 1B 0F 04
00D0 2F 1H 02 67 01 3F 0C 39 FA 4E 03 0C 2E F4 F5 03
00E0 FC 0C A1 F5 3F 0C 0C 9E 05 FF 3F 0C 26 06 00 CA
00F0 15 07 08 0L 2E F4 1A 00 9A 02 CA 0A 0E FE 77 F5
0100 03 98 6E A5 04 06 00 98 06 85 04 1B 1E FB 09 07
0110 08 0L 2E F4 77 10 02 52 77 10 L2 1A 04 04 20 1H
0120 02 04 4F 3F 0C 39 FA 65 65 03 E5 3F 9C 0C EA 04
0130 0L 3F F1 0L 0E EF E5 09 99 04 A5 0A 1H 78 01 24
0140 30 3F 0C 39 01 0E E8 06 00 E5 09 99 0E 86 01 E6
0150 09 99 02 06 00 A5 0A E5 09 19 72 02 24 30 3F 0C
0160 39 01 24 30 3F 0C 39 1F 0C 7E 3E 0B 07 09 C2 82
0170 FF 7D C2 3E 02 82 17 3F 0C 5A E4 30 1A 79 E4 39
0180 19 75 C3 3F 0C 39 03 44 0F 17 3F 0C 39 3E 5E 0C
0190 0E EL 05 0C E4 0C 1C 0E B0 05 15 E4 0F 1A F8 05
01A0 1F E4 37 1A F2 05 21 E4 4C 1A E0 05 27 1E E8 05
01B0 04 06 04 0L 4E F4 C4 E4 34 0E 6F 30 0C 6E F0 5A
01C0 02 06 04 59 6E 05 FF 07 03 85 03 3E 33 08 2F 50
01D0 50 C8 2A 3E 14 A5 03 3E 27 08 22 C8 22 3E 0A 0D
01E0 2E F4 77 10 C1 07 0E 1B 3E 0A 13 08 10 CA 0E 0D
01F0 10 C8 JE 44 03 CF 6E 20 F8 02 07 03 17 00 00 00
0200 2C 0C 7F C8 79 06 03 0D 2E F0 44 55 88 70 C8 6E
0210 0L 6F F0 5C 44 55 88 65 C8 63 85 03 1A 69 A5 0C
0220 17 00 00 00 FF 0E 75 10 F5 03 18 04 3E 52 1E 04
0230 08 4D C8 49 75 10 3F 0D E9 77 10 08 64 88 63 88
0240 62 E4 04 18 10 E4 03 18 05 01 1A 05 1E 07 01 1A
0250 04 66 01 25 80 L1 5E 4C 75 10 01 44 03 02 0E 6E
0260 FC 0C 6E F0 77 10 C1 75 10 0E 6E F0 F5 03 9C 0D
0270 LF E5 3F 9C 0L C9 05 04 0L 4E FC 0D 6F 30 59 78
0280 77 10 02 1C 0E A8 06 00 75 10 05 01 8D 0E EE E5
0290 64 98 0A 05 01 8L 0E EF 0E EF 05 00 0D 0E EE E5
02A0 0C 0E EL A4 01 0C 0E EF 0E EF 0E EF 0E EF 0E EF
02B0 3F 0E E6 1F 0L AF 0L 2E FF E4 00 14 3F 0C 39 1E
02C0 75 0L 0E 22 4C 49 46 45 22 05 3E 53 00 20 3C 31 35 53
02D0 35 36 47 2E 00 20 3C 35 53 00 20 3C 31 35 53
02E0 00 20 3C 32 30 53 00 20 3C 32 35 53 00

```

# ROTATE: HEX LISTING

```

0440 06 FF 3F 05 64 05 08 07 F5 1E 00 12 9A 06 D9 79
0450 DB 79 1H 71 06 05 66 80 F6 03 18 0A 26 FF F6 03
0460 18 02 66 80 26 FF 1E CA 6C 46 0F 3F 04 C4 18 64
0470 19 62 1H 60 07 0F 1F 0A 65 3F 05 44 06 0F 3F 05
0480 64 3F 02 86 E4 58 1C 05 03 F4 0D 18 0L 3F 04 E7
0490 58 5F 3E 30 98 1A 06 19 1B 6A 03 C2 3E 26 18 76
04A0 3E 22 3E 20 07 0F 06 21 3F 05 64 A5 01 95 1E 49
04B0 02 C3 85 67 95 05 40 0E 25 76 E2 18 7A 60 1C 05
04C0 56 1F 04 79 F6 03 14 F6 0C 14 0E 65 8B C8 13 0E
04D0 65 FC 0E 65 E8 0E 65 E8 0E 65 E8 0E 65 E7 0E 65
04E0 E8 04 4F 0E 65 E7 17 E4 5A 1C 05 58 E4 41 1A 12
04F0 E4 50 19 0E 06 10 EE 45 E7 18 04 5A 79 9E 22 3F
0500 02 B4 17 06 28 3F 05 64 3F 02 86 3E 5A 58 79 02
0510 C3 04 2C 3F 02 B4 3F 02 86 3E 4C 58 79 02 A3 9A
0520 02 03 A2 E4 01 18 08 E4 0A 18 04 06 19 1B 56 0E
0530 65 E7 C8 07 0F 65 E7 0E 65 E7 04 41 CF 65 E7 07
0540 0F 1F 04 E2 06 01 E2 25 B3 14 3F 02 B4 F6 03 98
0550 75 3F 00 8A 1B 7C 3E 6C 06 3A 3E 08 3F 02 69 3E
0560 03 1F 04 40 0E 25 6D 14 3F 02 B4 1B 77 0A 50 52
0570 45 53 53 20 41 4E 59 20 4E 45 59 00 0A 52 4F
0580 54 41 54 45 3A 20 00 0A 57 48 41 54 3F 20 00 43
0590 41 4E 43 45 4C 00 0A 45 58 43 48 41 4E 47 45 3A
05A0 20 00 0A 59 4F 55 20 54 4F 4F 4E 20 00 4D 4F
05B0 56 45 53 0A 00 0L 0A 4F 4L 44 49 45 4A 4C 50 48
05C0 46 41 4E 42 43 4E 47 00

```

# MUSIC: HEX LISTING

```

04A0 04 00 0C A5 56 1E 05 58 C2 44 70 24 70 C1 51 51
04B0 51 81 51 77 10 83 C3 86 00 75 11 46 0F D2 0E 65
04C0 BC C3 0E 25 EC 06 FF E5 00 18 05 D0 D3 L2 F9 7B
04D0 75 01 84 80 87 00 86 00 CA 22 CA 1F 77 10 87 44
04E0 86 01 9A 06 87 0F 86 00 1B 7A 0D 85 56 77 01 AB
04F0 09 AA 06 76 40 04 08 1B 31 28 00 F9 5A AB 7B AA
0500 78 93 07 8C 8B 76 8A 73 9A 06 87 0E 86 00 1A 7A
0510 53 13 53 E5 01 98 03 1B 01 C0 47 03 9F 05 1F C0
0520 C0 C0 93 12 24 40 92 13 77 11 AB 50 AA 4D 9A 05
0530 93 07 80 1B 4F F9 46 44 01 4C 04 FC 64 18 93 8F
0540 04 FC 8E 04 FB 75 11 0E 0E 0A 0B 87 02 86 00 CB
0550 06 CA 03 1F 04 0A 06 B6 77 10 E4 FF 1C 00 22 09
0560 F5 87 8A 86 00 9A 06 87 0E 86 00 1A 7A 74 40 75
0570 01 86 02 77 01 AF 04 FA AE 04 F9 87 1E 86 00 87
0580 0E 86 00 1A 7A F9 6E A6 02 1F 05 45 75 FF 3F 02
0590 DB 0C 04 2A 98 06 0A 23 09 20 1B 04 CA 1D C9 1A
05A0 CE 05 57 0L 05 56 3F 02 DE 77 08 0C 04 2A 1C 04
05B0 A0 CE 04 FA 0L 04 F9 1F 04 A0 05 D4 02 FC 11 2F
05C0 1E 97 2B 3E 37 2F 42 74 4D 17 57 22 60 9C 69 8E
05D0 72 00 79 F8

```

# YANKEE DOODLE: HEX LISTING

```

05D4 80 80 08 43 08 80 08 43 08 80 08 47 08 80 08 45
05E0 08 80 08 47 08 80 08 43 08 80 08 47 08 80 08 45
05F0 08 80 08 3A 08 80 08 43 08 80 08 43 08 80 08 45
0600 08 80 08 47 08 80 14 43 08 80 08 42 08 80 08 3A
0610 08 80 08 43 08 80 08 43 08 80 08 45 08 80 08 47
0620 08 80 08 48 08 80 80 47 08 80 08 45 08 80 08 43
0630 08 80 08 42 08 80 08 3A 08 80 08 40 08 80 08 42
0640 08 80 10 43 0C 80 10 43 10 80 08 40 10 80 08 42
0650 02 80 08 40 08 80 08 3A 08 80 08 40 08 80 08 42
0660 08 80 10 43 10 80 08 3A 10 80 08 40 02 80 08 3A
0670 08 80 08 38 08 80 10 37 10 80 10 3A 10 80 08 40
0680 10 80 08 42 02 80 08 40 08 80 08 3A 08 80 08 40
0690 08 80 08 42 08 80 08 43 08 80 08 40 08 80 08 3A
06A0 08 80 08 43 08 80 08 42 08 80 08 45 08 80 10 43
06B0 0C 80 10 43 80 80 02 FF

```

# BACH: HEX LISTING

```

06B8 80 80 06 43 06 45 04 47
06C0 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53 04 52
06D0 04 53 04 4A 04 47 04 43 04 45 04 47 04 40 04 4A
06E0 04 48 04 47 04 45 04 43 04 3A 04 43 04 42 18 43
06F0 0C 80 18 47 0C 48 18 4A 0C 4A 18 48 0C 47 18 45
0700 06 80 06 80 18 47 0C 48 18 4A 0C 47 06 45 03 47
0710 03 48 0C 47 0C 45 18 43 0C 80 02 80 06 43 06 45
0720 04 47 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53
0730 04 52 04 53 04 4A 04 47 04 43 04 45 04 47 04 40
0740 04 4A 04 48 04 47 04 45 04 43 04 3A 04 43 04 42
0750 04 43 04 47 04 4A 04 53 04 4A 04 47 04 43 04 47
0760 04 49 18 4A 06 80 02 80 03 80 06 43 06 45 04 47
0770 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53 04 52
0780 04 53 04 4A 04 47 04 43 04 45 04 47 04 40 04 4A
0790 04 48 04 47 04 45 04 43 04 3A 04 43 04 42 18 43
07A0 80 80 02 FF

```

Fig. 4 (top left): This is a hexadecimal listing of the Life program.

Fig. 1 (top right): The listing shown here is for the music program, and two tunes.

Fig. 3 (left): This is the listing for the game of Rotate. It uses routines from Pipbug.

1	2	3
4	X	5
6	7	8

The rules of cell life, death and birth are as follows:

1. A live cell will survive if it has two or three live neighbours.
2. A live cell will die if it has less than two or more than three live neighbours.
3. A birth in an empty cell will occur if it has exactly three live neighbours.
4. Births and deaths take place simultaneously.

To work with practical terminals the program operates with a limited size matrix, but makes it effectively "infinite" by having "wrap around" from side to side and from top to bottom.

In our version of Life, live cells are represented by O's, and dead or empty cells by blanks. The program starts with an initial pattern (fed in by the player), and calculates the new patterns "generation by generation".

"Mathematical Games". Further information was published in the November 1970, January 1971 and April 1971 issues of the same magazine.

Since it was first produced, Life programs have been developed by many different people (it is said to have been responsible for more "foreign order" computer time than anything

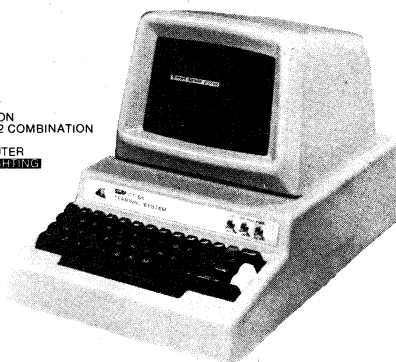
else). People all over the world have played Life, and come up with many interesting patterns.

Life is a matrix game concerned with the life, death and birth of cells. Imagine each cell to be in a two-dimensional linear matrix, such that each cell location has eight possible neighbours, as shown:

# MICROCOMPUTER

## CT-64

- 64 OR 32 CHARACTERS PER LINE
- UPPER AND lower case LETTERS
- FULL 8 BIT MEMORY
- 128 CHARACTER ASCII SET
- 110/220 Volt 50 - 60 Hz POWER SUPPLY
- SCROLLING OR PAGE MODE OPERATION
- CONTROL CHARACTER DECODING - 32 COMBINATION
- PRINTS CONTROL CHARACTERS
- USABLE WITH ANY 8 BIT ASCII COMPUTER
- REVERSED BACKGROUND - **INSTRUCTIONS**



IN KIT FORM.



FOR FURTHER INFORMATION PLEASE PHONE 31 3273  
OR WRITE TO:  
P.O. Box 380, Darlinghurst NSW 2010

## Music play

\*G440

PRESS ANY KEY

OJMD

EPLI

BFHK

CNAG

ROTATE: F

OJMD

EPLI

BNFK

CAHG

ROTATE: N

OJMD

EPLI

BANK

CHFG

ROTATE: N

OJMD

EPLI

BAFN

CHGK

ROTATE: F

OJMD

EPLI

BAGF

CHKN

ROTATE:

EXCHANGE: L,M

OJLD

EFMI

BAGF

CHKN

ROTATE: P

OJLD

EAFI

BGMF

CHKN

ROTATE: M

OJLD

EAFI

BGKM

CHNF

ROTATE: CANCEL

OJLD

EAFI

BGMF

CHKN

ROTATE: G

OJLD

EAFI

BHGF

CKMN

ROTATE:

YOU TOOK 07 MOVES

Fig. 2 (above). Here is a sample printout from the Rotate game. The "Z" command was used to terminate the game.

Fig. 5 (at right on facing page): This sample listing is an example of the "Life" program in operation. This cell pattern stabilises at generation four.



## e and Rotate programs for the 2650

The program listing provided (Fig. 4) is intended for use with the Low Cost VDU of February and April 1978, and uses a matrix of 16 rows of 32 cells. To use the program, type GC00 cr, and then switch to the appropriate baud rate (110 or 300 baud). The type a U for 110 baud operation, or a Y for 300 baud operation.

The program will respond with the word "LIFE", followed by the prompt character ":".

If you respond with "N", the program will expect a new matrix to be supplied. The program will echo the N, followed by a carriage return and line feed. A pattern may then be written in (or "seeded") by using the space bar for blanks (these are printed as dots), O's (for Oboe) for live cells, and line feeds (LF) for new lines.

Blanks are not required on the right hand side of the pattern. Carriage return (CR) will permit overwriting of a line, allowing error correction. Once your pattern is complete, use LFs if necessary to advance to the bottom of the matrix.

Once the pattern is completed, the program will reprint it, and give the prompt sign again. If you now respond with a Gxx, X'xx generations will be evolved, with a printout after the last generation. G00 will produce printout after 256 generations, while G01 will produce a printout after only one generation. And so on . . .

Immediately after you have typed in this command, the program will respond with a message such as <15S, to indicate that in less than 15 seconds it will print out the result of the Gxx instruction. After printing the result the new generation count and prompt will appear at the bottom left hand corner

```

                                0
GENERATION 1  0 0 0

                                0
GENERATION 2  0 0
                                0

                                0 0
GENERATION 3  0 0
                                0 0

                                0 0
GENERATION 4  0 0 0
                                0 0

                                0 0
GENERATION 5  0 0 0
                                0 0

```

of the screen. This may overwrite live cells, so try and keep your patterns in the centre of the screen (patterns to the right will wrap around to the left).

The remaining instruction is P, which causes the program to printout the existing matrix. The instruction is not used a great deal.

Fig. 5 shows the result of a simple pattern. This stabilises after four generations, and then continues forever unchanged.

One of the most interesting and simple patterns has been named the "Glider". The seed for this is shown below:

```

      0
      0
     000

```

Can you work out what will happen with this pattern? (the name is a good clue!).

If your VDU can cope with 24 lines, the program can be adapted to produce a 24 x 32 matrix. The EME-1 VDU, described in the January and February 1977 issues, is such a terminal.

To do this change the following locations:

location	C95	From	46	to	66
CE4	3F	5F			
D2B	3F	5F			
DB8	34	54			
DBB	30	50			
E72	3F	5F			
E7D	30	50			

The complete Life program occupies locations X'C00 to X'EEC inclusive, and requires additional RAM extending to X'F54. However, the first part of the program is a self-contained I/O module containing a baud rate initialisation routine and some subroutines which duplicate the functions of Pipbug's CHIN, COUT and CRLF subroutines. The I/O module may be used by other programs, either where it is or moved elsewhere.

The memory locations occupied by the module are from X'C00 to X'C75 inclusive. The baud rate initialisation routine begins at X'C00 and ends at X'C58-59 with a BCTR, UN instruction which currently produces a branch to the start of the main section of the Life program at X'C76. To make the routine branch to the start of another program instead, the displacement of this instruction would need to be changed, or possibly the instruction changed into a BCTA, UN type.

Incidentally although the I/O module at present offers a choice of either 110 or 300 baud operation, the higher rate may be changed quite easily to 1200 baud if you desire (and if your terminal will work at this speed). Simply replace the contents of location X'C18 (currently X'59) with X'14.

To use the baud rate initialisation

routine, type G C00cr. Then type U for 110 baud operation, Y for 300 baud operation (assuming the routine is set to give this alternative speed), or E for 1200 baud operation. Of course it is necessary to switch the terminal for the appropriate baud rate as well.

Note also that you may need to reduce the value of hash filter capacitor on the asynchronous input of your computer, in order to operate reliably at 1200 baud (or even 300 baud in some cases). In the case of the EA 2650 Mini Computer, the value of the capacitor should be reduced from 1.5uF to about 0.1uF.

Once it has selected the desired baud rate, the initialisation routine will branch to the desired main program, with the I/O subroutines set up for the correct baud rate.

The actual subroutines are used in exactly the same manner as those in Pipbug. The calling addresses are:

```

CRLF  X'C26
CHIN  X'C5A
COUT  X'C39

```

Needless to say, you can also run the Life program at 1200 baud, simply by making the above change to the I/O module with the output branch still pointing to X'C76.

However, if you are running Life at 1200 baud, it is better to change the program so that it prints out after every generation, and stops automatically when the pattern stabilises. To do this, change the following locations:

location	D68	From	0C	to	0D
D69	7B	A9			
DA9	1A	77			
DAA	EC	10			
DAB	05	06			
DAC	27	00			
DAD	1B	75			
DAE	E8	10			
EB4	0E	0C			
EB5	AB	7B			
EA0	0C	04			
EA1	0E	00			
EA2	E0	05			
EA3	A4	02			
EA4	01	06			
EA5	CC	02			
EA6	0E	F8			
EA7	ED	7E			
EA8	E4	F9			
EA9	00	7C			
EAA	1C	FA			
EAB	0C	7A			
EAC	E8	C0			
EAD	1F	1F			
EAE	0D	0C			
EAF	AF	EB			

Once you have made these modifications, simply feed in a starting pattern (using the N command), and sit back and watch. The program will continue until a stable pattern is achieved, at which time it will stop. Note, however, that it cannot detect recurring cyclical patterns, so watch out for these. To stop them, you will have to use the reset facility of the 2650.

# Add a low cost printer to your 2650

Here are the details of how to interface the Matsushita model EUY-10E023LE printer and its companion driver board, model EUY-PUD024C to your microprocessor. Details of a suitable power supply are also provided, as well as driver routines to suit the 2650 microprocessor.

**by DAVID EDWARDS**

Obtaining hard copy has always been one of the major bugbears of the home computerist. Secondhand ASCII teleprinters are available, but can cost several times the price of the rest of the system put together. (If you can afford a new teleprinter, you needn't read any further!)

Baudot teleprinters are available at quite reasonable cost, but require a code conversion from ASCII to Baudot, which as well as being messy, tends to raise the overall package price quite markedly. So when Philips (the local

agents for Matsushita printers) supplied us with details of the new printer, we at once decided that this would be a boon for the home hobbyist.

Approximate price of the printer unit and the interface board is \$200.00 plus tax if applicable. Power supply requirements are quite modest, and could possibly be met from existing supplies, or from junk box (or redundant, if you want to be nice!) components.

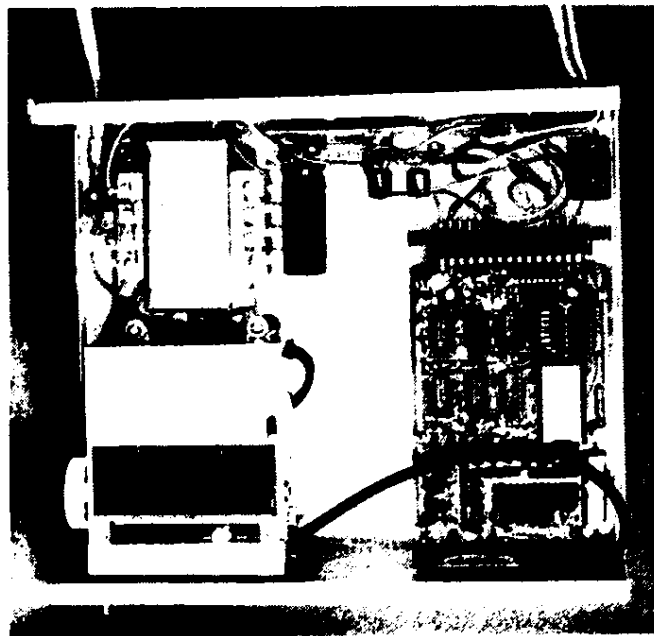
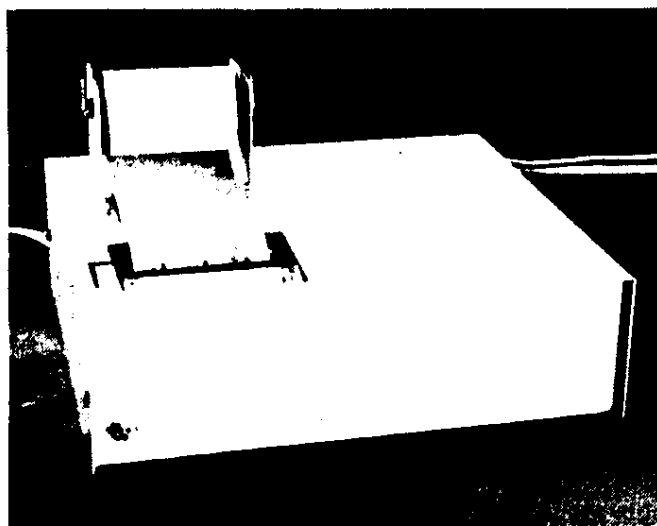
The units are supplied with comprehensive instruction manuals, giving

full details on both mechanical and electrical interfaces, as well as flowcharts and programs suitable for use with the Motorola 6800 "D2" evaluation kit.

Overall size of the printing unit is 110 x 90.5 x 39.5mm. Printing is on electrosensitive paper 60mm wide, utilising a travelling head containing seven electrodes. The head scans from left to right, and can print 32 characters on each line. Approximately two lines can be printed each second.

Characters are formed from a 7 x 5

*These two photographs show the printer module and interface board assembled in a small aluminium case, along with the power supply components. Note the paper roll holder mounted on the lid of the case.*



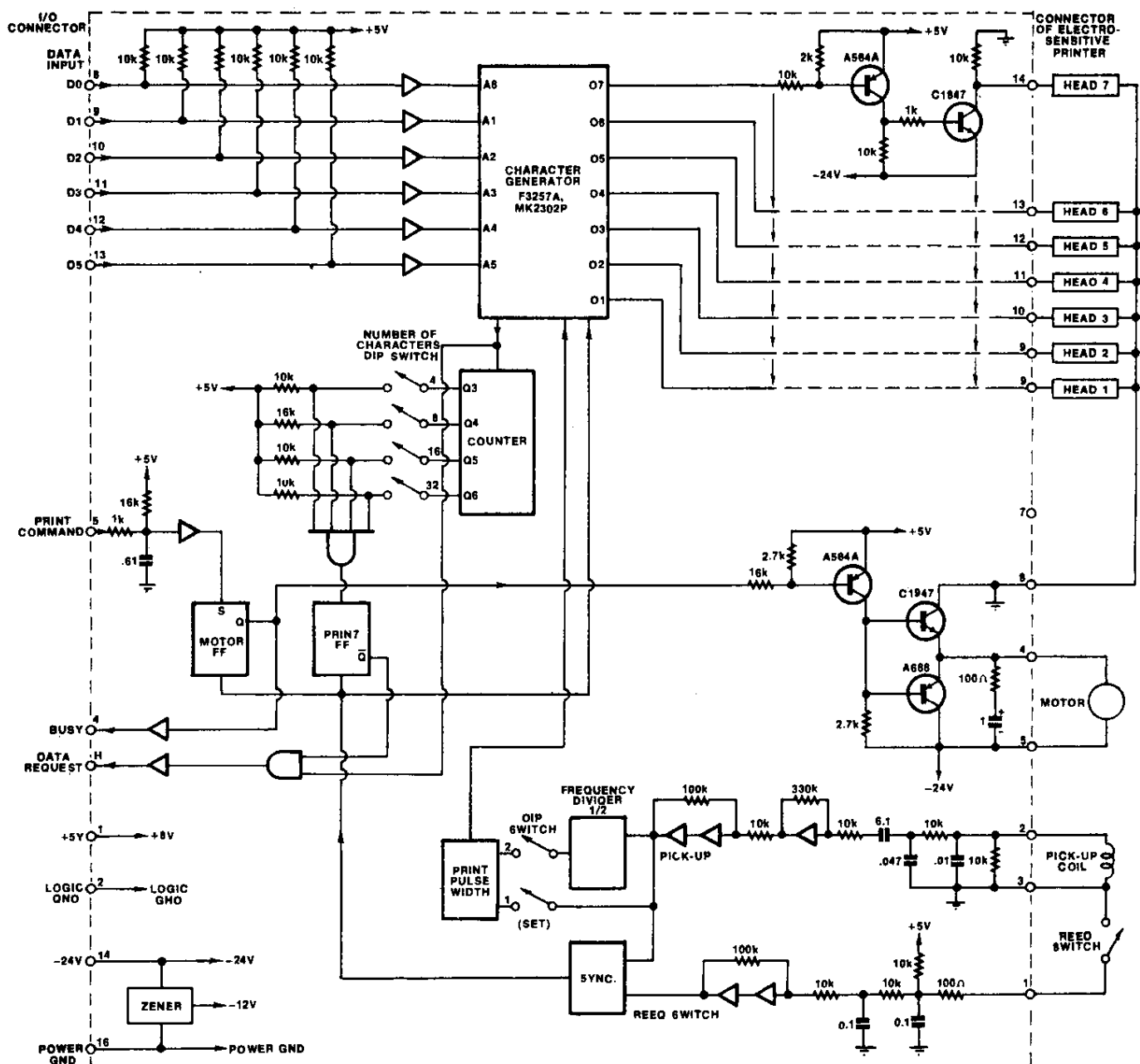


FIG. 1

Here is the schematic diagram of the interface board, which is used to generate the head drive signals from the computer outputs.

dot matrix, and are approximately 2.4mm high. Lines are spaced 2mm apart. The dots are formed by passing current pulses from the appropriate electrodes to the aluminised paper. This burns off the top layer of the paper, leaving a black dot.

Head movement is achieved with the aid of a 24V DC motor. A cam and switch synchronised to the head provide pulses to strobe the characters in time, thus ensuring even character spacing, even if the motor speed varies.

The motor unit does not contain any electronics, and is provided with two cables to connect to the interface board. This board (Fig. 1) contains an I/O circuit, an input data processing circuit, a timing circuit and a character

generator ROM. Data is input in the form of a six bit parallel ASCII code, and converted to the appropriate dot format by the character generator.

A DIP switch is provided to select either 16 or 32 characters per line; normally this would be set at the 32 character position. Three control signal lines are provided, but only two are necessary for a simple interface.

The PRINT signal is used to initiate printing. When this signal is received from the CPU on the interface board, the printing motor is started, and the head starts to scan across the paper. Once the first character has been printed, the DATA REQ signal is pulsed, to signify to the CPU that a second character is required. This sequence is

repeated until the complete line has been printed.

The third control signal is BUSY, which is used to tell the CPU that the printer is occupied in printing a line. Provided the CPU waits for an adequate time after the last character has been sent to the printer, it is not necessary to use this signal.

Two power supplies are required for the interface board: +5V at 50mA and -24V at 200mA. The printer derives its supply voltages from the interface board. Fig. 2 shows a simple supply which is suitable, using two three-terminal regulators and a readily available transformer.

The printer is connected to the CPU via an 8-bit I/O port. For 2650-based

# Adding a printer to your 2650 system

systems, this would normally be the D port. The connections we used, based on the I/O port diagram published in the November 1978 article on expanding the 2650 Minicomputer, are shown in Fig. 3.

Connections to other processor systems will be broadly similar in concept. The BUSY signal, if required, is available at pin 4 of the edge connector.

Needless to say, one needs suitable driver routines in the computer so that it can communicate properly with the printer via the interface. For the benefit of those with 2650 systems I have written some utility routines to do this. One is a basic printer driver subroutine, while the other two are a hex memory dump routine and a message printing routine. Both of these call the basic subroutine for the actual printing.

Fig. 4 shows a flow chart of the main subroutine, PRINT. This treats a portion of memory as a 32 byte buffer, and transfers the ASCII characters stored in it to the printer with the appropriate timing. It will detect a null character in the buffer, and then fill the remainder of the line with spaces. This gives the effect of a carriage return.

To print a number of lines, it is necessary to call the routine the appropriate number of times, changing the buffer contents between calls. To achieve the effect of a blank line, place a null character (X'00) in the first buffer location.

Fig. 5 is a disassembler listing of the PRINT subroutine in 2650 machine code. It occupies locations X'1400 to X'143E inclusive, and requires a 32 byte area of RAM to be set aside as the buffer. At present this occupies

```

14C9 77 02
14CB 3F 00 A4      BSTA,UN
14CE 07 00         LODI,R3
14D0 0C 84 01     LODA,R0
14D3 CF 34 3E     STRA,R3
14D6 18 14       BCIF,EE      14EC
14D8 09 17       LULR,R1      14D1
14DA 0E 04 0F     LODA,R2
14DD 1A 02       BIRH,R2      14E1
14DF 09 00       BIRH,R1      14E1
14E1 3B E9       BSTH,UN      14CC
14E3 E7 20       CUMI,R3
14E5 9B 69       BCPR,EE      14D0
14E7 3F 14 00     BSTA,UN
14EA 1B 62       BCIR,UN      14CE
14EC 3B FA       BSTH,UN      14E8
14EE 17
    
```

FIG. 7

Shown above is a disassembler listing of the message printing routine, while below is a flow chart of the print routine, which is used to control the printer.

```

145F 51
1460 51
1461 51
1462 51
1463 45 0F      ANDI,R1
1465 0D 62 59   LODA,R1
1468 CF 34 3E   STRA,R3
146B 17
146C 77 02
146E 3F 02 DB   BSTA,UN
1471 3F 00 A4   BSTA,UN
1474 3B 19      BSTH,UN      146F
1476 CD 04 0F   STRA,R1
1479 CE 04 10   STRA,R2
147C 07 00      LODI,R3
147E 09 94      LULR,R1      1514
1480 3B 5D      BSTH,UN      145F
1482 09 90      LULR,R1      1514
1484 3B 5D      BSTH,UN      1463
1486 DD 04 DE   LODA,R1
1489 3B 54      BSTH,UN      145F
148B 09 FA      LULR,R1      1487
148D 3B 54      BSTH,UN      1463
148F 04 20      LODI,R0
1491 3B 55      BSTH,UN      1468
1493 0D 84 0D   LODA,R1
1496 3B 47      BSTH,UN      145F
1498 0D 84 0B   LODA,R1
149B 3B 46      BSTH,UN      1463
149D 08 E8      LODR,R0      1487
149F EA D9      COMR,R0      147A
14A1 98 0E      BCPR,EE      14B1
14A3 08 F4      LULR,R0      1499
14A5 E8 D0      COMR,R0      1477
14A7 98 08      BCPR,EE      14B1
14A9 20         LOHZ,R0
14AA 3B 96      BSTH,UN      1542
14AC 3F 14 00   BSTA,UN
14AE 9B 22      BCPR,UN      14D3
14B1 09 E6      LODR,R1      1499
14B3 0A D2      LULR,R2      1487
14B5 DA 02      BIRH,R2      14B9
14B7 D9 00      BIRH,R1      14B9
14B9 3F 00 A4   BSTA,UN
14BC 46 D7      ANDI,R2      148F
14BE 9B 4F      BCPR,EE      148F
14C0 20         LOHZ,R0
14C1 3F 14 68   BSTA,UN
14C4 3B E7      BSTH,UN      14AD
14C6 1F 14 7C   BCIA,UN
    
```

FIG. 8

```

1400 77 10 07 00 F3 04 2A F8
1408 7E 07 40 F3 07 00 06 00
1410 02 98 18 0F 34 3E 18 19
1418 64 40 F0 04 C7 C0 F8 7D
1420 E7 20 10 11 70 F4 00 98
1428 7B 16 65 0B 00 04 20 18
1430 67 06 FF 1B 78 20 05 20
1438 F8 7E F9 7C 75 10 17 31
1440 34 34 30 20 33 34 20 33
1448 34 20 33 33 20 33 33 20
1450 32 30 20 33 33 20 33 33
1458 20 33 33 00 47 4F 4F 51
1460 51 51 51 45 0F 00 62 59
1468 CF 34 3E 17 77 02 3F 02
1470 DB 3F 00 A4 3B F9 C0 04
1478 0F CE 04 10 07 00 09 94
1480 3B 5D 09 90 3B 5D 00 04
1488 0E 3B 54 09 FA 3B 54 04
1490 20 3B 55 00 64 00 3B 47
1498 0D 84 0D 3B 46 00 E8 E8
14A0 D9 98 0E 00 F4 E8 00 98
14A8 00 20 3B 96 3F 14 00 98
14B0 22 09 E6 0A D2 DA 02 D9
14B8 00 3F 00 A4 46 07 98 4F
14C0 20 3F 14 68 3B E7 1F 14
14C8 7C 77 02 3F 00 A4 07 00
14D0 0C 84 0D CF 34 3E 18 14
14D8 09 F7 0E 04 0E DA 02 D9
14E0 00 3B E9 E7 20 98 69 3F
14E8 14 00 1B 62 3B FA 17
    
```

FIG. 8

To the left is a listing of the hex listing routine. This was used to produce the listing above of all the programs presented in this article, on the new printer. It is reproduced actual size.

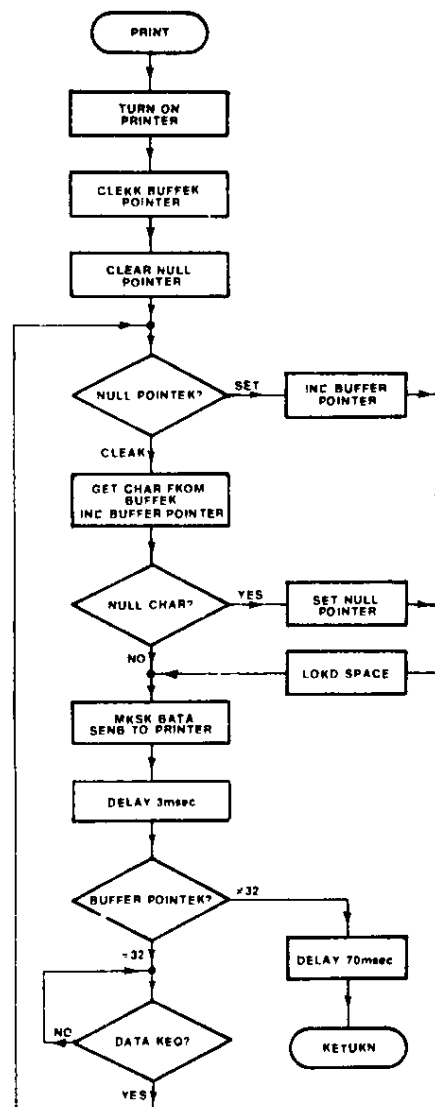


FIG. 4 FLOW CHART FOR PRINT ROUTINE

# Adding a printer to your 2650 system

locations X'143F to X'145E inclusive.

The PRINT subroutine is relocatable. To change the starting address of the buffer, which must be in the same page as the subroutine, put the address of the byte before the desired starting address into locations X'1414 and X'1415. Note that bits 6 and 7 of location X'1414 must remain as zeroes, while bit 5 must remain a 1.

Fig. 6 is a disassembler listing of the listing program, which will print out a listing of a specified area of memory. As the print format is only 32 characters wide, it prints only the line starting address and eight bytes per line. To call the program, type G 146C XXXX YYYY cr, where XXXX and YYYY are the start and end respectively of the required memory block.

The program occupies locations X'145F to X'14C8 inclusive. It uses Pipbug routines GNUM, STRT, ANSI and PIPBUG. It contains absolute addresses, but can be moved fairly easily.

Locations X'1469 and X'146A must point to the byte before the start of the PRINT subroutine's buffer memory.

Bytes X'14AD and X'14AE must point to the starting address of the PRINT subroutine. Bytes X'14C2 and X'14C3 must point to the new location of

1400	77	10	
140B	07	00	LOLT,R3
1404	F3		
1405	04	2A	LOLT,R0
1407	F8	7E	BDRR,R0
1409	07	40	LUDI,R3
140B	F3		
140C	07	00	LUDI,R3
140E	06	00	LUDI,R2
1410	02		LUDZ,R2
1411	98	18	BCFR,EO
1413	0F	34	LODA,R3
1416	18	19	BCFR,EO
1418	64	40	IORI,R0
141A	F0		
141B	04	C7	LOLI,R0
141D	C0		STRZ,R0
141E	F8	7D	BERR,R0
1420	E7	20	COMI,R3
1422	18	11	BCFR,EO
1424	70		
1425	F4	80	
1427	98	7B	BCFR,EO
1429	1B	65	BCFR,UN
142B	DB	00	HRRR,R3
142D	04	20	LUDI,R0
142F	1B	67	BCFR,UN
1431	06	FF	LUDI,R2
1433	1B	78	BCFR,UN
1435	20		EOHZ,R0
1436	05	20	LUDI,R1
1438	F8	7E	BDRR,R0
143A	F9	7C	BLRR,R1
143C	75	10	
143E	17		

FIG. 5

The hex listing reproduced above is for the routine used to control the printer. It is written as a subroutine.

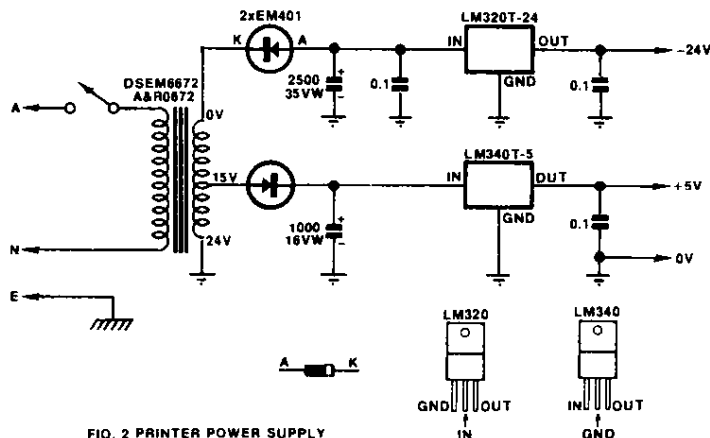


FIG. 2 PRINTER POWER SUPPLY

The connection diagram for the 2650 Mini Computer is shown below. We used the "D" output port.

Shown above is the circuit diagram of the suggested power supply. It uses a readily available transformer.

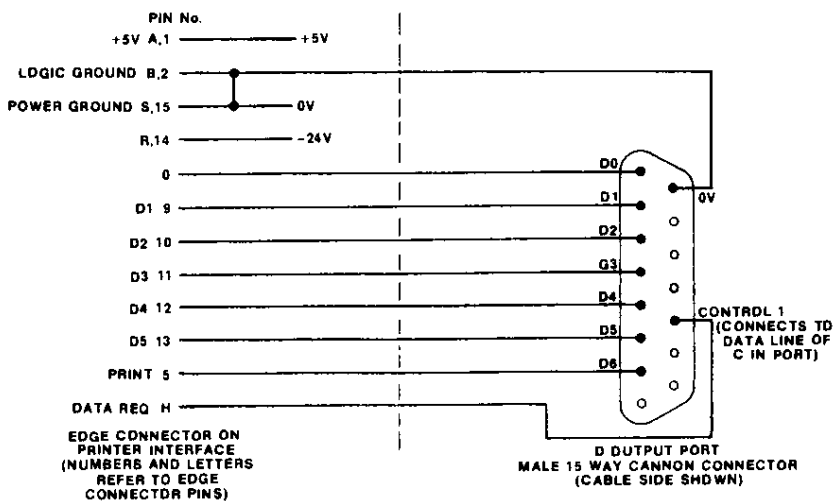


FIG. 3 CONNECTION DIAGRAM

current location X'1468, and bytes X'14C7 and X'14C8 must point to the new location of current location X'147C.

Fig. 7 is a disassembler listing of a message printing subroutine. This expects R1 and R2 to contain the starting address of an ASCII message. This message is printed when the subroutine is called. The end of the message is signified by a null character. Messages can be stored anywhere in available memory, provided they are in the same page as the message program.

The message subroutine is relocatable, and uses the Pipbug routine STRT. Locations X'14D4 and X'14D5 must point to the byte before the first byte of the PRINT buffer, and locations X'14E8 and X'14E9 must point to the PRINT routine itself.

Fig. 8 is a hex listing of all three routines, produced by the hex listing

program on the new printer itself. It is reproduced actual size, so you can see directly the size and quality of the printing.

Further details on the model EUY-10E023LE printer and companion interface board (model EUY-PUD024C) can be obtained from the local agents, ELCOMA, of 67 Mars Road, Lane Cove, NSW 2066.

# Notes & Errata

**NEW GAME PROGRAMS FOR YOUR 2650** (December 1978, File No. 8/M/32): The hexadecimal listings of the programs given in this article did not reproduce with full clarity. To assist readers who found difficulty in feeding the programs into their system, we reproduce a new and (hopefully) clearer set of listings below.

## LIFE: HEX LISTING

```
0C00 76 40 75 FF 04 F9 C8 27 12 1A 7D 3B 25 12 1A 78
0C10 3B 1E 3B 1C 12 9A 04 05 59 1B 07 3B 13 12 1A 04
0C20 05 F9 C9 0B 1B 32 04 0D 3B 0F 04 0A 3B 0B 17 59
0C30 3B 00 0B 7B 14 14 FB 7D 17 77 10 05 08 3B 71 3B
0C40 6F 74 40 3B 6B 50 1A 04 74 40 1B 03 76 40 14 F9
0C50 72 76 40 3B 5B 75 10 17 1B 1C 77 18 05 00 06 08
0C60 12 1A 7D 14 3B 4C 3B 48 12 14 D0 51 FA 78 3B 40
0C70 45 7F 01 75 18 17 05 01 3F 0E B6 04 3A 3F 0C 39
0C80 3F 0C 5A E4 50 1C 0C E8 E4 47 1C 0D 8A E4 4E 98
0C90 6F 3F 0C 39 05 46 20 CD 4E EE 59 7B 05 FF 3F 0C
0CA0 26 06 08 07 00 D3 3F 0C 5A E4 20 13 22 E4 4F 18
0CB0 22 E4 0A 18 10 E4 0D 98 6D 3F 0C 39 85 01 45 FC
0CC0 A5 01 1B 5D D3 FA 7D 03 CD 2E F4 65 03 1B 0F 04
0CD0 2E 1B 02 67 01 3F 0C 39 FA 4B 03 CD 2E F4 F5 03
0CE0 BC 0C A1 E5 3F BC 0C 9E 05 FF 3F 0C 26 06 00 CA
0CF0 15 07 08 0D 2E FA DA 00 9A 02 CA D0 FB 77 F5
0D00 03 98 6E A5 04 06 1C 98 06 85 04 1B 1D FB 09 07
0D10 08 0D 2E F4 77 10 C2 52 77 10 D2 1A 04 0A 20 1B
0D20 02 04 4F 3F 0C 39 FA 65 65 03 E5 3F 9C 0C E0 04
0D30 0D 3B F1 0D 0E EF E5 09 99 04 A5 0A 1B 78 01 24
0D40 30 3F 0C 39 0D 0E EE 06 00 E5 09 99 0E 86 01 E6
0D50 09 99 02 06 00 A5 0A E5 09 19 72 02 24 30 3F 0C
0D60 39 01 24 30 3F 0C 39 1F 0C 7B 3B 0B 07 09 C2 82
0D70 FB 7D C2 3B 02 82 17 3F 0C 5A E4 30 1A 79 E4 39
0D80 19 75 C3 3F 0C 39 03 44 0F 17 3F 0C 39 3B 5B CC
0D90 0E ED 05 0C E4 00 1C 0E B0 05 15 E4 07 1A F8 05
0DA0 1B E4 37 1A F2 05 21 E4 4C 1A EC 05 27 1B E8 05
0DB0 04 06 04 0D 4E F4 CE 4F 34 0E 6F 30 CD 6E F0 5A
0DC0 02 06 04 59 6E 05 FF 07 03 85 03 3B 33 08 2F 50
0DD0 50 C8 2A 3B 14 A5 03 3B 27 08 22 C8 22 3B 0A 0D
0DE0 2E F4 77 10 C1 07 08 1B 3B 0A 13 08 10 CA 0E D0
0DF0 D0 C8 7A 44 03 CF 6E 20 FB 02 07 03 17 81 21 48
0E00 20 C8 7A C8 79 06 03 0D 2E F0 44 55 88 70 C8 6E
0E10 0D 6E F0 50 44 55 88 65 C8 63 85 03 FA 69 A5 0C
0E20 17 01 00 01 FB 0E 75 10 F5 03 18 04 3B 52 1B 04
0E30 08 4D C8 49 75 10 3F 0D E9 77 10 08 64 88 63 88
0E40 62 E4 04 18 10 E4 03 18 05 01 1A 05 1B 07 01 1A
0E50 04 66 01 25 80 D1 5B 4C 75 10 01 44 03 C2 0E 6E
0E60 F0 CD 6E F0 77 10 01 75 10 CE 6E F0 F5 03 9C 0D
0E70 DF E5 3F 9C 0D C9 05 04 0D 4E F0 CD 6F 30 59 78
0E80 77 10 02 1C 0E A8 06 00 75 10 05 01 8D 0E EE E5
0E90 64 98 0A 05 01 8D 0E EF 05 00 CD 0E EE EE
0EA0 0C 0E ED A4 01 CC 0E ED E4 00 1C 0C E8 1F 0D AF
0EB0 3F 0E B6 1F 0D AF 4D 2E BF E4 00 14 3F 0C 39 1B
0EC0 75 0D 0A 22 4C 49 46 45 22 0D 0A 00 20 3D 20 32
0ED0 35 36 47 2E 00 20 3C 30 35 53 00 20 3C 31 35 53
0EE0 00 20 3C 32 30 53 00 20 3C 32 35 53 00
```

\*

## ROTATE: HEX LISTING

```
0440 06 FF 3F 05 64 05 08 07 F5 1B 00 12 9A 06 D9 79
0450 DB 79 1B 71 06 31 66 80 F6 03 18 0A 26 FF F6 03
0460 18 02 66 80 26 FF 52 CA 6C 46 0F 3F 04 C4 18 64
0470 D9 62 DB 60 07 0F 1F 04 B5 3F 05 44 06 0F 3F 05
0480 64 3F 02 86 E4 58 1C 05 03 E4 0D 18 0D 3F 04 E7
0490 58 6F 3B 30 98 1A 06 19 1B 64 03 C2 3B 26 18 76
04A0 3B 22 3B 20 07 0F 06 21 3F 05 64 A5 01 95 1B 49
04B0 02 C3 85 67 95 06 40 0E 25 76 E2 18 7A 60 1C 05
04C0 56 1F 04 79 F6 03 14 F6 0C 14 0E 65 BB C8 13 0E
04D0 65 BC CE 65 BB 0E 65 B8 CE 65 BC 0E 65 B7 CE 65
04E0 B8 04 4E CE 65 B7 17 E4 5A 1C 05 58 E4 41 1A 12
04F0 E4 50 19 0E 06 10 EE 45 B7 18 04 5A 79 9B 22 3F
0500 02 B4 17 06 28 3F 05 64 3F 02 86 3B 5A 58 79 02
0510 C3 04 2C 3F 02 B4 3F 02 86 3B 4C 58 79 02 A3 9A
0520 02 03 A2 E4 01 18 08 E4 04 18 04 06 19 1B 56 0E
0530 65 B7 C8 07 0F 65 B7 CE 65 B7 04 44 CF 65 B7 07
0540 0F 1F 04 B2 06 01 0E 25 B3 14 3F 02 B4 F6 03 98
0550 75 3F 00 8A 1B 70 3B 6C 06 3A 3B 08 3F 02 69 3B
0560 03 1F 04 40 0E 25 6D 14 3F 02 B4 1B 77 0A 50 52
0570 45 53 53 20 41 4E 59 20 4B 45 59 00 00 0A 52 4F
0580 54 41 54 45 3A 20 00 0A 57 48 41 54 3F 20 00 43
0590 41 4E 43 45 4C 00 0A 45 58 43 48 41 4E 47 45 3A
05A0 20 00 0A 59 4F 55 20 54 4F 4B 20 00 20 4D 4F
05B0 56 45 53 0A 00 0D 0A 4D 4E 48 41 4B 50 49 44 43
05C0 45 4A 4F 46 4C 42 47 00
```

\*

## MUSIC: HEX LISTING

```
04A0 04 00 0C A5 56 1E 05 58 C2 44 70 24 70 C1 51 51
04B0 51 81 51 77 10 83 C3 86 00 75 11 46 0F D2 0E 65
04C0 BC C3 0E 25 BC 06 FF E5 00 18 05 D0 D3 D2 F9 7B
04D0 75 01 84 80 87 00 86 00 CB 22 CA 1F 77 10 87 44
04E0 86 01 9A 06 87 0E 86 00 1A 7A 0D 85 56 77 01 AB
04F0 09 AA 06 76 40 04 08 1B 31 28 00 F9 5A AB 7B AA
0500 78 93 07 8C 8B 76 8A 73 9A 06 87 0E 86 00 1A 7A
0510 53 13 53 B5 01 98 03 1B 01 C0 47 03 9F 05 1F C0
0520 C0 C0 93 12 24 40 92 13 77 11 AB 50 AA 4D 9A 05
0530 93 07 80 1B 4F F9 46 44 01 4C 04 FC 64 18 93 8F
0540 04 FC 8E 04 FB 75 11 0B 0E 0A 0B 87 02 86 00 CB
0550 06 CA 03 1F 04 A0 06 B6 77 10 E4 FF 1C 00 22 09
0560 F5 87 8A 86 00 9A 06 87 0E 86 00 1A 7A 74 40 75
0570 01 86 02 77 01 AF 04 FA AE 04 F9 87 1E 86 00 87
0580 0E 86 00 1A 7A F9 6E A6 02 1F 05 45 75 FF 3F 02
0590 DB 0C 04 2A 98 06 0A 23 09 20 1B 04 CA 1D C9 1A
05A0 CE 05 57 CD 05 56 3F 02 DB 77 08 0C 04 2A 1C 04
05B0 A0 CE 04 FA CD 04 F9 1F 04 A0 05 D4 02 FC 11 2F
05C0 1E 97 2B 3E 37 2F 42 74 4D 17 57 22 60 9C 69 8E
05D0 72 00 79 F8
```

\*

## YANKEE DOODLE: HEX LISTING

```
05D4 80 80 08 43 08 80 08 43 08 80 08 45
05E0 08 80 08 47 08 80 08 43 08 80 08 47 08 80 08 45
05F0 08 80 08 3A 08 80 08 43 08 80 08 43 08 80 08 45
0600 08 80 08 47 08 80 14 43 08 80 08 42 08 80 08 3A
0610 08 80 08 43 08 80 08 43 08 80 08 45 08 80 08 47
0620 08 80 08 48 08 80 08 47 08 80 08 45 08 80 08 43
0630 08 80 08 42 08 80 08 3A 08 80 08 40 08 80 08 42
0640 08 80 10 43 0C 80 10 43 10 80 08 40 10 80 08 42
0650 02 80 08 40 08 80 08 3A 08 80 08 40 08 80 08 42
0660 08 80 10 43 10 80 08 3A 10 80 08 40 02 80 08 3A
0670 08 80 08 38 08 80 10 37 10 80 10 3A 10 80 08 40
0680 10 80 08 42 02 80 08 40 08 80 08 3A 08 80 08 40
0690 08 80 08 42 08 80 08 43 08 80 08 40 08 80 08 3A
06A0 08 80 08 43 08 80 08 42 08 80 08 45 08 80 10 43
06B0 0C 80 10 43 80 80 02 FF
```

\*

## BACH: HEX LISTING

```
06B8 80 80 06 43 06 45 04 47
06C0 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53 04 52
06D0 04 53 04 4A 04 47 04 43 04 45 04 47 04 40 04 4A
06E0 04 08 04 47 04 45 04 43 04 3A 04 43 04 42 13 43
06F0 0C 80 13 47 0C 43 18 4A 0C 4A 18 48 0C 47 13 45
0700 06 80 06 80 13 47 0C 48 13 4A 0C 47 06 45 03 47
0710 03 48 0C 47 0C 45 18 43 0C 8F 02 80 06 43 06 45
0720 04 47 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53
0730 04 52 04 53 04 4A 04 47 04 43 04 45 04 47 04 40
0740 04 4A 04 48 04 47 04 45 04 43 04 3A 04 43 04 42
0750 04 43 04 47 04 4A 04 53 04 4A 04 47 04 43 04 47
0760 04 49 18 4A 06 80 02 90 03 80 06 43 06 45 04 47
0770 04 4A 04 48 04 48 04 50 04 4A 04 4A 04 53 04 52
0780 04 53 04 4A 04 47 04 43 04 45 04 47 04 40 04 4A
0790 04 48 04 47 04 45 04 43 04 3A 04 43 04 42 18 43
07A0 80 80 02 FF
```

\*

**Program a 2708 in under five minutes!**

# Simple EPROM burner suits SC/MP and 2650

Using only four low cost ICs, this single board design will allow any static microprocessor, such as the SC/MP or 2650, to program 2704 and 2708 type EPROMs. Programming time for 1K bytes is just under five minutes, and no special interface circuits are required.

**by DAVID EDWARDS**

The basic circuit configuration and idea used in this project came originally from one of our readers, Mr M.J. Ogden of Hope Valley, South Australia. As soon as we saw it, we decided that it was too good an idea to publish purely in basic circuit form. So with Mr Ogden's approval we have expanded the original concept into a full construction project.

Like all good ideas, Mr Ogden's is delightfully simple. PROM programming interfaces normally have to provide address and data latches, as this

information must be held static during the relatively long periods taken to program PROM locations. But some microprocessors, like the 2650 and SC/MP, are static devices, and are capable of being forced into a "hold" or "wait" state without loss of data. This is to allow them to be used with slow memory or peripheral devices. Why not take advantage of this facility, and use the processor itself as the address and data latches for the PROM programmer?

With static processors like the 2650

and the SC/MP the idea turns out to be very easy and straightforward. All that is necessary is to arrange simple logic so that to program each PROM location the processor is made to begin a normal instruction cycle storing the required data to the appropriate address, then "held" or frozen with the data and address information present on the bus lines until the programming circuitry has done its job.

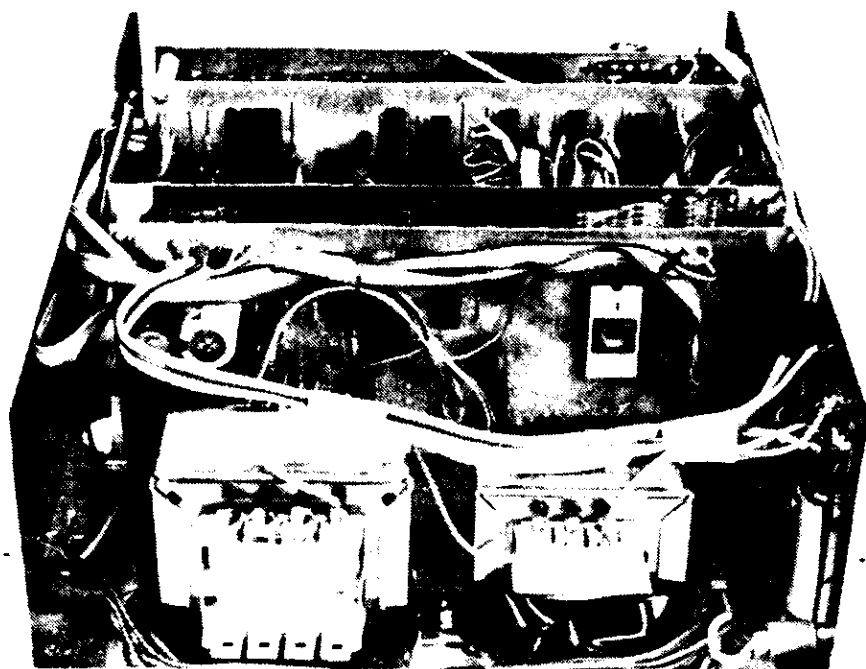
Before discussing the operation of the circuit in detail, an explanation of the operation and programming requirements of an EPROM is in order.

The popular 2708 EPROM uses floating-gate avalanche mode MOS transistors as the storage cells. Stored charges on the floating gates are used to control the conduction of the MOS transistors, to determine whether they effectively store a "1" or a "0".

The floating gate's charge is produced by inducing a non-damaging avalanche breakdown in the drain-channel junction of the cell. High energy electrons from the avalanche breakdown are then injected into the floating gate, charging it negatively. Since the floating gate is surrounded by an extremely effective insulator, this charge will remain practically indefinitely, and hence the stored pattern will also remain.

To erase a programmed EPROM, the chip is irradiated with ultra-violet light. The resulting photons impart enough energy to the trapped electrons to allow them to escape from the floating

*In this photograph, you can see how the new board fits into the 2650 Minicomputer case.*





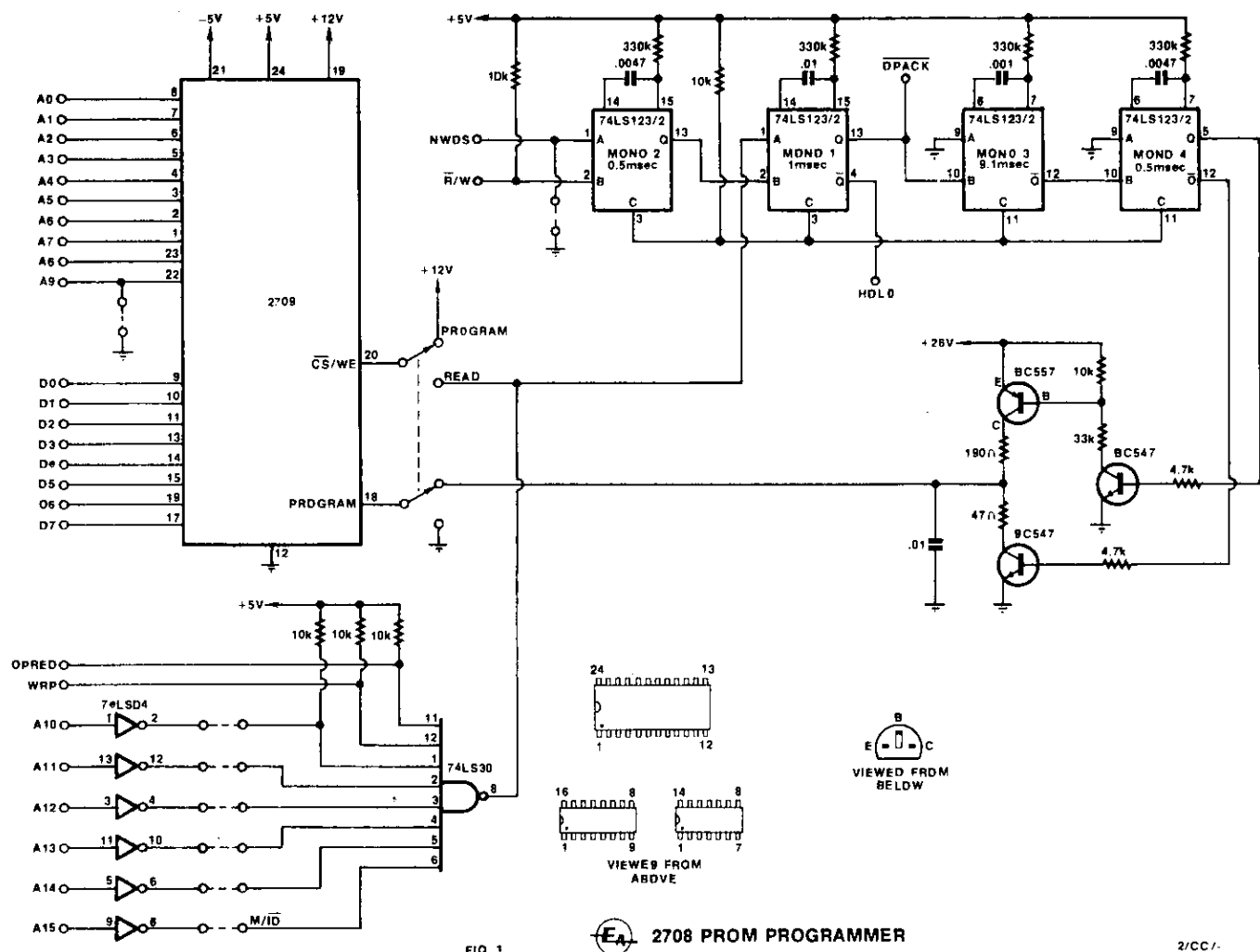


FIG. 1

## 2708 PROM PROGRAMMER

2/CC1

gate, leaving it uncharged.

An erased EPROM has all memory cells effectively containing 1's, so programming consists of inducing avalanche mode breakdowns in the appropriate cells to produce the required zeros. In principle one programming pulse is required for each memory location. The appropriate address and data information must be applied to the address and data pins of the EPROM.

In practice, due to power dissipation limits, it is necessary to apply relatively short programming pulses, and to cycle repetitively through all memory locations until a sufficient number of programming pulses have been applied to each location.

Turning now to Fig. 1, we can see how the basic idea can be used to implement a simple EPROM programmer. The microcomputer's address and data lines are connected directly to the EPROM. In the read mode, a chip select signal is derived from the high order address lines by a decoder implemented with a hex inverter and an eight input NAND gate.

This decoder allows the EPROM to be patched into any available area of memory. To use the top 1K section of memory, it is not necessary to use the

inverter; the address lines can simply be connected directly to the NAND gate.

In the program mode, the chip select input is connected to the +12V line. The output of the address decoder is now used to enable a monostable (mono 1) with a period of 1ms. This monostable is triggered from the output of a second monostable (mono 2), which itself is triggered from the write select signal.

Thus the first monostable is only triggered when a write instruction occurs to a valid EPROM address. The output of this monostable is used to drive the hold line of the processor, halting the write operation in midstream, and leaving the appropriate address and data information on the processor buslines. Fig. 4 shows the timing relationships schematically.

At the same time, a third and delaying monostable (mono 3) is triggered, with a pulse width of 0.1ms. The trailing edge of this pulse is used to trigger a fourth monostable (mono 4) which has a pulse width of 0.5ms. The outputs of this monostable are used to drive the programming pulse generation circuitry.

The programming pulse has an

Above is the complete circuit diagram of the programmer, while below is a diagram illustrating the timing relationships between the monostable multivibrators.

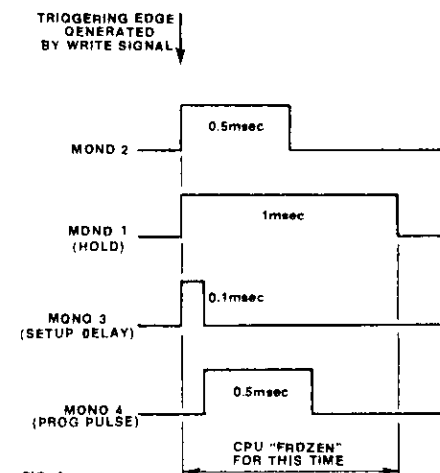


FIG. 4

amplitude of 26V, and lasts for 0.5ms. Approximately 0.4ms after the end of the programming pulse, the output from the first monostable returns to the quiescent state, and the hold is removed from the processor.

Thus to program the EPROM, all that

# 2708 EPROM programmer

```

0440 08 0F C8 13 03 0C C8 10 08 0B C8 0D 08 08 C8 0A
0450 17 3C 00 3D BC 7C 00 3D BC 7D BC 08 7C 09 7B D9
0460 02 D8 00 C8 74 C9 73 08 6E 09 6D D9 02 D8 00 C8
0470 66 C9 65 E8 5E 16 E9 5C 17 3F 00 8A C9 83 CA 8A
0480 0C 84 25 14 3F 02 B4 09 F8 0E 04 26 DA 6E D9 6C
0490 1B 6A 76 40 77 02 75 18 3F 02 DB CD 04 51 CE 04
04A0 52 3B F6 DA 02 D9 00 CD 04 53 CE 04 54 3B EA CD
04B0 04 55 CE 04 56 07 00 05 05 06 2A 3B 9E 3F 02 86
04C0 05 05 06 74 3B 95 FF 05 1B 05 05 06 50 3B 8C 06
04D0 3A 3F 04 8C 39 E8 05 05 06 5F 3F 04 79 3F 04 40
04E0 07 06 3F 00 8A 20 C8 32 0C 84 59 EC 84 57 18 19
04F0 09 F7 3F 02 69 0D 04 5A 3B F9 04 20 3F 02 B4 FB
0500 04 07 06 39 DE 04 80 C8 11 3F 04 5B 1A 5A 08 0A
0510 1A 06 05 05 06 70 3B C3 9B 22 80 3B C1 0C 84 57
0520 CC 84 59 3B E5 1A 76 1F 04 C6 53 57 49 54 43 48
0530 20 54 4F 20 50 52 4F 47 52 41 4D 0D 0A 54 48 45
0540 4E 20 50 52 45 53 53 20 41 4E 59 20 4B 45 59 00
0550 53 57 49 54 43 48 20 54 4F 20 52 45 41 44 00 45
0560 52 52 4F 52 20 4C 4F 43 41 54 49 4F 4E 53 3A 00
0570 4E 49 4C 00 50 52 4F 47 52 41 4D 4D 49 4E 47 0D
0580 0A 00

```

FIG. 5

This hex listing is a 2650 version of the program required to control the EPROM programming operation.

Shown below is the power supply circuit. The components marked with an asterisk (\*) are already present in the 2650 Minicomputer.

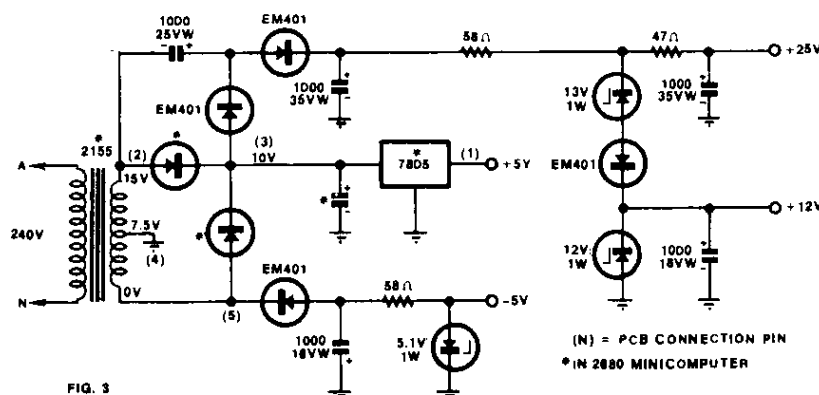


FIG. 3

is required is to switch to the program mode, and then to program the processor to write the appropriate data to each location in turn, repeating this writing sequence until the required number of programming pulses have been applied to each location.

All of the timing requirements are provided by the four monostables, all that the program has to do is provide a repeated "block move" function. A block diagram of a simplified routine to do this is shown in Fig. 2.

Power supply requirements for the 2708 are quite complicated. -5V, +5V and +12V supplies are required for normal operation, while +26V is required during programming. Fig. 3 shows how these voltages can be derived from a standard transformer, using zener diode regulators. The +5V supply can be obtained from the existing circuitry.

We have designed a suitable printed circuit board for mounting the EPROM, address decoder, monostables and power supply components. It is coded 79upl, and measures 218 x 81mm.

Provision has been made so that this board can be used with any suitable processor. Positive and negative going hold signals are available, and the write monostable can be triggered from either positive or negative going signals. Any starting address for the EPROM can be decoded, up to H'7C00.

Construction of the board should be well within the capabilities of most enthusiasts. We recommend that a good quality socket, preferably a zero-insertion force type, be used for the 2708. The remaining ICs can be soldered directly in place.

The programming switch can be mounted directly on the board, using tinned copper wire, or it can be ex-

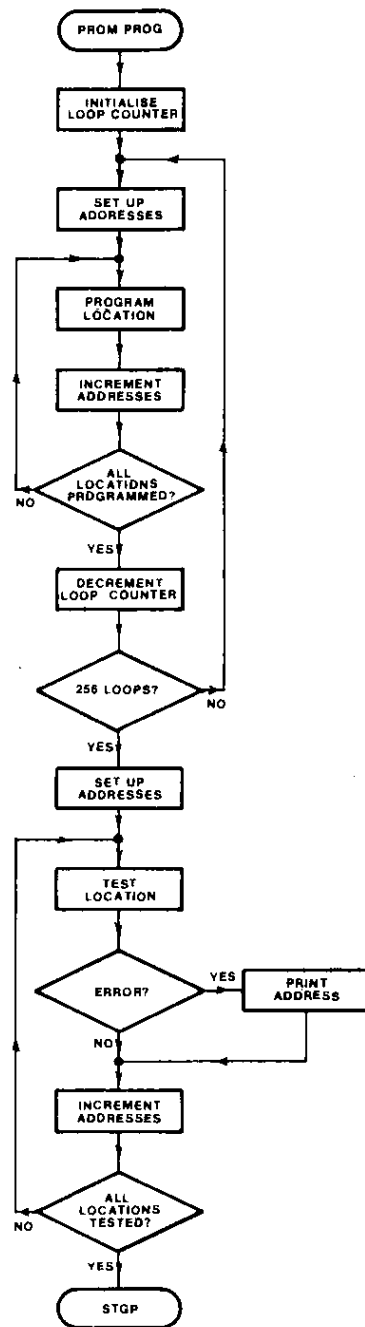


FIG. 2

This is the flowchart for the controlling program. Use it to write the routines required by your processor.

tended with a cable if desired. It should be placed in a position where accidental operation is unlikely, to ensure that no accidental programming occurs.

As the board has to be wired directly to the address and data busses, we recommend that it be mounted in the main computer case. Make sure, however, that access can be gained relatively easily to the EPROM socket and the read/program switch.

## 2708 EPROM programmer

In order to illustrate the use of the Prom Programmer board, we have used it with the 2650 Mini Computer. In this case, the M/IO-bar signal is connected to the address decoder instead of AD15. The OPREQ and WRP signals are also connected to the decoder, using the spare inputs to the NAND gate.

The A input of monostable 2 is grounded, and the R-bar/W signal applied to the B input. The Q output of monostable 1 is used to drive the OPACK-bar line. We elected to make the EPROM occupy locations from H'3C00 to H'3F00, the last 1k of page 1. To do this, it is necessary to apply AD10, AD11, AD12, AD13, and AD14-bar to the NAND gate.

We chose this area so that it would be relatively easy to provide a small amount of RAM in the same page. This is required because of the limitations of the 2650 absolute addressed memory reference instructions. We used the spare RAM sockets on the main CPU board, wired up as the first 4K of page 0.

The 2650 program we developed to control the Prom Programmer is given as a hex listing in Fig. 5. It occupies locations H'0440 to H'0581 inclusive, and is not easily relocatable. To call the program, which uses Pipbug routines CRLF, COUT, GNUM, CHIN and BOUT, type G492 XXXX YYYY ZZZZ cr, where XXXX and YYYY are four digit hex numbers representing the start and end addresses of the area of RAM to be copied into the EPROM, and ZZZZ is a similar number representing the address of the first EPROM location.

The program will respond with the message "SWITCH TO PROGRAM THEN PRESS ANY KEY". The read/program switch (it should normally be in the read mode) is now moved to the program position, and any keyboard key of the terminal pressed.

The program will then respond with the message "PROGRAMMING", and then appear to do nothing while it actually programs the EPROM. A 1K "burn" will take nearly five minutes, shorter burns proportionately less.

When the programming is complete, the program will print out "SWITCH TO READ THEN PRESS ANY KEY". When this command has been carried out, the program begins to verify that the data has been stored correctly in the EPROM. First it responds with "ERROR LOCATIONS:" and then gives a list of any locations not correctly programmed. If there are no errors, the message "NIL" will be displayed.

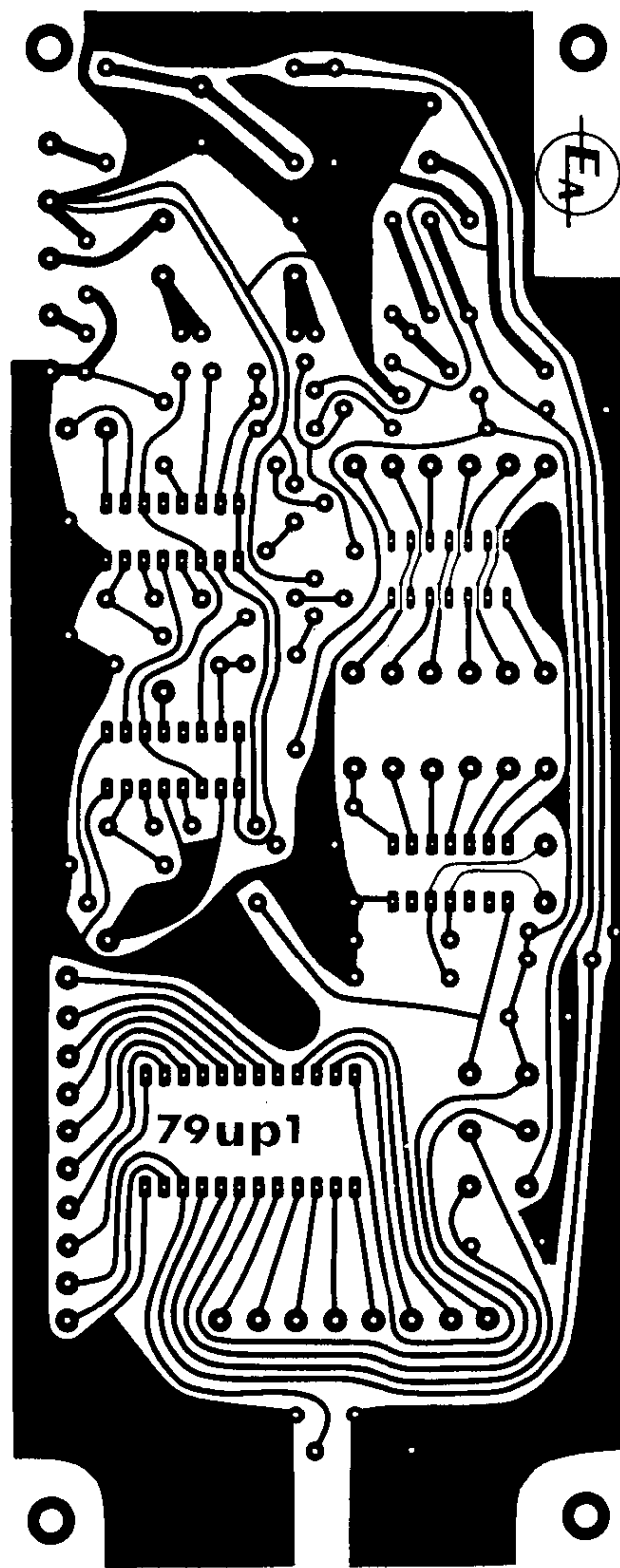
If you obtain a small number of errors, this indicates that there was insufficient programming at these locations. To remedy this, simply repeat the programming process for all of the block you are attempting to program.

To use the programmer board with a SC/MP system, connect the address and data lines to the appropriate points on the board. Insert the two 10k pullup resistors on the OPREQ and WRP lines, but leave these lines unconnected. The Q-bar output of mono 1 is used to drive the SC/MP HOLD line, while the SC/MP's NWDS signal is used to drive the A input of mono 2. The B input of this monostable is pulled permanently high by the 10k pullup resistor provided.

That is all the hardware alterations required, apart from providing the appropriate power supplies. Of course, you will need to write an appropriate controlling program, to output the required data to the EPROM. Use the flow chart provided as a guide.

Do not attempt to program a block of memory smaller than about 256 bytes, as otherwise the power dissipation limits of the 2708 may be exceeded. If you have to program a small block, reduce the number of program loops (specified in location H'04B6), by a proportionate amount, and then program the PROM repeatedly until a correct burn-in is achieved.

Once an EPROM has been programmed, it is recommended that the transparent quartz window above

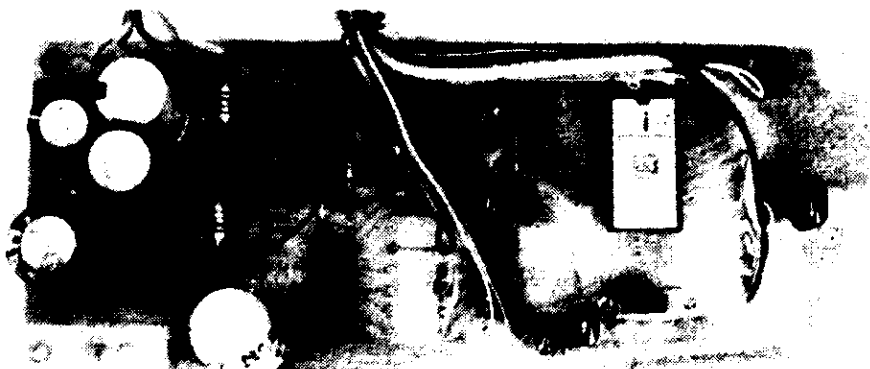
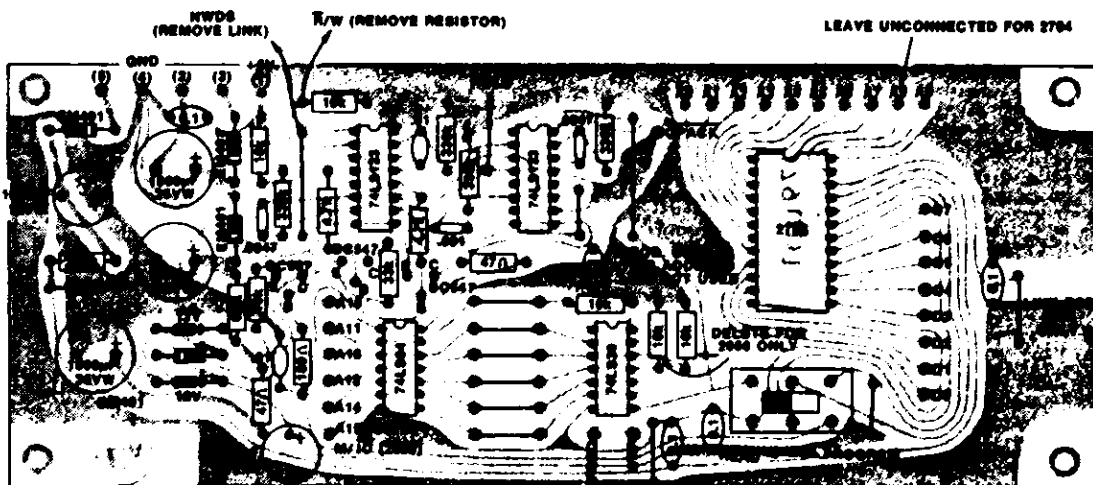


*You can either copy this actual sized reproduction of the PCB artwork, or trace it directly. Alternatively, commercial boards should be available in due course.*

the chip be covered with an opaque layer, to prevent possible leakage currents generated by ambient light from causing malfunctions.

If you wish to erase a programmed 2708 EPROM, you will need a source of ultra-violet light with a wavelength of 2537 Angstroms. A suitable source is the TUV 15W lamp (cat. no. 57415P/40), available from Philips. It fits in a standard 20W fluorescent holder, and should be ordered from an elec-

The photograph below is of the completed prototype. Note in particular the way in which the address wiring has been completed for addresses 3C00-3FFF.



With the window of the 2708 about 25mm from the tube, an exposure time of approximately 30 minutes will be required. This will erase all sections of the device. Note that lamps of this type must be used with caution, as eyes and skin may be affected by long exposures.

In the next issue, we plan to publish details of suitable 2650 utility routines for programming into a 2708. These will include a hex input routine, a hex listing routine, a block move routine, a search routine and a tape verify routine, and possibly other useful sub-routines. ●

Can be stored in a 2708 EPROM:

# 2650 utility programs

Here are five utility programs for the 2650 microcomputer, suitable for loading into a 2708 EPROM using the programmer recently described. The routines allow you to perform hex listings, enter programs rapidly in hex from the keyboard, search memory blocks for an instruction, move program or data blocks in memory, and verify program tapes. A number of useful subroutines are also available for use by other programs.

by DAVID EDWARDS

The routines presented in this article are modifications of those originally presented on the Philips/Electronics Australia software record, described in the April 1978 issue. The original routines are quite useful, but have one disadvantage: they have to be loaded into memory every time that the computer is switched on.

By having them stored permanently in a ROM, however, you can avoid this trouble, and make them available for use at a moment's notice. So after completing the EPROM Programmer, my thoughts immediately turned towards these routines, and whether they could be stored in a ROM.

My first idea was to have the EPROM occupy the uppermost memory locations, i.e. from X'7C00 to X'7FFF, so as to leave all of the space below this for memory expansion. However, when I examined the programs in greater detail, I realised that it was necessary to have a small amount of RAM available in the same page as the EPROM, because of the limitations in 2650 memory reference instructions.

The additional hardware required to shift 1k of the existing RAM up into page 3 proved to be too complicated, so I compromised, and decided to put the EPROM at locations X'3C00 to X'3FFF inclusive — i.e., at the top of

page 1. The modifications to achieve RAM in this page then became quite simple, involving only one extra gate.

My system at the moment has page 0 completely filled, with the 1K PIPBUG ROM at the bottom of the page, and 7K of RAM filling up the remainder. This RAM is mounted on the prototype 8K RAM board (see December 1978), with pairs of 2114s occupying all locations except those corresponding to the addresses occupied by PIPBUG.

Note that this involves a rearrangement of the high-order address decoding. The 74LS138 decoder on the expansion board is used as the page decoder, and controls the 74LS138s on both the RAM board and the CPU board. The 74LS138 on the RAM board becomes the page 0 decoder, while that on the CPU board becomes the page 1 decoder. Refer to Fig. 1 for a diagram of the wiring.

The chip select signal for PIPBUG is now obtained from the 74LS138 on the RAM board, while the four "spare" RAM pairs on the CPU board are controlled by the 74LS138 on that board. Strictly speaking, only three of these pairs should be used, to avoid overloading the address bus, but in practice we have found that all four pairs can be used without problems.

It is now necessary to disable the main data buffers whenever either PIPBUG or the four RAM pairs are selected. This is the function of the additional gate, the 74LS30 shown in Fig. 1. This is an eight input gate, and is used to replace the inverter provided on the expansion board. It can be mounted on a small piece of Veroboard.

These modifications allow a maximum of 13K of memory to be used, including 11K of RAM. PIPBUG occupies locations X'0000 to X'03FF, RAM from X'0400 to X'2FFF, and the EPROM from

When in ROM, the programs must reside at location X'3C00 to X'3DBD.

```
0600 CD 0F FA CE 0F FB 17 76 40 77 02 75 18 3F 02 DB
0610 3B 6E 3B FA 3B 0F CD 0F FC CE 0F FD 3B F0 CD 0F
0620 FE CE 0F FF 17 DA 02 D9 00 17 0D 0F FA 0E 0F FB
0630 3B 73 3B 4C ED 0F FC 16 EE 0F FD 17 3F 00 8A 0D
0640 0F FA 3F 02 69 0D 0F FB 3B F9 04 20 3F 02 B4 17
0650 3F 3C 07 3B 67 0D 0F FA 3B E9 3B 6E 3B 4C 9E 00
0660 22 0C 0F FB 44 0F 98 6D 1B 69 3B E5 3B CF 0C 8F
0670 FA EC 0F FE 98 0D 07 01 0F EF FA EC 0F FF 98 03
0680 3F 3C 3C 3F 3C 2A 9A D7 1B 64 3B C5 3B F3 3F 02
0690 86 C3 3F 02 B4 E7 07 18 C6 E7 20 18 0B 0C 0F FF
06A0 CC 0F FE CF 0F FF 1B 66 0C 0F FE 3F 02 46 D3 D3
06B0 D3 D3 CF 0F FE 0C 0F FF 3B F2 6F 0F FE CF 8F FA
06C0 3B C2 0C 0F FB 44 0F 18 43 1B 43 3F 00 8A 3F 3C
06D0 00 0C 0F FA 14 3F 02 B4 3F 3C 2A 1B 71 76 40 3F
06E0 02 86 E4 3A 98 79 20 C8 97 3F 02 24 CD 0F FA 3B
06F0 F9 CD 0F FB 3B F4 59 0E 05 3D 06 31 3B 95 9B 22
0700 04 2C 04 28 04 29 C9 FA 3B E0 08 F4 18 09 05 3D
0710 06 34 3F 3C CB 9B 22 C3 CB EA 3B CE 0B E6 EB E2
0720 18 08 01 EF EF FA 98 66 DB 6E 08 D4 98 60 1F 3C
0730 DF 4F 4B 00 46 41 55 4C 54 59 00 3F 3C 07 ED 0F
0740 FA 19 34 EE 0F FB 1D 3D 84 0C 8F FA CC 8F FE 3F
0750 3C 2A 9E 00 22 3B 07 3F 3C 25 3B 09 1B 6B 0D 0F
0760 FE 0E 0F FF 17 CD 0F FE CE 0F FF 17 0D 0F FC 0E
0770 0F FD 3B 07 CD 0F FC CE 0F FD 17 FA 00 E6 FF 98
0780 02 F9 00 17 3B 66 77 09 3B 54 AE 0F FB AD 0F FA
0790 75 01 8E 0F FD 8D 0F FC 3B 4B 75 08 0C 8F FC CC
07A0 8F FE 3F 3D 5E 3B 54 3F 3D 65 3B 40 0C 8F FC CC
07B0 8F FE ED 0F FA 19 6B EE 0F FB 19 66 9B 22
```

FIG. 2

X'3C00 to X'3FFF. This should allow quite large programs to be run.

The uppermost RAM locations can be reserved for scratchpad use by programs in the ROM. Only six locations are required by the programs presented in this article, so this leaves nearly 11K of RAM available for your programs.

Now that the hardware has been sorted out, we can discuss the programs themselves. These use PIPBUG routines GNUM, CRLF, BOUT, COUT, LKUP, CHIN and BIN, as well as RAM locations CNT, BCC and MCNT.

The first program provided is titled HEX LIST. This produces a hexadecimal listing of any desired memory block, with each line consisting of an address followed by 16 data bytes. To call this routine, type G3C50 AAAA BBBB cr, where A is the start address of the memory area to be dumped, and B is the end address. The listing will include the specified start and end addresses.

If you wish to have fewer data bytes per line, change the contents of location X'3C65 to the appropriate hexadecimal number before you burn the EPROM.

The second routine is called SEARCH. It will list all locations within a given memory block that match a given test pair of data bytes. The matching addresses are printed out in a single column. To call this program, type G3C6A AAAA BBBB XYYY cr, where A and B are the start and end addresses of the range to be searched, and XYYY is the test pattern.

The addresses printed out are those of the first byte of the matched pairs. The search is inclusive, and includes the start and end addresses.

The third program is called HEXIN, and will enable data or programs to be entered into RAM much faster than using the PIPBUG "A" routine. To call the program, type G3C8A AAAA cr, where A is the address of the first RAM location at which bytes are to be entered.

The program will respond by printing out the start address on a new line, and then wait for you to enter hexadecimal characters. Bytes are separated by spaces, and only the last two characters entered before a space are accepted by the program. This means that if you make a mistake, you can simply type in the correct characters before typing the space.

After 16 bytes have been accepted, the program will give a CRLF, and then print the current address at the start of the new line. In this way, if you are careful, you will produce a hex listing as you input the bytes. To terminate the entry mode, type a control-G "BELL" after the space entering the last byte.

The fourth program is titled VERIFY, and is used to check that a PIPBUG absolute object format dump tape is correct and contains no errors, before the master in RAM is destroyed. To use the program, simply type G3CDD cr, and

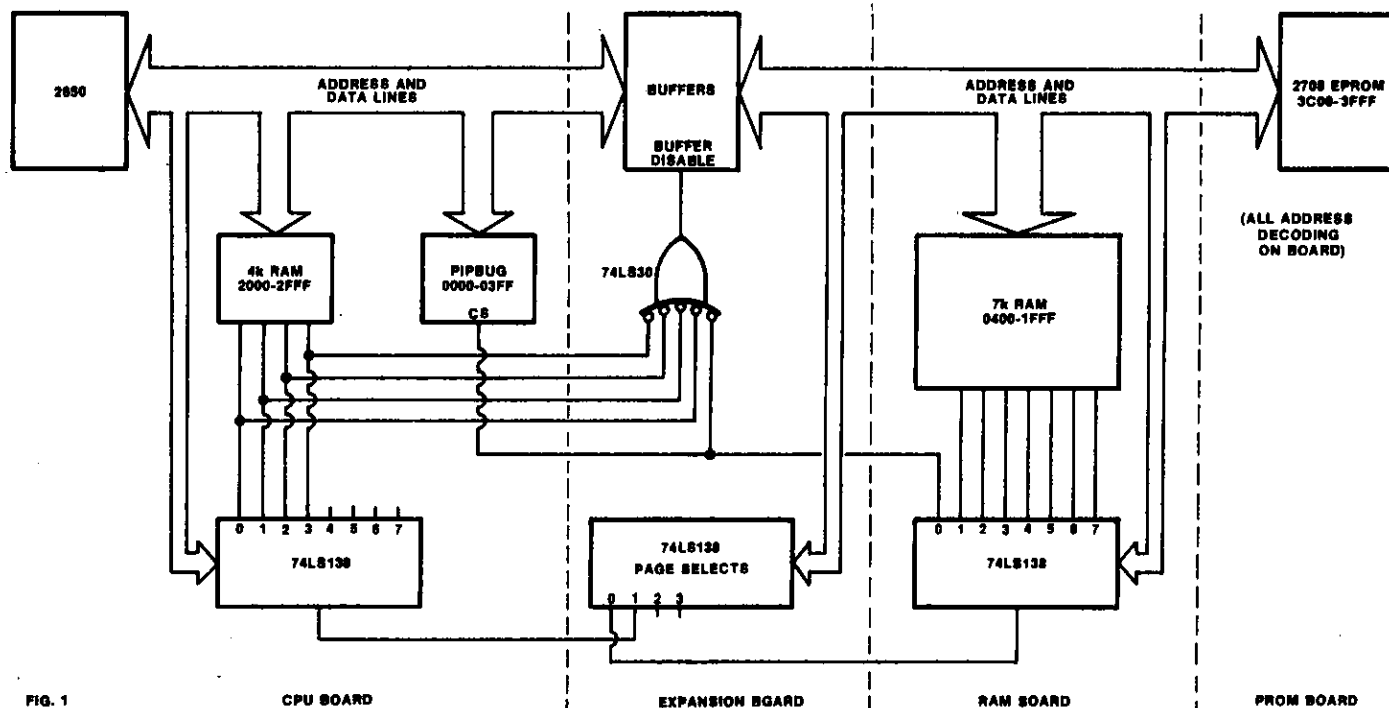


FIG. 1

CPU BOARD

EXPANSION BOARD

RAM BOARD

PROM BOARD

The schematic diagram shows how the author's system is configured.

then play back the tape to be checked. The program will then read from the tape, and compare its contents with those of the appropriate section of RAM.

If all is correct, the program will respond with the message "OK". If a fault is found, the message "FAULTY" will be printed. This program can only be used to check 110 baud tapes produced by the PIPBUG Dump command. The RAM dumped must still be in memory when the verification is performed, of course.

The fifth and final program is called MOVE. It will shift a specified block of memory to any other location in memory. A memory block can be any size, and can be moved either upwards or downwards in memory by any amount. To use the program, type G3D3B AAAA BBBB CCCC cr.

A and B represent the start and finish locations of the block of memory to be moved, and C represents the new start location. The program will move the memory starting at A and ending at B so that it starts at C and ends at C + A - B. The original memory block will only be changed if the new locations overlap the old locations.

The MOVE program can be used to copy memory from one page to another page, and can also move blocks straddling page junctions. Memory locations will not be

destroyed if the new start location is the same as the old start location.

A number of useful subroutines are also included as part of the programs. If you branch to location X'3CF8, the message "OK" will be printed, and if you branch to location X'3D0E, the message "FAULTY" will be printed. In both cases control will return to PIPBUG after the message is printed.

A message printing subroutine is included at locations X'3CCB to X'3CDC. This expects R1 and R2 to point to the start of an ASCII message string. The string must be terminated by the null (X'00) character. If you enter this routine at location X'3CCB, the message will be printed on a new line, while if you enter at location X'3CCE, the message will be printed on the current line.

A subroutine called GPAR is located at address X'3C07. This uses the PIPBUG subroutine GNUM to get three parameters from the PIPBUG line buffer, and store them as bytes in locations X'2FFA to X'2FFF inclusive. The first parameter is stored in locations X'2FFA and X'2FFB, and is called START.

The second parameter is incremented, and then stored in locations X'2FFC and X'2FFD. It is called END. The third parameter, called NEW, is stored in locations X'2FFE and X'2FFF.

The subroutine INCRT is called at

location X'3C2A, and increments the value START. It then compares START with END, and sets the condition code bits accordingly before returning. The condition code is set to "less than" (10) if START is less than END.

Another useful subroutine is called PADR, and is called at location X'3C3C. It will print the value of START, as a four digit hexadecimal number, at the start of a new line. The address is followed by a single space.

A number of smaller subroutines are also contained among the programs, but these are rather specialised, and will not be used very often. Interested readers can use the disassembler to disassemble the listing, and hence locate them.

To burn the program into a 2708, simply load it into a convenient area of RAM, and use the program supplied with the Prom Programmer article (Jan 1979) to copy it into the PROM. The program contains absolute addresses, and will only run at the correct locations, starting at X'3C00. RAM must exist at locations X'2FFA to X'2FFF inclusive.

Note that the listing of the programs given in this article shows them stored temporarily in the RAM at locations X'0600-07BD. This should be a convenient place to store them initially in most systems, before burning them into your PROM.



# Memory test routine

Here is a memory diagnostic routine for your 2650 Mini Computer. It will exercise each and every bit in a specified memory range with four distinct tests, and produce a printout of any faulty locations. It can also be used to track down intermittent faults.

by DAVID EDWARDS

Memory testing can be a very tedious and time consuming process, so most operators of small systems simply assume that all is OK, and get on with writing programs. But when a program you have triple checked and are sure is OK fails to operate correctly, you start to wonder about your memory.

Ninety-nine times out of a hundred, of course, the memory is working correctly, and the bug is in your program (moral: check, check and check again, and if you can get a second opinion, do so!). But what do you do if the program still refuses to operate correctly?

Well, you can always employ the old standby, the walking finger test. This involves placing your index finger in turn on all of the memory chips. The chip (or chips) that sends you running to the first aid cabinet is then faulty. Don't laugh, this does work, and I have used it in the past.

But this test will not show up faults like open circuit address or data lines, or short circuits between adjacent PCB tracks. This type of fault is quite common on large memory boards, as they

have more and more memory crammed onto them.

In these situations, what is required is some sort of software test routine which will exercise all memory locations of interest, and provide clues as to where the fault is. This is the function of the program described in this article.

The tests described here are based on those presented by Charles E. Cook, in the October 1977 issue of the US magazine, "Kilobaud". Two of the tests are quite simple, and check that each location can store and read back both a null (X'00) and a delete (X'FF).

The third test is known as the "walking bit test", and is perhaps the most important test. It verifies the "changeability" of each bit of the test location, by storing first the pattern 00000001, then 00000010, and so on up to 10000000, each time checking that only the correct pattern can be read back from the memory. The test bit (the 1) has been "walked" through the test byte.

The fourth test is really a combina-

tion of the three earlier tests. The whole of the test area of memory is first cleared, and then tested for correct clearing (this is the first test). Next, the walking bit test and the delete test are performed on the first test location. Then before these two tests are carried out on the second location, it is tested to see if it is still zero. If it is not, then there is obviously a memory fault of some type or other.

This process is repeated in turn throughout the test memory area, and forms the fourth test.

In order for the operator to be able to use these test results, it is necessary to know not only the type of faults encountered, but also their locations. To simplify matters, we have called the first test the Z test, the second the L test, the third the W test, and the fourth the S test. Then all the program has to do is print out the code letter of the test, followed by the appropriate address.

A flowchart for the basic test routine is shown in Fig. 1. Test S is carried out at the start of the main loop. The failure sections incorporate the error message printing routines, and produce a listing five entries wide, which can be accommodated on a 32 character-per-line VDU.

If the test routine is run once, it will catch and record all permanent faults, but is unlikely to give any indications of intermittent faults. To catch this type of fault, we must repeat the basic test routine a large number of times.

It would be wise, of course, to arrange that once a fault has been detected, that the program stops at the end of the current basic test. If this is not done, then there is a fair chance that you will be rewarded with a great screed of endlessly repeated error message sets, whereas only one set is required.

The complete program, incor-

```
0440 09 1E 0A 1D DA 02 D9 00 C9 16 CA 15 17 3B 71 E9
0450 0D 16 EA 0B 17 09 05 0A 04 3B 6D 17 00 00 00 00
0460 00 00 00 04 5A 1B 0C 04 53 1B 08 06 40 04 57 1B 02
0470 04 4C 3F 02 B4 09 69 3F 02 69 09 65 3B FA 04 20
0480 3B F1 FB 05 3F 00 8A 07 05 04 01 C8 01 17 00 76
0490 40 77 02 75 18 3F 02 DB C9 42 CA 41 3B F8 DA 02
04A0 D9 00 CD 04 5E CE 04 5F 3B EC CA 62 3B D7 07 05
04B0 3F 04 55 20 CC 84 60 3F 04 4D 1A 77 3B F3 0C 34
04C0 60 BC 04 62 3B F2 1A 76 3B E7 0C 84 60 BC 04 66
04D0 06 80 CE 84 60 EE 84 60 BC 04 6A D2 9A 74 04 FF
04E0 CC 84 60 EC 84 60 BC 04 70 3B CD 1A 5D 0E 04 8E
04F0 1E 04 B0 FA 00 CA F7 5A F8 9B 22
```

FIG. 2

This is a hexadecimal listing of the 2650 memory test program. You can use the disassembler program to produce a mnemonic listing of it.

## 2650 memory test routines

porating all of these points, is given as a hexadecimal listing in Fig. 2. It occupies locations X'440 to X'4FA inclusive, and is not easily relocated. It uses PIPBUG routines COUT, BOUT, CRLF and GNUM.

To be able to use this program, the memory area it occupies must be working correctly, and so must the processor. If you are not sure about this, try it anyway; if it works, then all is OK. Otherwise, you will have to do some fault-finding and corrections first.

To call the program, type G48F XXXX YYYY ZZ cr, where X is the start address of the memory range to be tested, and Y is the end address. Remember that the existing contents of the test area will be destroyed, and that you cannot test the area of memory occupied by the test program.

The parameter Z determined how many basic tests are to be carried out. X'01 gives one test, X'02 gives two, and so on up to X'7F, which produces 127 tests. All negative numbers such as X'80 and X'FF, produce an unlimited number of tests, terminated only when an error is detected.

The first time you use the program, specify only one test. Any errors you get will almost certainly be permanent faults, and should be found and corrected first. Only when this has been done should you attempt to trace intermittents using multiple tests.

In these initial tests, it may be advantageous to test only small amounts of memory at a time, say 1K blocks. This will allow you to isolate any faults more rapidly.

At this stage, you are probably wondering what all the rather strange lists of error locations mean, and how they can be used to locate faults in your memory. Well, simple faults should show up as easy to understand patterns.

For instance, if a data line to a particular chip is open, then all locations in this chip should fail the W and L tests. Similarly, if an address line to a particular chip is open, then we would expect test S to fail at all locations where this address line would normally go high. This is because the open line will normally float high, so that when we address lower bytes, we will actually write into higher locations, and will get an S message when we do address these bytes.

Further information on the types of faults which can occur in memory, and the results they produce with our test program, can be obtained from Cook's article. In any case, you will have to play at being a detective, and apply a little deductive reasoning.

Finally, a few detailed comments on the program for those who may wish to modify it. The start, end and current ad-

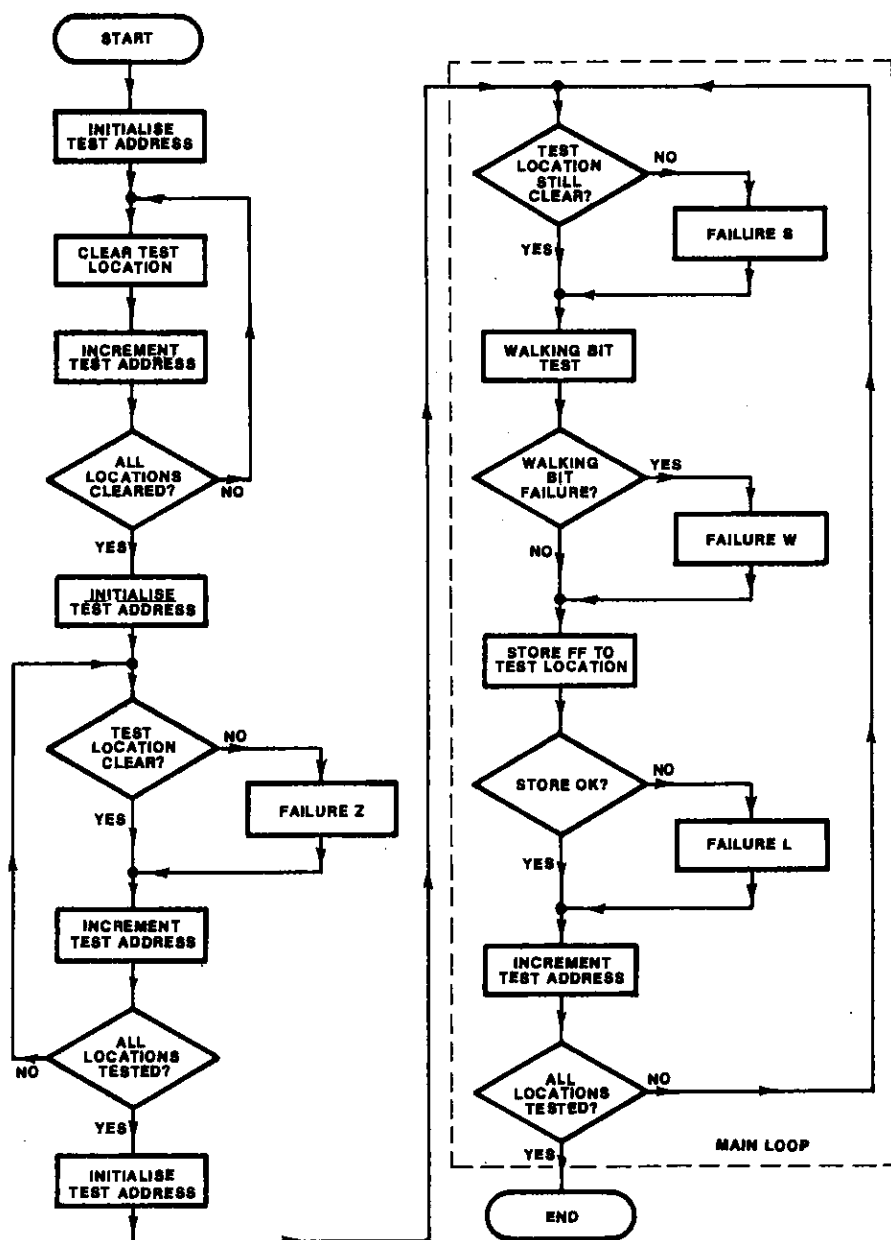


FIG. 1 BASIC TEST ROUTINE

Readers with systems based on CPUs other than the 2650 can use this flowchart to write their own diagnostic routines.

addresses are stored in locations X'45C to X'461, while the number of tests is stored in location X'48E. The number of error messages on each line is specified in locations X'488 and X'4AF.

To remove the auto-stop facility when errors occur, change locations X'48B and X'48C to the NOP code, X'C0. If you wish to obtain an error message every time the walking bit test fails, rather than just once for each walking bit test, change locations X'46A and X'46B to NOPs.

By changing locations X'4ED to X'4F2 inclusive to NOPs, you can delete the repeat forever facility, and obtain a maximum of 256 basic tests (specify X'00 in the calling line).

In conclusion, I wish you happy fault hunting, and successful debugging of your own programs. Because once you have assured yourself that your memory is OK, then you realise that the reason your program won't run correctly is because you have written a bug into it!

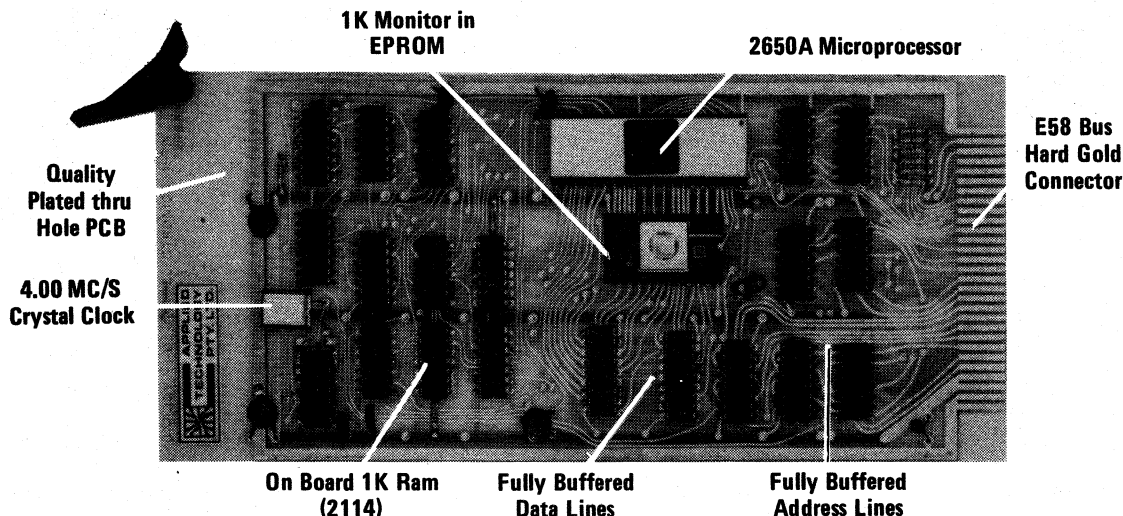
---

## Notes & Errata

**2708 PROM PROGRAMMER** (February 1979, File No. 2/CC/35): In the overlay diagram on page 89, the 0.01 $\mu$ F and 0.0047 $\mu$ F capacitors connected to pins 14 and 15 of the 74LS138s should be exchanged. The circuit diagram is correct.

To prevent spurious triggering of the monostable chain, we have found it advisable to ground the clear line (pins 3 and 11) of the 74LS138s during reads from the PROM. This is best achieved by using a three pole instead of a two pole switch for the read/program switch.

**Educational, Hobbyist, Business, Industrial, Microcomputer users...**



## ...EUROCARD 2650: a professional quality, expandible single card computer engineered to meet today's needs.

### COMPLETE COMPUTER

The DB1001 is the heart of an incredibly flexible computer system based on the 2650 microprocessor. Designed by BOB ARMSTRONG the DB1001 features; on board 1K RAM, 1K EPROM monitor, serial I/O, 4.00 crystal clock, fully buffered address and data lines.

Memory expansion and extra I/O devices can be readily connected using the E58 BUS which is also Z80 and S-100 compatible. The 1K EPROM can be readily reprogrammed to various operating systems such as 1200 BAUD PIPBUG or BINBUG V3.6.

The DB1001 is available in kit form or assembled and tested. Conversion kit is available to convert the EA2650 starters kit (DB1001 Kit \$135.00\*)

### DATA/BYTE 100: ENORMOUS EXPANDABILITY

DATA/BYTE100 is the system configuration using the DB1001 CPU card. Additional memory and I/O cards

readily plug into a mother board to produce an enormously versatile main-frame which can also incorporate floppy disc drives and high speed printer. By selecting from the individual modules you can design a DATA/BYTE 100 system to meet your exact specifications.

### NOW AVAILABLE

DB1006; a 6K RAM card which with the 1K RAM and 1K ROM on the DB1001 becomes a full 8K system. (Kit \$140.00\*)

DB1008; a 8K static RAM card configured as 2 separate 4K byte blocks selected with DIP SWITCHES for each address boundary. (Kit \$175.00\*)

DB1048; a dual 4/8K ROM and high speed cassette interface card. Accepts 2708 or 2716 EPROMS with your resident software. Also contains 2708 containing the full software to generate the cassette interface and digitally controlled dual cassette system together

with full file handling. (Kit \$130.00\*)

DB1500; a plug in power unit supplying 5V @ 5A regulated, -5V, ±12V 750mA from an external DB1505 transformer and bridge rectifier. (Kit \$45.00, transformer \$27.00\*)

DB1202; a wire wrap card for custom applications. (Kit \$25.00)

DB1203; an extender card for trouble shooting the E58 BUS. (Kit \$25.00)

### EASY TO USE:

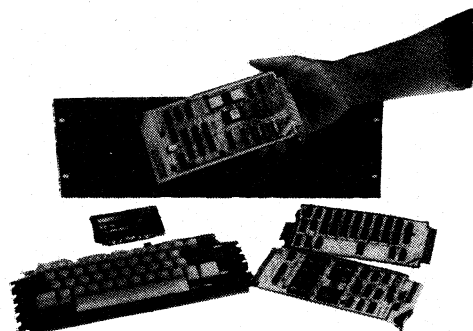
No matter what your application, DATA/BYTE 100 system is easy to use and understand. You can select from the extensive software base for the 2650 or use the newly released MICRO-WORLD BASIC which is a powerful MICROSOFT™ compatible 8K BASIC which will run in ROM or RAM on your system. The soon to be released 16K dynamic memory card and floppy disc controller will enable you to build your system to over 100K capacity!

\*Prices include sales tax. Please allow \$2.00 post and package.

**FOR MORE INFORMATION  
SEND \$1.00 FOR OUR  
1979 CATALOGUE:**



**APPLIED TECHNOLOGY, 1A PATTISON AVE., WAITARA. (02) 487 2711 (3 lines)  
SHOWROOM 9-5 MON TO SATURDAY  
MAIL ORDERS P.O. BOX 355, HORNSBY 2077.**



# Faster dumping & loading for the 2650

Here are some utility routines which will enable your 2650 system to dump programs, verify and reload them — all at 300 baud and using a binary format. This gives dumping, loading and verifying times roughly one sixth of those using PIPBUG's 110 baud hex format. The routines are intended for storage in a 2708 EPROM along with the utility routines described in March 1979.

by **DAVID EDWARDS**

Once your 2650 system is up and running, one of the first things you discover is that a lot of your time can be spent waiting while programs are dumped to or read from cassette tape. So naturally, any means of speeding up this process is most welcome.

We have presented 300 baud routines in the past, but these have mainly been intended for use with a PIPBUG format bootstrap loader, rather than to be stored in ROM. They also provided an "autostart" facility, where a program could be arranged to begin executing automatically as soon as it was loaded.

The present author feels that in a small cassette-based system, such as the majority of 2650-based systems current-

already existing in the ROM (see March 1979), and thus minimise the amount of code to be stored; it also ensured that the routines would be ROM compatible.

In fact, the finished routines require only 251 bytes storage, which still leaves a total of 327 bytes unused in the 1k 2708 EPROM. 6 bytes of RAM are required as a scratch pad, at locations X'2FFA to X'2FFF, but this is the same RAM as used by the earlier routines.

The recording format used by the new routines is shown in the diagram. As the routines were intended only for use with cassette tapes, the leader and trailer consist of 10 second periods of continuous mark. Only a single block is used for each dump, and it is nearly im-

possible to ensure that the start and end addresses are read in correctly from the tape. The second BCC checks for a faulty data byte.

The format used differs from that used by PIPBUG, in that both start and end addresses are specified initially on the recording for the memory area to be dumped. This change was made solely because it suited the existing ROM routines.

The routines are intended to occupy locations X'3DBE to 3EB8, as shown in the listing. However I suggest that you use the hex input routine to load them initially into another area in your RAM (say X'1DBE—1EB8). The PROM programming program given in the February 1979 issue can then be used to store them into the EPROM at the correct addresses.

The first section of the listing, from locations X'3DBE to X'3E01 inclusive are the actual 300 baud input and output routines, called 3IN and 3OUT. These are completely self contained, and are fully relocatable without modifications, as all relative addressing is used. They are written as subroutines, and are equivalent to CIN and COUT of PIPBUG. The calling address for 3OUT is X'3DBE, while that for 3IN is X'3DE4.

3OUT and 3IN can be used to communicate with your terminal at 300 baud. The bit rate is set by the LODI instructions at locations X'3DDB and X'3DDF, and assumes a 1MHz clock rate.

The remainder of the space is occupied by the DUMP, LOAD and VERIFY routines. To dump a program, type G3E02 AAAA BBBB cr, where A is the start address of the memory area to be dumped and B is the end address. The dump will include locations A and B. A ten second blank leader is provided at the start of the dump, with a similar sized trailer. A 4k dump will take just under three minutes.

To verify a tape, rewind it, type G3EA2 cr, and then start the tape. The contents of the tape must still be stored in memory of course, as the verification consists of comparing the data from the tape with the corresponding data still in memory. The program will respond with "OK" if the tape is correct, or "FAULTY" if a BCC or data error is detected.

To load a tape, type G3E53 cr, and

```
3DE5 77 1C
3DC2 C2 05 03 74 47 3E 14 52 1A 74 74 40 13 02 76 40
3DD0 F9 73 39 77 75 40 33 03 75 17 74 B5 F3 7F 74
3DE7 E5 F8 7E 17 77 10 05 00 06 09 12 1A 77 35 70 12
3DF0 1A 72 3E 67 12 44 30 51 61 C1 FA 76 3B 5D 01 75
3E00 12 17 3F 3C 27 3F 3E 43 20 CC 9C FF 04 3A 30 93
3E10 0D 0F FA 3B 2C 0D 0F FE 3F 27 0D 2F FC 3F 22 0D
3E20 0F 0D 3C 1F 0C 9C FF 3F 3D 3E 27 CC 9C FF 0D 3F
3E30 FA 3E 0F 3F 3C 2A 1A 75 7C 9C FF 3F FB 3F 09 9B
3E40 22 3F 02 3D 01 3E E1 17 20 01 76 11 F3 7E F9 7C
3E50 FA 7A 17 33 26 9C 3D 0F 37 1C 3E 13 CD 5F FA 3F
3E60 3C 2A 1A 76 3B 9A 2C 9C FF 1C 3C FB 1F 3D 0F 3F
3E70 3F C1 3F 02 3D 17 20 CC 9C FE 17 76 40 77 02 3F
3E80 3D E4 E4 3A 27 79 33 6E 3B 65 CD 0F FA 3F 60 CD
3E90 7F F5 3B 5B CD 7F FC 3F 56 CD 0F FD 3D E2 5C 9C
3EA0 FE 17 33 57 9C 3D 02 3B 4D 3E 44 FD 3F FA 9C 3D
3EB0 7F 3F 3C 2A 1A 73 1F 3F 64
```

Here is a full hex listing of the two 300 baud routines.

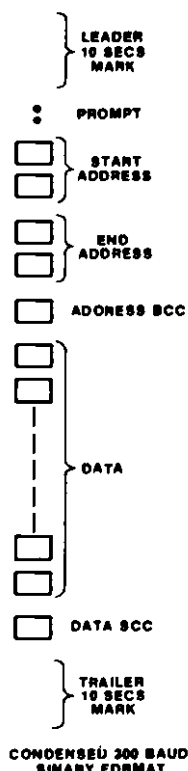
ly in use, an autostart facility is not a great deal of use. This is because many small systems have no easy means of automatically stopping the tape transport once a load has been completed. The tape transport must really be stopped by hand, before the loaded program is started.

For this reason I have chosen to write new routines from scratch, specifically to be stored in an EPROM. This made it possible to use some of the routines

possible to separate blocks in a cassette recording (unlike paper tape). In addition, gaps between blocks take up additional time during both dumping and loading.

No provision for autostarting is made. At the end of a load, control is passed back to PIPBUG. A colon (:) is used as the prompt to signify the start of the block.

Two block checking code (BCC) bytes are included. The first one is used



*This is the recording format used. All numbers are in binary.*

then start the tape. If a BBC error is detected in the addresses read from the tape, the message "FAULTY" will be produced, and the load will stop. Assuming the correct addresses are read from the tape, the load will proceed.

Once the data file has been read in, the data BCC is checked. If the BCC from the tape agrees with the calculated BCC, the message "OK" is printed. A mismatch will produce the message "FAULTY".

I have found the routines to be quite reliable, and have made quite a few 4k memory dumps with complete success. The reduced loading time is very convenient, allowing quite large programs to be reloaded very quickly.

The routines use the PIPBUG routine CBCC, and the existing ROM routines GPAR, FAULTY, OK and INCRT. Only 4 instructions require changes to relocate the program; these are located at addresses X'3E05, 3E27, 3E7F and 3EB6. Other absolute instructions in the programs point to locations in PIPBUG, the RAM buffer area, and the existing ROM.

To burn the programs into your 2708 EPROM, load them into a convenient area of RAM, as well as the PROM program. It is not necessary to reprogram the complete PROM: simply program in the new routines at the correct locations.

Note that in order to allow the routines to operate correctly, it is necessary to disable the monostables on the PROM board when in the read mode, as detailed in the Notes and Errata section of the March 1979 issue.

# 2650 mini assembler simplifies programming

Here is a handy "real time" assembler program for small 2650 microcomputer systems. You can use it to load programs directly into memory in mnemonic assembly language — much faster, easier and more reliable than having to do all the detailed coding and displacement calculations yourself!

by JAMIESON ROWE

Programming a computer in machine language tends to be a very slow and tedious business. If you're doing it this way at the moment, you'll know what I mean. It can be challenging enough to work out the basic flow of a program — then you have to sit down and painstakingly slog through the coding, instruction by instruction.

But time and tedium aren't the only problems. When you try running such a program coded by hand, the odds are that you'll find quite a few "bugs" caused by coding errors and mistakes in working out relative addressing displacements.

People using larger computers generally don't need to worry about such problems, because they don't have to program in machine language. In fact many couldn't do so even if they wanted to (which is unlikely), either because they've never learned how or because the operating system on their computer has no provision for loading

or running programs in this form!

The closest such folk ever need to come to machine language is assembly language programming, using easy-to-remember mnemonic symbols for the various instructions. An assembler program running in the computer itself is then used to translate this symbolic version of the program into machine language. The assembler takes over all of the detailed coding, and works out all of those tedious displacements. Not only that, but it does them much faster and far more reliably than mere humans!

Assemblers for some microcomputer systems have been available for quite a while now, giving users of these systems most if not all of the advantages possessed by larger systems. For industrial and commercial users of the 2650 microprocessor, Signetics themselves provide a "cross assembler" — an assembler for 2650 code which itself runs on another machine.

For smaller 2650 systems, more conventional "resident" assemblers have recently become available. A limited-facility "line" assembler called Prometheus was developed by the British Mullard company, and made available in a special ROM/RAM application card. However it was rather too expensive for hobby applications. Similarly an assembler was developed within the 2650 Users' Group in Sydney, but was memory-orientated and required quite a deal of RAM memory. Neither assembler was really well suited for small hobby systems.

Now for the good news. In this article, you will find details of a new 2650 assembler which I believe is almost ideal for small hobby systems. It occupies only 1300-odd bytes, so that it should fit into almost any 2650 system. Yet it will let you perform convenient and fast assembly of programs, from your terminal keyboard and in real time. You type in the mnemonics; it works out the code and plugs it into memory.

As you might expect, it is not a full-scale assembler like those you would find on large systems. It is basically a line assembler, which treats each instruction as a separate entity. But it does offer a very useful feature not found on many small line assemblers: limited forward referencing, which lets a branch instruction reference a memory location not yet known. This means that once you get used to its limitations, you can do almost as many things with this assembler as you can with its bigger brothers.

Incidentally I can't take much of the credit for this assembler. I haven't written it from scratch, but have developed it from a small assembler called PIPLA written by the software people at Signetics. I came across PIPLA last year when I toured the Signetics plant during my trip to California.

The people at Signetics told me they had written PIPLA to go into a special ROM device along with a modified and enhanced version of PIPBUG. When I showed interest in it, they let me have a copy along with a source listing.

I didn't have much of a chance to look closely at PIPLA during the trip but was able to do so when I came home. It didn't take long then to make a rather important discovery. Not unex-

```
*G1600
2650 LINE ASSEMBLER

0440.*THIS IS A DEMONSTRATION
0440.*
0440.*
0440.  ORG 500
0500.  DATA S 14
0502.  LODI,R3 FF      SET UP R3 AS INDEX
0504.  LODA,R3 **500    FETCH CHAR
0507.  COMI,R0 00      CHECK IF EOF (NUL)
0509.  BCTA,EQ 01      LEAVE IF FOUND
050C.  ZBSR *20        OTHERWISE GO PRINT
050E.  BCTR,UN 504     & CONTINUE
0510.01 ZBSR *25      END: GIVE CRLF
0512.  ZBRR 22        & LEAVE--RETURN TO PIPBUG
0514.  ASCI "HELLO THERE!"
0520.  DATA 0
0521.  END

*G500
HELLO THERE!
```

Fig. 1: A demonstration of the mini assembler in action. As you can see, a program may be run immediately following assembly.



pectedly, PIPLA used quite a few utility routines from the modified PIPBUG — but the modified PIPBUG was so different from the familiar old PIPBUG that the two were virtually incompatible.

Obviously PIPLA in its original form was not going to be all that much use to all those 2650 users who were already committed to the old PIPBUG. If it was to be of value to such people, someone was going to have to sit down and convert it to use the routines in "old PIPBUG"....

Well, the rest is fairly obvious. The job took a while, as it had to be fitted in between more urgent things. There were a few complications, because some of the required routines in the modified PIPBUG were so different from those in the old PIPBUG that the routines in "old PIPBUG" could not easily be used at all. I had to add these to PIPLA itself, at the same time reducing the size of PIPLA wherever possible to minimise the increase in memory space.

Eventually I finished the basic conversion job, and after the inevitable debugging the modified PIPLA began running on my system with "old PIPBUG". But this wasn't quite the end of the story.

Once you got used to its limitations, it was a very handy piece of software. But there were a few mildly irritating little shortcomings. When you called it, it simply printed out a suggested initial "origin" or starting address for assembly. Wouldn't it be nicer if it announced itself with a suitable message?

Similarly, it lacked a facility for accepting numbers and other data constants, in hexadecimal. Wouldn't it be nice if it had a "DATA" directive like bigger assemblers?

To cut a long story short, these facilities were added and the result is presented here. Based on PIPLA but with quite a bit of modification and a couple of additional features, it is quite a capable little assembler. Certainly you should find it a big step forward in speed and convenience if you're still programming in machine language.

What will it do? Well, it will accept all of the standard 2650 instruction mnemonics — LODA, STRR, BCTA, BSTR and so on. It can also recognise all of the commonly used register/condition code mnemonics R0, R1, R2, R3, P, Z, N, LT, EQ, GT and UN. It will accept symbols for indirect and indexed addressing, up to 10 label symbols for forward referencing, four different pseudo-operation or assembler directives, and comments.

The input format required by the assembler for the symbolic source lines is:

LBL OPC R/C SYM OPND

where the symbols have the following meaning:

```

1500 04 0C 18 02
1500 45 1F 6D 04 2A 1F 17 5B FA 0A 98 0F 3F 1A 95 CE
1500 84 0D 3B 0F EF 0A 29 9A 25 1B 71 3B F0 1B 1D 02
1500 69 1A CD 0D 04 0D 0E 0A 02 D9 0B 0F 0F 00 A4
1600 20 07 14 CF 5A 40 5B 7B 3F 1A 55 C0 3B F0 0D 04
1610 0D 3B DC 0D 04 0E 3B D7 04 2E BB A0 3B D3 0C 1A
1620 02 E4 2A 18 69 E4 40 98 3C 0F 3A 02 A4 30 1E 02
1630 50 C3 E7 09 19 F9 D3 06 01 0F 7A 40 CC 04 0F C1
1640 0F 7A 41 CC 04 10 61 18 1A 0C 84 0F CF 7A 40 0E
1650 E4 0F CF 7A 41 0C 04 0D CC 84 0F 0C 04 0E CE E4
1660 0F 1B 56 07 02 20 CC 04 2A 3B BE 3F 17 7A CC 04
1670 11 60 9A 2B 44 0F 1C 00 22 FA 02 98 9C 3B AA 0F
1680 7A 02 C2 0F 3A 02 EF 0A 29 9E 16 0E E2 18 FB CC
1690 84 0D 02 3F 15 F3 C2 1B 6A 15 D8 C0 C0 1A 95 CE
16A0 84 0D E4 10 9A 21 87 01 3F 17 6B 0F 7A 02 E4 40
16B0 99 0A 3F 17 7A 84 10 18 05 1F 02 50 3B DF 46 03
16C0 0C 84 0D 62 CC 84 0D 3B CB 0C 04 11 FA 01 1C 16
16D0 0E FA 02 1C 17 22 3B D1 0F 7A 02 E4 30 9A 1B 87
16E0 01 05 FF ED 37 CF 18 06 E5 0A 1A 77 1B CC D1 D1
16F0 D1 D1 D1 6D 04 2A C9 FC 1B 5C E4 40 98 2A 0F 3A
1700 02 A4 30 1E 02 50 C3 E7 09 19 F9 D3 0F 7A 40 C1
1710 0F 7A 41 C2 0C 04 0D CF 7A 40 0C 04 0E CF 7A 41
1720 1B 07 3F 1A 95 0C 04 11 FA 08 18 AD FA 04 98 24
1730 77 09 A6 01 A5 00 77 01 AE 04 0E AD 04 0D 75 08
1740 18 0D 85 01 9C 02 50 F6 C0 98 FA 46 7F 1B 05 04
1750 C0 42 98 F1 6E 0A 2A 1B 0A 15 CC CD 84 0D 02 3F
1760 15 F3 C2 CE 84 0D 3B FB 1F 16 0E 0A 20 FB 00 EF
1770 3A 02 18 7B EF 04 29 9A CC 17 06 FC A7 01 0F 3A
1780 02 EB F2 9A 0D E4 30 1A 09 CE 79 40 DA 70 87 01
1790 1B 07 04 20 CE 79 40 DA 7B CF 04 28 75 01 77 08
17A0 05 17 06 D4 CD 04 0F CE 04 10 07 FF 0F A4 0F 1C
17B0 02 50 EF 7A 3C 18 06 86 06 85 00 1B 67 E7 03 1A
17C0 6B 0F A4 0F C2 0F A4 0F 0F 04 28 75 08 17 00 2C
17D0 2B 2D 23 2A 52 30 20 20 00 F0 52 31 20 20 01 F0
17E0 52 32 20 20 02 F0 52 33 20 20 03 F0 50 20 20 20
17F0 01 F0 5A 20 20 20 00 F0 4E 20 20 20 20 20 20 20
1800 20 20 02 F0 45 51 20 20 00 F0 47 5A 20 20 01 F0
1810 55 4E 20 20 03 F0 45 4E 44 20 00 80 4F 52 47 20
1820 00 81 41 53 43 49 00 82 4C 4F 44 5A 00 01 4C 4F
1830 44 49 04 02 4C 4F 44 52 08 04 4C 4F 44 41 02 08
1840 53 54 52 5A C0 01 53 54 52 52 C8 04 53 54 52 41
1850 CC 08 49 4F 52 5A 60 01 49 4F 52 49 64 02 49 4F
1860 52 52 68 04 49 4F 52 41 6C 08 41 4E 4A 5A 40 01
1870 41 4E 44 49 44 02 41 4E 44 52 48 04 41 4E 44 41
1880 4C 08 45 4F 52 5A 20 01 45 4F 52 49 24 02 45 4F
1890 52 52 28 04 45 4F 52 41 2C 08 42 43 5A 52 18 04
18A0 42 43 54 41 1C 0C 42 43 46 52 98 04 42 43 46 41
18B0 9C 0C 43 4F 4D 5A E0 01 43 4F 4D 49 E4 02 43 4F
18C0 4D 52 E8 04 43 4F 4D 41 EC 08 41 44 4A 5A 80 01
18D0 41 44 44 49 84 02 41 44 44 52 88 04 41 44 44 41
18E0 8C 08 53 55 42 5A A0 01 53 55 42 49 A4 02 53 55
18F0 42 52 A8 04 53 55 42 41 AC 08 52 45 54 43 14 01
1900 52 45 54 45 34 01 42 53 54 52 38 04 42 53 54 41
1910 3C 0C 42 53 46 52 B0 04 42 53 46 41 BC 0C 52 52
1920 52 20 50 01 52 52 4C 20 D0 01 43 50 53 55 74 12
1930 43 50 53 4C 75 12 50 50 53 55 76 12 50 50 53 4C
1940 77 12 42 52 4E 52 58 04 42 52 4E 41 5C 0C 42 49
1950 52 52 D8 04 42 49 52 41 DC 0C 42 44 52 52 F8 04
1960 42 44 52 41 FC 0C 42 53 4E 52 78 04 42 53 4E 41
1970 7C 0C 4E 4F 50 20 C0 11 48 41 4C 54 40 11 54 4D
1980 49 20 F4 02 57 52 54 44 F0 01 52 45 44 44 70 01
1990 57 52 54 43 B0 01 52 45 44 43 30 01 57 52 54 45
19A0 D4 02 52 45 44 45 54 02 5A 42 53 52 BB 10 5A 42
19B0 52 52 9B 10 54 50 53 55 B4 12 54 50 53 4C B5 12
19C0 4C 50 53 55 92 11 4C 50 53 4C 93 11 53 50 53 55
19D0 12 11 53 50 53 4C 13 11 42 53 58 41 BF 1C 42 58
19E0 41 20 9F 1C 44 41 52 20 94 01 4C 44 50 4C 10 1C
19F0 53 54 50 4C 11 1C 44 41 54 41 00 84 00 00 00 00
1A00 00 00

```

```

1A55 05 1A 06 6C 3F 00 A4 07 FF 0F A4
1A60 0D 18 04 BB A0 1B 77 05 04 06 40 17 32 36 35 30
1A70 20 4C 49 4E 45 20 41 53 53 45 AD 42 4C 45 52 0D
1A80 0A 0A 00 00 A4 30 1A 0A E4 0A 16 A4 07 1A 03 E4
1A90 10 16 1F 02 50 20 C1 C2 CC 04 12 0B FC 15 EF 04
1AA0 29 14 0F 7A 02 E4 20 98 02 DB 70 3B 57 D2 D2 D2
1AB0 D2 CE 04 28 A6 F0 62 C2 D1 D1 D1 D1 45 F0 08 F2
1AC0 44 0F 61 C1 04 01 C8 D1 DB 54 0A 0D 5E 07 00 E7
1AD0 3C 1C 00 1D 3F 02 86 E4 7F 98 0A 03 18 71 0F 5A
1AE0 02 BB A0 1B 6A 05 03 ED 7A C9 18 09 F9 79 CF 7A
1AF0 02 BB A0 DB 5A CF 04 2D CD 04 2A 07 00 9B A5

```

Fig. 2: A complete hex listing of the assembler. The gap from 1A02 to 1A54 is occupied by the input and labels buffers.

## 2650 MINI ASSEMBLER

LBL is an optional label; if present it must be one of the labels used in the operand field of a previous instruction, for forward referencing.

OPC is the instruction or pseudo-operation mnemonic; the standard 2650 mnemonics are used, as given in the Signetics manual.

R/C is the register or condition code, if one is required; either the symbols given earlier may be used, or a single-digit hexadecimal number.

SYM is a special symbol or symbols to indicate indirect addressing and/or indexing, if required.

OPND is the operand for the instruction; it may be a hexadecimal data number or an address, and if an address it may be given either as a hex number or one of the labels for forward referencing. In the case of relative addressing, the assembler expects an absolute hex address, and will calculate the required displacement. The only exception is for ZBRR and ZBSR instructions, where the actual displacement must be typed in.

Each of the above symbol fields should normally be separated from those adjacent by one or more spaces. If the label field is not used, a leading space is not required although one or

more spaces may be used if desired for appearance. The separator between the OPC and R/C fields may be a comma instead of a space, and the space between the SYM and OPND fields may be omitted if desired.

If the first character of a line is an asterisk (\*), the assembler assumes the line is a comment only and ignores it. A comment line may have up to 56 characters apart from the asterisk.

The symbols used to indicate indirect addressing and indexing in the SYM field are as follows:

'\*' Means indirect addressing.

⌘ Means normal indexing. Note, however, that when indexing is specified the index register must be given in the R/C field, unlike the normal assembler format. This is no real problem since R0 is always the implied source/destination register for indexed instructions.

'+' Means indexing with auto-increment. Again the index register must be given in the R/C field.

'-' Means indexing with auto-decrement. The index register must be given in the R/C field.

Where indirect addressing and indexing are to be specified in the one instruction, the two appropriate symbols

are used together with the indirect addressing symbol given first. For example:

LODA,R3 \*+8A0

which is a load indirect through address X'8A0, using R3 as the index register and with auto-increment. Thus R3 will be incremented and added to the address found in location 8A0 to generate the final effective address for the instruction.

The function of the label operators is to help you in writing forward memory references. That is, references in the operand field of instructions to locations in the program which have yet to be fed in, and are therefore not known in terms of their exact absolute address.

There are restrictions on the use of the label operators, as follows. They can only be used in the OPND field of branch instructions, and they cannot be used in relative addressing instructions. Nor can they be used with indirect addressing or indexing. This limits the use of the labels fairly severely, but they can still be quite handy.

Ten different label operators are allowed, represented by the symbols @0—@9. Each one can be used in the operand field of instructions any number of times before it is finally defined by specifying it in the label field of an instruction or pseudo-op. Note that all references to a label must precede its definition, due to the way in which the assembler handles the labels.

## 2650 MINI ASSEMBLER

However after being defined a label operator may be re-used again.

As mentioned earlier, the assembler recognises four different directives, or pseudo-operators. These are basically instructions to the assembler itself, rather than symbolic instructions to be assembled into machine code. The four directives recognised are as follows:

**ORG** is a directive to the assembler to reset its program counter; i.e., the pointer which the assembler uses to store the assembled program instructions into memory. The format of this directive is

**ORG nnnn**

where 'nnnn' is a hexadecimal number specifying the new program counter value. Leading zeroes are not required.

**ASCII** is a directive to the assembler to store in memory a string of alphanumeric characters, in ASCII code. Following the directive mnemonic the assembler skips any leading spaces, then takes the next character it finds as a string delimiter. All of the following characters up to the next occurrence of the delimiter character are then stored as an ASCII string. The actual string may be up to 52 characters long. The format for this directive is thus

**ASCII < delim >< string >< delim >**

**DATA** is a directive to the assembler to store one or more data bytes in memory, beginning at the location given by the current value of the assembler's program counter. The directive format is

**DATA nn nn nn nn nn nn nn ...**

where each 'nn' is a two-digit hexadecimal number, and the numbers are separated by spaces. If an error is made while typing a number, it may be corrected merely by typing in the two correct digits before the terminating space. Leading zeroes are not required. Up to 18 data bytes may be entered on a line if no corrections are made.

**END** is the directive which is used to indicate to the assembler that no further source material is to be assembled. When this directive is encountered the assembler returns to PIPBUG.

If desired, comments may be added after the operand field on most source lines, providing the comments are separated from the operand by at least one space. The only type of source line where this cannot be done is one consisting of a **DATA** directive, as the assembler searches to the end of the source line for data numbers for this directive. No special symbol is required

to distinguish comments following source instructions or directives.

The assembler resides in memory from location 15CC to 1AFE, inclusive. Part of this range is not used by the program itself, but is used as a line input buffer, scratchpad and label buffer area (1A02—1A54). The initial starting address is 1600, so after loading into memory the assembler is called by giving PIPBUG the command **G1600r** (where 'r' is carriage return).

When called, the assembler first types out an identifying message: "2650 LINE ASSEMBLER". It then types out a suggested initial origin, which is X'0440 — the start of the available RAM above PIPBUG's scratchpad area. If you don't wish the assembled program to start at this address, you can immediately change the program counter to another value by using the **ORG** directive.

You can now type in your program to be assembled, line by line. When you conclude each line with the usual carriage return, the assembler will attempt to assemble it. If you have made no format (syntax) errors and it can do so, it will indicate this and its ability to accept a further line by typing the new value for its program counter at the start of the next line. You thus get a continuous indication that all is well, along with an indication of the memory space being used by your program.

If you make a format error and the assembler cannot assemble the line, it will abort and return to PIPBUG via the '?' error message routine. After working out what went wrong, you can return to continue the assembly by

either re-starting at address 1600, or by starting at address 160E. The latter preserves any forward reference labels you may have been using, although the assembler's program counter is disturbed. You thus have to reset it with an **ORG** directive.

I have prepared a small demonstration of the assembler's use, which is shown in Fig. 1. As you can see the program assembled is a very short message printing routine which starts at X'0500, but its assembly illustrates most of the things you need to know about the assembler and the way it is used.

Note that the first three input lines are comments, which are effectively ignored by the assembler. Note also the way the assembler prints out the current value of its program counter at the start of each line, so that you can see how much memory the program is taking up. Needless to say you also make use of these addresses when typing in backward-referencing operands — an example of this is shown in the line commencing at address 050E.

Finally, note that after assembly, the program which had just been assembled was called from PIPBUG by typing **G500**. It then ran, typing out the simple message "HELLO THERE".

Needless to say, once you have assembled a program and checked that it runs, you can dump it in the normal way to cassette tape or paper tape using the normal PIPBUG dump routine.

Well, there it is — a small but very practical assembler which should make programming your 2650 very much easier. Incidentally for those who would like to analyse the assembler's operation in detail, full source listings will be available from our Information Service for a fee of \$4.00, to cover photocopying and postage.

# Lunar Lander game

This moon landing game program is written in TCT BASIC, and can be run on the 2650 Mini Computer. It is quite realistic, taking into account the moon's gravity, and the decreasing mass of the lunar lander as the fuel is used up. It also has limits on both the rate of fuel usage and the acceleration to which the lander is subjected.

by DAVID EDWARDS

Moon landing is a simple mathematical game which has been played on computers from the very early days. In its simplest form, as described here, all that is required is a terminal capable of displaying about 16 lines of text.

The scenario is that the operator (LEM pilot) is in the lunar lander a specified distance above the lunar surface. The LEM has a certain initial velocity, and a quantity of fuel. The

pilot has to specify when and how much fuel to use, so that the LEM can be made to land on the moon with zero, or at least minimal, speed.

It sounds quite simple, doesn't it! But you will probably change your mind once you have actually tried to do it, as there are a number of traps for unwary pilots.

First of all, you can simply run out of fuel before the LEM reaches the surface. Once this happens, the LEM simply

drops, and digs a big crater (this is quite soundless however, as there is no atmosphere on the moon to support soundwaves!).

If you specify too high a fuel rate, one of two things can happen. Firstly, you may overload the motor, causing a burnout, followed by a long drop to the moon, and another big crater. Or you could exceed the allowable G forces on the LEM. In this case, it will simply fall apart, and the pilot will proceed to the lunar surface unaided!

Assuming that you can avoid all these pitfalls, you still have to ensure that your landing speed is sufficiently low, because even though moon gravity is approximately one fifth of earth gravity, your inertia is still the same. In fact, to achieve a good landing, you need to have a terminal velocity of less than 1 metre per second, or about 2.2mph.

In fact, the only good point about this simulation is that it is not in real time, and you have lots of time to think between moves. A typical landing will take about 50 seconds of simulator time, and about 10 minutes real time.

Fig. 1 is a listing of the program. You will need about 2K of RAM to run it, apart from the 5K required by the TCT BASIC. Putting it another way, you will need to have page 0 full of RAM, apart from the 1K occupied by PIPBUG.

Load it in exactly as per the listing, remembering that the punctuation forms part of the program, and should not be changed. To start the program, simply type RUN. After the program name and trumpet blowing section, it will give you a small list showing initial height, velocity, time and fuel stocks. Velocity is measured positive downwards, i.e., towards the lunar surface.

The program will then expect you to type in a fuel rate in kg/s, followed by a duration in seconds. This is how you specify to the program what propulsive

```
0005 PR"":PR"LUNAR LANDER":PR"BY D.W. EDWARDS 3/9/78"
0010 FIX 2:T=0:V=-25:E=200:G=2:H=1000:M=10000
0015 GOSUB 100:IF R<500 GOTO 25
0020 PR"":PR"FUEL RATE TOO HIGH!":PR"Motor BURNS OUT":GOTO 50
0025 GOSUB 200:IF A<13 GOTO 35
0030 PR"":PR"G FORCES TOO HIGH!":PR"LANDER BREAKS UP":GOTO 75
0035 IF M<=5000 GOTO 45
0036 N=N-1:IF H<=0 GOTO 57
0040 IF N<=0 GOTO 15
0041 GOTO 25
0045 PR"":PR"NO FUEL LEFT!"
0050 PR"PREPARE FOR LANDING":R=0
0055 GOSUB 200
0056 IF H>0 GOTO 55
0057 IF V>=0 V=-V
0060 PR"":PR"TOUCHDOWN AT",T,"S":PR"TERMINAL SPEED =",V,"M/S"
0065 IF V>-1 GOTO 85
0066 IF V>-5 GOTO 90
0067 IF V>-10 GOTO 95
0070 PR"":PR"A NEW LUNAR CRATER",M*V*V/50000,"M"
0071 PR"DEEP WILL BE DISCOVERED SOON!"
0075 PR"":PR"DO YOU WANT TO PLAY AGAIN":
0076 INPUT ?$1:$2="NO":$3="YES"
0080 IF $1=$2 STOP
0081 IF $1=$3 GOTO 10
0082 GOTO 75
0085 PR"GOOD LANDING":GOTO 75
0090 PR"ROUGH LANDING":GOTO 75
0095 PR"LANDER DESTROYED":GOTO 75
0100 PR"":PR"HEIGHT =",H,"M":PR"SPEED =",V,"M/S"
0105 PR"FUEL LEFT =",M-5000,"KG":PR"TIME =",T,"S"
0110 PR"FUEL RATE (KG/S) ":
0111 INPUT=R:IF R>=0 GOTO 115
0112 GOSUB 120:GOTO 110
0115 PR"DURATION (S) ":
0116 INPUT=N:N=INT(N):IF N>0 GOTO 118
0117 GOSUB 120:GOTO 115
0118 RETURN
0120 PR"IMPOSSIBLE - TRY AGAIN":RETURN
0200 M=M-R:A=E*R/H-G:T=T+1:H=H+V*A/2:V=V+A:RETURN
```

FIG. 1

This listing of the Lunar Lander was written in TCT BASIC, but is adaptable to other types of BASIC.

LUNAR LANDER  
BY D.W.EDWARDS 3/9/78

HEIGHT = 1000.00 M  
SPEED = 25.00 M/S  
FUEL LEFT = 5000.00 KG  
TIME = 0.00 S  
FUEL RATE (KG/S) = 600  
DURATION (S) = 3

FUEL RATE TOO HIGH!  
MOTOR BURNS OUT  
PREPARE FOR LANDING

TOUCHDOWN AT 22.00 S  
TERMINAL SPEED = 69.00 M/S

A NEW LUNAR CRATER 952.20 M  
DEEP WILL BE DISCOVERED SOON!

DO YDU WANT TO PLAY AGAIN?YES

HEIGHT = 1000.00 M  
SPEED = 25.00 M/S  
FUEL LEFT = 5000.00 KG  
TIME = 0.00 S  
FUEL RATE (KG/S) = 480  
DURATION (S) = 10

G FORCES TOO HIGH!  
LANDER BREAKS UP

DO YOU WANT TO PLAY AGAIN?YES

HEIGHT = 1000.00 M  
SPEED = 25.00 M/S  
FUEL LEFT = 5000.00 KG  
TIME = 0.00 S  
FUEL RATE (KG/S) = 200  
DURATION (S) = 20

HEIGHT = 1046.44 M  
SPEED = -38.51 M/S  
FUEL LEFT = 1000.00 KG  
TIME = 20.00 S  
FUEL RATE (KG/S) = 0  
DURATION (S) = 100

TOUCHDOWN AT 77.00 S  
TERMINAL SPEED = 75.48 M/S

A NEW LUNAR CRATER 683.84 M  
DEEP WILL BE DISCOVERED SOON!

## FIG. 2

*Illustrated above is a printout showing how the program reacts to a variety of "wrong" inputs.*

force you require, and for how long. The program will then calculate your new height and velocity, and present these, along with the elapsed time and amount of fuel remaining.

All you have to do then is supply the appropriate numbers, till the program terminates. Note that only positive fuel rates and times are accepted, and that the program turns all times into integer numbers.

Fig. 2 shows some sample printouts of typical games. Note that all outputs have less than 32 characters per line, although the program listing does not. If you are using the Low Cost VDU (February and April 1977), the automatic carriage-return line-feed facility will let you see all of the listing as you feed it in.

If you let your family and friends play this game, be warned. It is very engrossing, and you may have trouble getting them away from it!

# A "Micro BASIC" for small 2650 systems

If you have a 2650 microcomputer with only PIPBUG and 4K bytes of RAM, you probably think it's too small to run even a cut-down version of BASIC. Well, not any more — you can now get an interpreter called "Micro BASIC" which will run in systems this small. Editor Jim Rowe reviews Micro BASIC in this article.

Not long ago, I received a 'phone call from a reader, Mr Alan Peek of Woolwich NSW, who told me that he had successfully written a "micro BASIC" interpreter for very small 2650 systems. As he was proposing to offer it for sale to readers, would I be interested in trying it out and perhaps publishing a short review?

It sounded interesting, so I asked for a few more details. He explained that he had written the interpreter to run in systems with as little as 4K of RAM, to allow those with such systems to be able to program them rapidly and easily for useful tasks. He had managed to squeeze the interpreter itself into a mere 1.6K bytes of memory, by using single-character commands, reverse Polish notation, and an efficient way of packing the source program into memory.

At my invitation Mr Peek sent a cassette of his interpreter to me a few days later, along with a copy of the literature he is supplying with it. Since then I have been able to spend some time using it and discovering its capabilities.

For convenience the program is best visualised as divided into two sections:

the interpreter proper, which translates and executes the source program in "run" mode, and a text editor which is used for feeding in, modifying and listing the source program.

The text editor has similar functions to those found in other interpreters, although they are used a little differently because of the different way that this editor packs the source statements into the RAM buffer. Unlike other interpreters, this one does not accept line numbers from the programmer — it supplies its own, which are attached to lines in simple incrementing order.

Doesn't this make it hard to insert extra lines, when you need to? No, you can use the editor functions to insert or delete lines as required. All that happens is that when you do this Micro BASIC simply re-numbers all of the lines.

It takes a little while to get used to this if you have been using a more conventional BASIC interpreter, but once you do it is just as convenient as the conventional approach.

As far as the interpreter itself and its operation are concerned, probably the most obvious differences from conventional BASIC and Tiny BASIC inter-

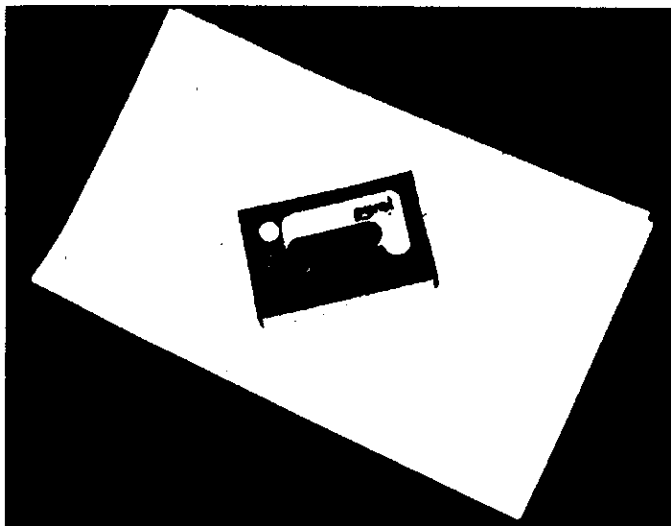
preters are the use of single-character statements and reverse Polish notation. But again these don't really take long to get used to, and many people prefer reverse Polish notation — as witnessed by the popularity of calculators which use it. Many people also like the ability to shorten BASIC statements to single characters, because it lets them pack in larger programs!

There are a few differences from normal BASIC in the actual statements, but not of a major nature. Instead of the familiar IF ... THEN statement, Micro BASIC has a "TEST" statement, but this functions in a similar fashion. Similarly string input and output statements ("A" and "O") and a "Memory (M)" statement which performs similar functions to the conventional PEEK and POKE.

Micro BASIC has a random number function, although this works in a fairly unorthodox fashion. When this function is reached during program execution, a "!" is printed out on the terminal and the operator is expected to press any key. The time delay before a key is pressed is used to generate a random number. Rather unusual, but then so are some other random number functions!

Two statements offered by Micro BASIC which are not found on many small BASIC interpreters are a variable increment and decrement. It also has a CALL statement, and the ability to have multiple statements on a line. Unlike most other BASICs you can also have comments anywhere on a line, even between statements.

A sample of a small program written in Micro BASIC is shown here so that



```

L1
1 P"MICRO BASIC NUMBER GUESSING GAME"
2 P "WHAT WILL BE OUR UPPER LIMIT " IA P
3 P"NOW PRESS A KEY" LA11--R,0-T ST= COUNT OF TRIES
4 P"RIGHT-HERE WE GO!"
5 P"GUESS= " IG LT1--T SINCREMENT COUNTS
6 TG>R P"TOO BIG" G5
7 TG<R P"TOO SMALL" G5
8 P"YOU GOT IT IN",T,"TRIES"
9 P"WANT TO PLAY AGAIN? I=YES,G=NO" IB
10 TB=I G3
11 P"BYE" SMUST HAVE BEEN NOS
12 E
13
>

```

LEFT: Micro BASIC comes as a cassette with accompanying literature. The early notes shown were handwritten, but those now supplied are typed. ABOVE: A sample program, written in Alan Peek's Micro BASIC.

```

G1
MICRO BASIC NUMBER GUESSING GAME

WHAT WILL BE OUR UPPER LIMIT ?100

NOW PRESS A KEY!*
RIGHT.HERE WE GO!
GUESS= ?50
TOO SMALL
GUESS= ?75
TOO BIG
GUESS= ?67
TOO SMALL
GUESS= ?71
TOO BIG
GUESS= ?69
TOO SMALL
GUESS= ?70
YOU GOT IT IN 6 TRIES
WANT TO PLAY AGAIN? 1=YES,0=NO?0
BYE
>

```

*How the sample Micro BASIC program looks when running on a small 2650 system.*

you can see how it looks. Note the comments, identified by dollar signs at each end. Also the input statements, represented by "I" characters, and the test statements ("T"). A listing is also given showing the same program when running.

The literature which comes with Micro BASIC includes a full source listing. This is all hand written, but includes plentiful comments. Alan Peek explains that he is happy for users to understand how the interpreter works, and to make mods and improvements if they wish. A generous attitude, to be sure.

The explanatory material supplied is quite helpful and easy to follow, although those with hawk eyes will be able to spot quite a few spelling errors. I did, but then that's part of my job! Despite this I think most people will find it tells them all they need to know about Micro BASIC.

In short, Alan Peek's BASIC seems a very practical piece of software, well suited for small 2650 systems despite a few unorthodox features. It seems good value for money at \$8.50 for a cassette with instructions and source listing, including postage.

You can get it from Alan Peek by writing to him at 10 Gale Street, Woolwich NSW 2110.





# A training system from Signetics: Instructor 50

Described by Signetics as a "desktop computer", the Instructor 50 has been designed primarily as a training tool. It offers a number of features not found on small evaluation systems, and comes complete with both a comprehensive set of training manuals and a tape cassette loaded with eight demonstration programs.

by JAMIESON ROWE

Since 1976 when microprocessors really began to "take off", many small microcomputer systems using them have appeared on the market. Some of these have been intended for the hobbyist, while others have been "evaluation" kits or systems intended to help engineers become familiar with the particular microprocessor concerned.

But very few systems have been designed specifically for training and educational purposes. This is a pity, because the concepts involved in microcomputer operation are relatively unfamiliar to many of the people who are going to have to operate them, program them, design them into equipment or service equipment which will use them.

Until now, those wanting to become familiar with microcomputer concepts have generally had to get hold of a small hobby or evaluation system, and largely use it to teach themselves by experience. Most such systems have been rather poorly supported by user literature, particularly when it comes to the introduction to basic concepts.

The Signetics "Instructor 50" system is an attempt to fill this very gap. It is a small desktop unit designed specifically for training, and comes complete with a comprehensive set of training manuals. Also supplied as part of the training package is a cassette tape with eight demonstration programs, ready to feed into the system via a standard cassette recorder.

Superficially the hardware side of the Instructor 50 looks rather like many of the small evaluation systems, except that it comes as a small cabinet rather than a naked PC board. It has a hexadecimal data input keyboard and an eight digit 7-segment LED display, with a separate 12-key pad for feeding in commands to the monitor program.

Like some of the evaluation systems it has an inbuilt cassette tape interface, which will operate with any normal

audio cassette recorder. However unlike the majority of evaluation systems it also has full buffering and decoding for system expansion using the S-100 bus convention — a feature which will no doubt make it of interest to hob-

biests and small business users.

As you might expect, the Instructor 50 is based on the Signetics 2650 microprocessor. Along with the 2650 it has 512 bytes of RAM for user programs and a 2656 SMI (system memory interface) device which contains a 2K byte monitor program in ROM, together with 128 bytes of RAM for the monitor scratchpad.



*Neatly housed in a small desk-top case, the Instructor 50 system comes complete with three comprehensive training manuals.*

biests and small business users.

The monitor program built into the SMI is rather more powerful than is usually found in evaluation systems. Besides the usual facilities for entering program instructions and data, examining memory and processor registers, and running programs, it offers a number of features which make the

Instructor 50 easier and more straightforward to use. For example there is a "fast patch" data entry mode, which allows instruction and data bytes to be loaded into memory rather faster and more conveniently than the normal "display and alter" mode. There is also a single-step run mode, in which you can step through programs instruction by instruction, and a breakpoint facility which enables you to exit from a program at any desired point with the processor's status preserved so that you can analyse what had happened to that point.

The monitor commands concerned with the cassette interface are also more powerful than is usual. The "write cassette" command used to dump a program or data block to tape allows the block to be given a file identification number (from 00 to FF hex), while the "read cassette" command may be used to seek and load either a specified file, or the first file encountered. There is also an "adjust cassette" command, in which the Instructor 50 can be used to indicate the optimum playback level for the cassette tape machine.

In short, then, the Instructor 50 hardware seems to have been designed with particular emphasis on flexibility and convenience of use — making it

especially suitable for use as a training tool.

Of course what tends to make it of even more interest as a training system is the accompanying literature. This comprises three separate manuals, all about 215 x 275mm, and with a total of about 600 pages between them.

By far the thickest of the three manuals is the Users' Guide, which is a comprehensive guide to the system's hardware, software and operation. This manual gives an introduction to microcomputer basics, a description of system operation, an explanation of the control functions and monitor commands, full details of the 2650 instruction set, and a useful glossary of microcomputer terms. It also gives full circuit details, a full listing of the monitor program, and calling details for useful monitor sub-routines.

The second of the manuals is an introductory guide for those who need additional background in logic, binary numbers and basic computer operation. It goes into these subjects in considerable detail, yet in a straightforward and easily understood fashion.

The third book is a software applications manual. Along with a brief revision of Instructor 50 operation it gives eight demonstration programs designed to illustrate various aspects of microcomputer programming. Each program is described in depth, with an explanation of its operation and use together with a full listing.

The eight programs described in the applications manual are in fact those provided on the demonstration cassette which comes with the Instructor 50, so none of the programs has to be fed into the system by hand. The programs are titled "Electronic Billboard", "Desk Clock", "Stop Watch", "Crap Game", "Beat the Odds", "Slot Machine", "Train" and "Instructor 50 Music Theme".

After looking through the manuals and using the Instructor 50 for a while my impression is that both have been very carefully planned. They integrate together to form an attractive teaching package, which seems particularly well suited for providing people with a sound but easy to follow introduction to microcomputers.

At the quoted price of \$390 plus 15% sales tax the Instructor 50 costs a little more than typical evaluation systems, but still seems quite good value for money considering its potential as a training tool. I imagine schools, colleges and industrial organisations will find it of considerable interest.

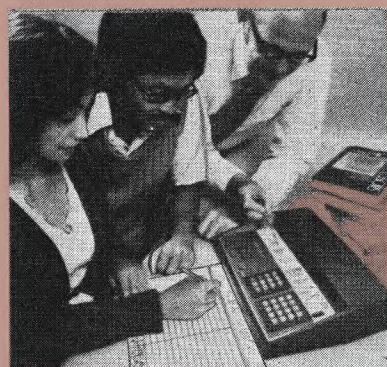
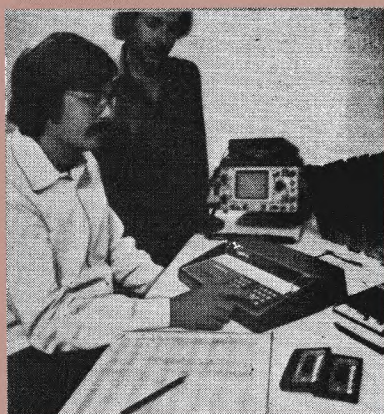
The Instructor 50 is available from Philips/Signetix stockists Cema Distributors, Soanar Electronics, Technico Electronics, Radio Parts, Fred Hoe & Sons (Brisbane), Applied Technology and Silicon Valley stores. It is also available from the Electronic Components and Materials division of Philips Industries, with offices in each state.





# Hands-on microprocessor experience from day one...

## The Instructor 50.



Learning by doing is still the best method of education. And when it comes to learning about the world of microcomputers, you won't find a better method than the Instructor 50.

It's the fast, ready-to-use learning device that immediately provides "hands on" experience for gaining microprocessor knowledge - in your home, office, or in the classroom.

The Instructor 50 is a COMPLETE package - including a power supply, a LED prompting display, and both functional and hexadecimal keyboards. You also get S-100 compatibility for adding memory and other peripherals. This lets you expand the machine's capability - and your microprocessing applications knowledge. Moreover, you can easily build a program library by recording your own audio cassettes.

The Instructor 50 comes complete with a Users' Guide, along with step-by-step

*\*one cassette supplied with each Instructor 50*

instructions for those with no previous microprocessor experience.

Signetics offers one of the broadest choices of microprocessors in the industry. This knowledge stands behind the Instructor 50. When you need to learn about microprocessors, start with Signetics. Start with Instructor 50.

**We can help you  
understand microprocessors.**

# signetics

a subsidiary of U.S. Philips Corporation

For more information write to:-  
Philips Electronics Components and Materials  
P.O. Box 50, LANE COVE N.S.W. 2066.

# PHILIPS

**We want you to have the best**

McCANN 153.0257



# LED display for your 2650

by **DAVID EDWARDS\*** \*69 Anglo Road, Campsie, NSW 2194

A four digit common-cathode LED display is required, and several options

The completed display unit can be

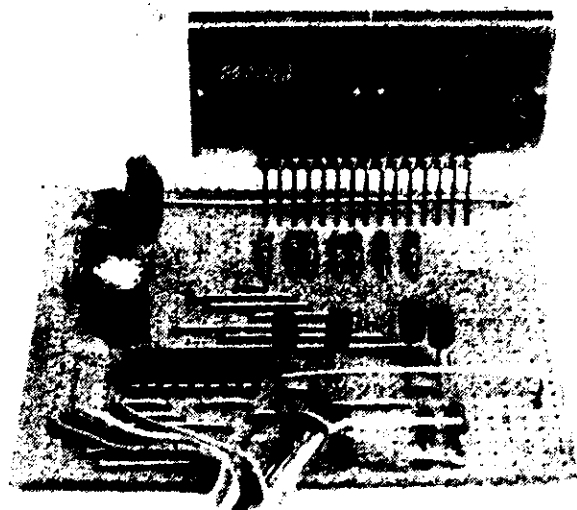
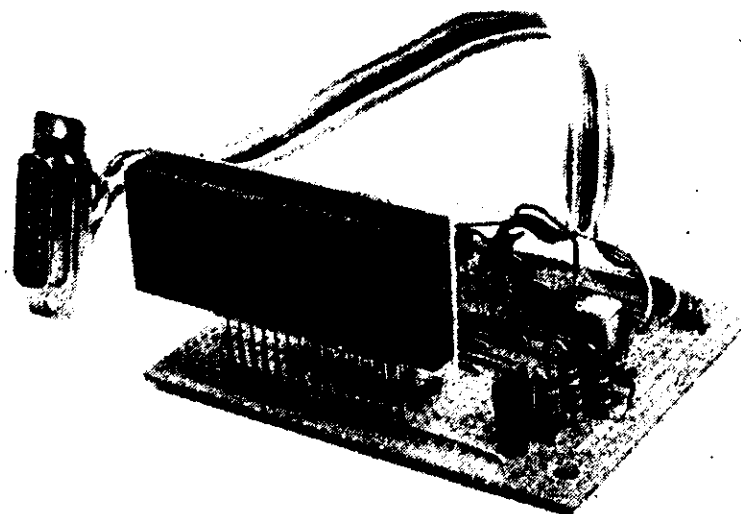
```
*G1E00 440 459
0440 76 40      PPSU 40
0442 75 18      CP5L 18
0444 04 82      LODI,R0 82
0446 F0         WRTD,R0
0447 F9 7E      BDRR,R1 0447
0449 04 46      LODI,R0 46
044B F0         WRTD,R0
044C F9 7E      BDRR,R1 044C
044E 04 25      LODI,R0 25
0450 F0         WRTD,R0
0451 F9 7E      BDRR,R1 0451
0453 04 10      LODI,R0 10
0455 F0         WRTD,R0
0456 F9 7E      BDRR,R1 0456
0458 1B 6A      BCTR,UN 0444
```

**FOUR DIGIT LED DISPLAY**

**VIEWED FROM ABOVE**

**VIEWED FROM BELOW**

*Fig. 1: The circuit for the author's software-driven four digit LED display. It interfaces to the 2650 Mini Computer via the non-extended "D" output port.*



```

0440 76 40 75 18 3F 02 DB 77 08 75 21 08 0C 82 94 C2
0450 E6 60 1A 20 06 00 75 20 0A 67 81 94 C1 E5 24 1A
0460 19 05 00 C0 77 10 05 51 06 1D 3B 1D F9 7C FA 7A
0470 75 10 1B 53 04 05 F8 7E C0 C0 1B 68 50 50 50 50
0480 44 0F 17 F0 04 DC F8 7E 17 75 18 01 3B 6E 64 80
0490 3B 71 01 44 0F 64 40 3B 6A 02 3B 60 64 20 3B 63
04A0 02 44 0F 64 10 3B 5C 77 10 C0 17

```

The photographs above show two views of the prototype display built by the author on a small piece of Veroboard. The wiring is not critical.

Fig. 3 (left): A HEX LISTING OF THE TIME program, which turns the 2650 and display into a 24-hour clock.

tested before connecting it to the computer. Connect +5V to the board, and observe the display. No numerals should be visible. If any are, switch off, and check the wiring associated with the four transistors. Assuming all is well, use a clip lead to connect the number 4 input bit to +5V. The right-most digit should now read "0", with all other digits off.

By applying +5V to inputs 5, 6 and 7 in turn, you should be able to make the digits read zero in turn. If you want to, you can apply BCD codes to the inputs of the 4511 by pulling the appropriate pins high, and check that the appropriate digits are displayed. However, if you were like the author, you will want to see the computer operate the display, and will not bother to carry out this test.

Fig. 2 is a listing of a small program which will exercise the display. It is completely relocatable, and can be stored anywhere in memory. The first address is the starting address. The program assumes that the display unit is connected to the D output port.

The program repeatedly writes four data bytes to the display, with a small delay between each successive write. The first data byte is X'82, and this displays the numeral 2 in the left most digit of the display. The nybble "8" (binary 1000) turns on this digit, while the nybble "2" is decoded by the 4511 to produce the seven-segment code for the numeral 2.

Similarly, the second data byte (X'46) displays a 6 in the 2nd digit from the left, and so on. The delay between each WRTD instruction, produced by the BDRR, R1 instruction, is necessary in order to provide a glitch free display. Without this delay, all segments of the display tend to glow, due to inherent

circuit delays caused by stray capacitance:

You can change the number displayed by altering the lower four bits of locations X'445, 448, 44F and 454. To turn a selected digit completely off, use a non-BCD number such as X' A to F.

The second program presented here is shown as a hex listing in Fig. 3. It occupies locations X'440 to 4AA inclusive, and is again completely relocatable. It is called TIME, and will make the computer and display unit appear to be a 24 hour clock. It uses the Pipbug routine GNUM to get an initial starting time from the line buffer.

To call the program, type G440 AAB8.

where AA is the current time in hours (e.g. 20 if it is 8PM), and BB is the current number of minutes past the hour. Do not press the carriage return key until the current minute has ended; the time displayed will then be correct to the nearest second (provided you press the cr key precisely at the 60 second time).

The program assumes that the CPU oscillator is running at exactly 1MHz. Changing the contents of location X'467 by one will vary the timing by approximately one part in 10,000. If location X'467 is incremented, the clock will slow down. To return to Pipbug, press the reset switch. ②

# An improved 2650 disassembler

Here is an improved disassembler program for small 2650 microcomputer systems, designed to complement the mini line assembler. It will translate all 2650 machine code back into mnemonic form, calculating operand addresses as it goes — making it ideal for program troubleshooting. With a minor change and the addition of a small routine it can also be used to prepare fully commented "source" listings.

by JAMIESON ROWE

An assembler program can be a very handy piece of software when you've written a program and want to feed it into your computer. But when your program is in the machine and won't run properly or doesn't do what you expected (one of these is usually the case!), the assembler won't help you much. Far more useful when you've reached this stage is a disassembler program, which as the name suggests does just the opposite of an assembler: translate from machine code back into human-readable mnemonic language.

On the surface, a disassembler mightn't sound as if it would be of much help when you're trying to track down those elusive program bugs. After all, in translating back to mnemonic language it merely gets you back to where you started! This might be so in theory, perhaps, but in practice things generally aren't that simple.

What tends to happen is that after feeding your program into the system, either via an assembler or directly in code you have assembled yourself, you try running it and then discover the first batch of bugs. Generally these are silly mistakes, which you correct as you find them by patching in small corrections — changing the condition criterion for a branch, adding in missing instructions, and so on.

Unless you have a major change to make, the tendency is to code the patches yourself, as this is faster than loading in and firing up an assembler. But in doing so, you tend to make coding errors which themselves produce new bugs. In any case it is all too easy to forget to change displacements in nearby instructions which, while not directly involved in a patch, may be affected by it.

The end result is that after you have made a certain number of patches, the program has become rather different from the way it was when you started.

This can make it quite difficult when it comes to tracking down the more subtle logical bugs, which are generally still in the program waiting to be discovered. At this stage of the proceedings it can be a big help if you can use a disassembler to provide an accurate mnemonic listing of the program as it now stands.

Another important area of use for a disassembler is when you acquire a program in "naked" machine language form, without any accompanying source listing or other descriptive literature. It may run on your system, but you want to see how it works in order to make sure that you use it properly. Or it won't run on your system, perhaps because it was written for a slightly different system and you want to work out how to modify it so that it will work on yours. Or you may want to see how to provide it with additional features, or how to adapt it to perform a similar but different job...

The listing produced by a dis-

assembler won't give you all of the information in a good source listing, but if you can't get hold of a source listing it's certainly the next best thing.

As those with 2650 microcomputer systems are probably aware, a small disassembler for 2650 code has been available for quite a while now. One of the pioneering software programs produced by the 2650 Users' Group, it was written by Ian Binnie. A modified version of this program prepared by the present author was made available to EA readers on our 1978 Software Record.

Helpful though this early disassembler has been, it did have a number of disadvantages. One problem was that it didn't disassemble quite a few of the single-byte and double-byte instructions; another was that it made errors in calculating the absolute address referenced by forward referencing relative indirect addressing instructions.

A further problem was that it didn't fully disassemble absolute addressing instructions, and gave no indication of indirect or indexed addressing modes.

Taken individually, none of these shortcomings was all that serious. But collectively they have tended to limit the disassembler's value.

While I was working on the 2650 mini line assembler, it occurred to me that it should be possible to write a more comprehensive disassembler which could make use of the assembler's mnemonic lookup table. So as soon as the assembler was completed, I set about writing a new disassembler along these lines. It took a while to write and debug, but finally here it is!

As the foregoing suggests, the new disassembler is meant to be used in conjunction with the mini assembler. This is because it shares the same mnemonic lookup table, located from X'17CF—1A01. It also uses the same RAM buffer area (X'1A02—1A54) for its own line buffer and scratchpad area, to save memory space. Needless to say this doesn't mean that you can't use the disassembler by itself — all you need to do in order to do this is load it in together with the lookup table.

The disassembler itself is 726 bytes long, occupying memory from X'1B00 to 1DD5 — so that it fits into memory immediately above the assembler.

\*G1B02 1B9E 1BBD 30

2650 DISASSEMBLER VERSION 2

1B9E 1F1BED	BCTA,UN	1BED
1BA1 3F1D4E	BSTA,UN	1D4E
1BA4 0604	LDDI,R2	04
1BA6 07FF	LDDI,R3	FF
1BA8 0FBA42	LDDA,R3	*1A42+
1BAE CF7A11	STRA,R3	1A11#
1BAE FA78	BDRR,N	1BA8
1BB0 0705	LDDI,R3	05
1BB2 0FFA42	LDDA,R3	*1A42#
1BB5 C2	STRZ,R2	
1BB6 C3	STRZ,R3	
1BB7 460F	ANDI,R2	0F
1BB9 47F0	ANDI,R3	F0
1BBB E60C	COMI,R2	0C
1BBD 1C1CC9	BCTA,Z	1CC9

Fig. 1: A sample of the disassembler's output. Its calling format is also shown.

The starting address is 1B00, and you call it by typing an extended PIPBUG "GO" command. In other words you type an input GO command which includes information for the disassembler, telling it the memory range you want it to work on, and whether or not you want it to split the output listing into pages of a certain size, with headings.

The precise calling format required is similar to that for the earlier disassembler:

G1B00sAAAA sBBBB sCCr

where "s" stands for a space and "r" stands for a carriage return. AAAA is the start address (in hex) of the section of program you want disassembled, while BBBB is the end address. These are the only two essential parameters required, and of course the end address should be greater (higher) than the start address — otherwise the disassembler will throw you out with a peremptory "??".

The third parameter "CC" is an optional one used to specify the number of lines per page, where a long listing is to be produced. CC is a two-digit hex number, so you can specify pages of up to 255 lines each. Each page will be given the title "2650 DISASSEMBLER VERSION 2", and at the end of each page the disassembler will pause to allow you to advance the paper in a printer, etc. You can then prompt it to continue with the next page by hitting any key on the terminal keyboard.

If you omit the third parameter, or give it a value of zero (00), the disassembler assumes that you don't want pagination. Accordingly it will omit the titles, and simply provide an unadorned continuous listing. This mode of operation is very suitable for quick disassembly of numerous short instruction sequences, when you are troubleshooting — you don't have to worry about typing in the third parameter, and operation is faster and more efficient because the disassembler doesn't have to provide a title each time.

A hex listing of the disassembler is given in Fig. 2. This is complete part from the mnemonic lookup table given as part of the assembler.

In operation, the disassembler will provide mnemonic translations of all bytes it finds in the designated memory address range, providing they represent valid 2650 instruction codes. Bytes which are not valid 2650 codes will be printed at the start of the appropriate line, but will not be translated. Naturally enough the disassembler has no way of knowing whether the memory range you specify contains a program, or data — it is up to you to look after that.

If you do make a mistake and set it loose on some data, don't worry. Nothing will be damaged. All that will happen is that the disassembler will try valiantly to make some sense out of the data, translating it into whatever

```

1B00 76 60 77 02 3F 02 DB 3F 1C 1C 3B F9 CD 1A 4B CE
1B10 1A 49 E9 AB 19 07 1E 02 50 EA AA 99 FA 3B E6 CE
1B20 1A 40 08 FC CC 1A 41 18 08 05 1D 06 9E 3F 1D 8A
1B30 C0 07 FF 04 20 CF 3A 02 E7 22 98 79 07 FF 0C 1A
1B40 46 3F 1D 30 0C 1A 47 3B F9 87 01 0C 9A 46 3B F2
1B50 75 09 77 02 07 00 06 12 EF 7B 6B 18 06 87 02 FA
1B60 77 1B 2C 05 19 0F 3B 6B C2 1B 36 74 2A 75 30 76
1B70 36 77 3C C0 72 40 78 BB A8 9B AE B4 B4 B5 BA 92
1B80 C0 93 C6 12 CC 13 D2 BF D8 9F DE 10 EA 11 F0 05
1B90 18 06 28 44 FC 3F 1D 55 E7 00 98 08 07 07 1F 1B
1BA0 ED 3F 1D 4E 06 04 07 FF 0F BA 42 CF 7A 11 FA 78
1BB0 07 05 0F FA 42 C2 C3 46 0F 47 F0 E6 0C 1C 1C C9
1BC0 E6 08 1C 1C 9B E6 04 1C 1C 65 E6 02 1C 1C 48 E6
1BD0 01 1C 1C 25 3F 1D 0D 3F 1C EF 3F 1D 70 C2 20 F6
1BE0 80 98 02 04 1F 07 18 3F 1D 30 02 3B FB 20 CF 3A
1BF0 02 05 1A 06 02 3F 1D 8A 3F 1C FB ED 1A 48 1D 00
1C00 22 1A 05 EE 1A 49 19 F7 75 09 04 FF 8C 1A 41 C8
1C10 FC E4 00 9C 1B 31 3F 02 86 1F 1B 22 CD 1A 46 CE
1C20 1A 47 17 C0 C0 F7 10 18 1A 0C 9A 46 C1 44 03 45
1C30 DC 25 14 98 04 06 EC 1B 02 06 D4 05 17 3F 1D 55
1C40 3F 1D 78 07 16 1F 1B ED F7 10 18 0D 0C 9A 46 44
1C50 03 05 17 06 D4 3B E7 3B E8 3F 1D 0D 07 18 3F 1D
1C60 30 07 1A 1B E1 0C 9A 46 C1 44 03 F5 10 18 04 06
1C70 D4 1B 02 06 EC 05 17 3F 1D 55 3F 1D 78 3B DB 3F
1C80 1C EF 3F 1D 70 C1 77 09 8C 1A 47 C2 3F 1D BD C1
1C90 3F 1D 23 02 3B C9 07 1C 1F 1B ED 0C 9A 46 44 03
1CA0 05 17 06 D4 3B D2 3B D3 3F 1D 0D 3F 1C EF 50 50
1CB0 50 50 50 44 03 18 07 C3 0F 77 CF CC 1A 1F 3F 1D
1CC0 23 3F 1D 17 07 1D 1F 1B ED 0C 9A 46 F4 40 18 04
1CD0 06 EC 1B 02 06 D4 05 17 44 03 3F 1D 55 3F 1D 78
1CE0 3C 2B 3B 0B 44 7F 3F 1D 2B 3B 2C 07 1C 1B D8 F5
1CF0 80 16 04 2A 07 17 CF 3A 02 01 17 77 0A 75 01 0D
1D00 1A 46 0E 1A 47 86 01 85 00 3F 1C 1C 17 3B 6C 0C
1D10 9A 46 07 06 3B 1A 17 3B 62 0C 9A 46 3B 12 07 08
1D20 3B 0E 17 45 1F 0C 1A 46 44 60 61 07 18 3B 01 17
1D30 C1 75 0A 50 50 50 50 3B 05 01 3B 02 01 17 44 0F
1D40 E4 0A 1A 04 84 37 1B 02 84 30 CF 3A 02 17 CD 1A
1D50 42 CE 1A 43 17 3B 77 07 04 EF FA 42 14 75 01 77
1D60 08 86 06 85 00 E5 19 98 6C E6 F6 1A 68 07 00 17
1D70 44 7F F4 40 16 64 80 17 04 2C CC 1A 15 06 02 07
1D80 FF 0F BA 42 CF 7A 16 FA 78 17 CD 1A 44 CE 1A 45
1D90 75 08 07 FF 0F BA 44 1C 00 8A BB A0 1B 76 0D 0A
1DA0 32 36 35 30 20 44 49 53 41 53 32 45 4D 4C 45
1DB0 52 20 56 45 52 53 49 4F 4E 20 32 0A 00 B5 01 98
1DC0 07 01 1A 0B 04 01 1B 08 01 9A 04 04 FF 1B 01 20
1DD0 75 09 8C 1A 46 17

```

Fig. 2: A full hex listing of the new disassembler, less its lookup table.

pseudo-program it may represent. The odds are that some of the data numbers won't even correspond to valid 2650 opcodes, so the listing will probably have a fair number of blanks in the mnemonic columns.

The main thing to note is that if you do force the disassembler to struggle through some data and then into some valid program coding in the one run, it may well be thrown out of kilter for the first few valid instructions after the data. This is because at the end of the data section it may be part-way through the disassembly of a "fake" multi-byte instruction, causing it to regard the first byte or two of the real instruction as the rest of the fake instruction.

If this happens the first few instructions after the data will be wrongly disassembled, until the coding forces the disassembler back into correct "phase" with respect to the start and finish of each instruction.

Incidentally the same sort of malfunction can occur if you have made a mistake in the coding being disassembled, so that the opcode of an instruction has accidentally been changed into that for an instruction of

different length. This will again throw the disassembler out of kilter, because it will be misled into regarding opcode bytes as operand bytes and vice-versa.

There is also a third way the disassembler can be led astray: by giving it the wrong memory range starting address, when you call it. Needless to say if you tell it to start in the middle of the first instruction rather than the start, it has no way of knowing. It will simply press on, translating away as best it can.

Incidentally, the fact that the disassembler can be led astray in these ways does not mean that it is faulty. There is no way in which any disassembler can tell if the numbers it is processing are instructions, or data — after all, the only difference between an instruction byte and a data byte is the way the computer is told to interpret them. Similarly where a disassembler has to deal with variable length instructions, there is no way it can infallibly identify opcode bytes and distinguish them from operand bytes — they're all just numbers.

In other words, make sure that you start the disassembler off on the right foot when you call it. And if it should



# AN IMPROVED 2650 DISASSEMBLER

become misled by some data you've forgotten to tell it to bypass, or by an opcode you have accidentally changed into one for an instruction of a different length, put the blame where it really lies. After all, it's only a dumb program — you're supposed to be the intelligent one!

After it has finished disassembly of the designated memory range, the disassembler will return to PIPBUG as usual. Or, to be more accurate, it will return to PIPBUG when it finishes disassembly of the last instruction which starts in the designated range. This means that when you specify the end of the range to be disassembled, you don't have to work out the very last byte of the last instruction in the range. Just specify the address of any byte in the last instruction to be disassembled

the disassembler uses the same format as the line assembler, showing the index register in the R/C field immediately after the opcode mnemonic. The other point is that for convenience the disassembler places its indexing symbols AFTER the operand address, not before it.

A third point to note is that the R/C mnemonic produced for the BDRR instruction is a condition code mnemonic (N) rather than the more usual register code mnemonic. This is a minor shortcoming of the disassembler, due to a programming compromise. It also occurs when BIRR instructions are disassembled.

Apart from these three minor differences, the listing produced by the disassembler follows the standard 2650 instruction format.

As you can see it is quite a short routine, which fits into memory immediately after the disassembler itself. To patch it into the disassembler, all you need to do is change the instruction beginning at address 1D97 from 1C008A into 1C1DD6.

What the routine does is cause the disassembler to pause after it has listed each disassembled instruction. You can then type in any comment you wish from the terminal keyboard.

If you end the comment by typing a carriage return, the routine will return to the disassembler via the CRLF subroutine and the next instruction will be disassembled after the usual carriage return and line feed. However if you end by typing "TAB" instead of carriage return, the routine will remain in comment mode and will provide a carriage return, line feed and 15 spaces. This lets you feed in a full line comment, of the type used to label routines, etc. The three comment lines at the top of Fig. 3 itself were added in this way.

You can provide line spaces between parts of your listing by using the TAB key, or using the LF key.

Together the disassembler and supplementary routine provide a very convenient means of making fully commented source listing. As well as using them to produce the listing shown in Fig. 3, I have already used them to produce a full source listing of the dis-

\*ROUTINE TO PROVIDE COMMENT ADDITION  
\*FACILITY FOR THE IMPROVED 2650  
\*DISASSEMBLER. J. ROWE 1/4/1979

1DD6 3F0286	BSTA,UN	0286	ACCEPT CHAR VIA CHIN SR
1DD9 E40D	COMI,R0	0D	TEST FOR CR
1DDB 1C008A	BCTA,Z	008A	EXIT VIA CRLF IF FOUND
1DDE E409	COMI,R0	09	TEST FOR TAB (HT)
1DE0 1B04	BCTR,Z	1DE6	GO SET UP IF FOUND
1DE2 BBA0	ZBSR	*0020	NOT CR OR TAB: ECHO VIA COUT
1DE4 1B70	BCTR,UN	1DD6	& LOOP BACK
1DE6 BBA5	ZBSR	*0025	TAB: GIVE CRLF
1DE8 070F	LODI,R3	0F	SET R3 AS COUNTER
1DEA 3F0361	BSTA,UN	0361	& USE AGAP SUBR FOR 15 SPACES
1DED 1B67	BCTR,UN	1DD6	THEN LOOP BACK FOR COMMENT

Fig. 3 (above): An optional add-on routine which lets you add comments to the listing.

Fig. 4 (right): A further routine, called separately, which will print out ASCII message strings stored in memory.

— the disassembler will automatically finish the instruction before it bows out.

This can save valuable time, because often you're working from an earlier listing for reference, and it's convenient to give the end of the range as the address of the first byte in the last instruction.

Like the assembler, the new disassembler uses a number of utility sub-routines from PIPBUG. In this case it uses GNUM to fetch its input parameters, CHIN and COUT to communicate via the terminal, and CRLF to provide carriage return/line feeds.

As you can see from the sample listing in Fig. 1 (which is actually part of the disassembler itself), the basic listing produced by the disassembler is 30 characters wide. This makes it suitable for all normal terminals and printers.

Note two things about the disassembler's listing, as illustrated in Fig. 1. One is that for indexed instructions

## 2650 DISASSEMBLER VERSION 2

\*ROUTINE TO PRINT OUT ASCII MESSAGES  
\*STORED IN MEMORY. J. ROWE APRIL 1979  
\*USES MESSAGE PRINTING SUBR IN MY  
\*IMPROVED DISASSEMBLER, ALSO GNUM IN  
\*PIPBUG. CALL BY GIDF0 AAAA, WHERE  
\*AAAA IS START OF MESSAGE. NOTE THAT  
\*MESSAGE MUST END WITH A NULL

1DF0 7660	PPSU	60	SET FLAG FOR MARK, INHIBIT INT.
1DF2 3F02DB	BSTA,UN	02DB	FETCH MESSAGE START
1DF5 3F1D8A	BSTA,UN	1D8A	& GO PRINT
1DF8 9B22	ZBRR	0022	THEN RETURN TO PIPBUG

You may also have noticed from Fig. 1 that the basic listing produced by the disassembler is not all that much different from a full "source" listing — the only thing lacking is the comments. This suggests that the disassembler could be used to produce source listings of any program stored in your system's memory, merely by modifying it so that you can add comments.

In fact I have produced a supplementary routine which can be added to the basic disassembler to let you do just that. The supplementary routine is shown in Fig. 3 — as a full listing produced when it was working with the disassembler, so you can see the type of listing it lets you produce.

assembler itself.

For those who would like to analyse the disassembler's operation in detail, copies of the full listing are available from our Information Service for a fee of \$4.00, to cover photocopying and postage.

Finally, there's one thing the disassembler won't do: print out ASCII message strings in memory, so you can see what they say. But there's an easy way to get around this — use a separate little routine which makes use of the disassembler's message printing subroutine. The routine you need is shown in Fig. 4 above. It occupies only 10 bytes, fitting in above the comment addition routine; you call it as shown. ②

# Adapter PCB for 300-baud PIPBUG mod

Here is an item which should be of special interest to those with 2650-based microcomputer systems using the PIPBUG monitor. It is a small adapter board which lets PIPBUG operate at either 110 or 300 baud, without the need to cut or patch the main CPU board.

by **ANTHONY HAGEN**

11 Stewart Street, Hawthorne Qld 4171.

Like many other readers I built the 2650 Mini Computer of May 1978. After using it for a while, I felt the need to have PIPBUG run at 300 baud rather than 110 baud, in order to dump and load more rapidly. However I hadn't worked out how to do this before Mr R. W. Brown's solution was published in the February 1979 issue, in the "Circuit and Design Ideas" column. Mr Brown's idea was such a good one that I resolv-

ed to put it into practice, but I didn't like the idea of cutting the tracks on the main CPU board.


To avoid having to do this, I designed a small adapter PCB which uses the same basic circuit as Mr Brown's, but with a few pin connections changed. The idea is that the adapter PCB connects via a cable and 24-pin DIL plug to the main PCB, plugging into the original socket used for the PIPBUG ROM; the ROM then plugs into a similar socket on the adapter PCB.

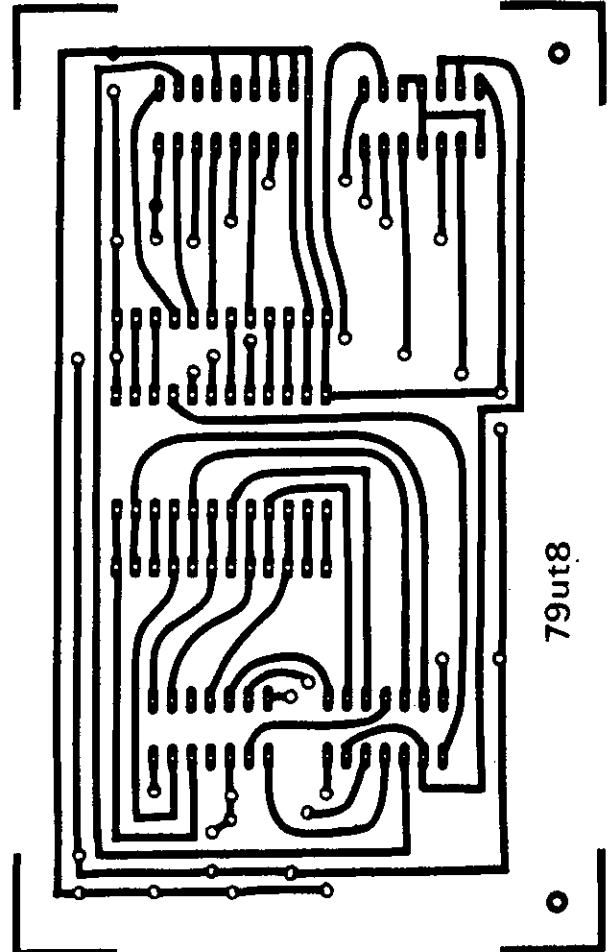
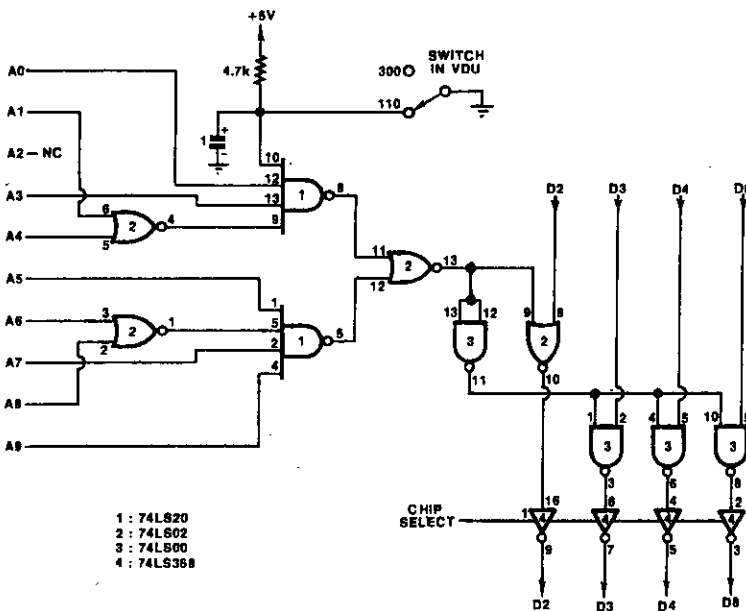
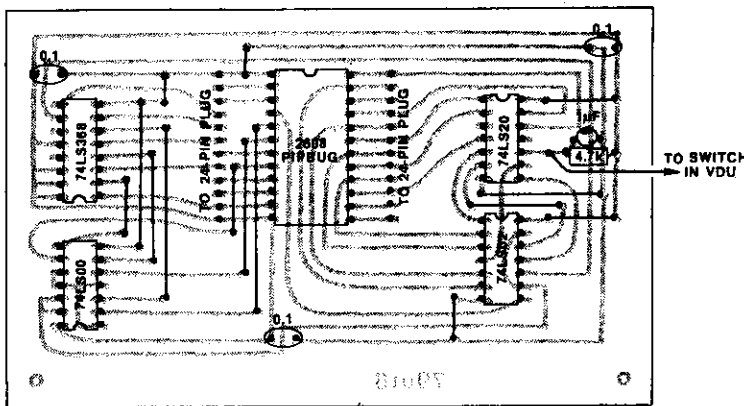
Details of the adapter PCB are shown

below, along with the slightly modified circuit. I mounted the PCB between the main PCB and the end of the case, on two small brackets. Apart from the 24-way cable back to the original PIPBUG ROM socket, the new PCB has only one other connection: a wire to a 110/300 baud control switch.

For convenience I extended this control wire to a spare set of contacts on the baud selector switch of my Low Cost VDU, by using a spare pin of the DIN connectors.

The earthy side of the 20mA serial output circuit serves as the return. This makes the VDU switch control both the VDU and the CPU baud rates together.

My adapter worked immediately, and surpassed all expectations. I hope other readers will be able to use my adapter layout with equal benefit. 



Here are full details of the adapter board: the circuit, the full-size PCB pattern and the board wiring diagram.

# Cassette Interface runs Kansas City, 1200 baud

Microcomputer users looking for a flexible, high performance cassette interface should find the E & M Electronics CI-1 of special interest. It will operate at the "Kansas City Standard" frequencies for compatibility with other systems, but also allows storage and retrieval of data at up to 1200 baud.

by JAMIESON ROWE

The easiest way of storing and retrieving both data and programs with small microcomputer systems is to use magnetic tape, usually in compact cassette form. Ordinary audio cassette recorders can be used for the job, providing a suitable interface is used to convert between the computer's logic levels and the audio frequencies handled by the recorder.

To date most cassette tape interfaces used with hobby computers have used the frequency-shift keying (FSK) technique, and in particular the "Kansas City Standard" method whereby a digital 1 is recorded as a tone of 2400Hz or 4800Hz and a digital 0 as a tone of 1200Hz or 2400Hz. The higher frequency in each case is for data at 300 baud, while the lower frequencies are for data at 110 baud.

While capable of quite reliable operation at these lower data transfer rates, the Kansas City Standard is not really suitable for higher rates. The problem is that as soon as your computer grows beyond a modest size, and you develop some useful programs, a data rate of 300 baud becomes irritatingly slow. Programs and data seem to take ages to get into and out of memory, and you long to be able to dump and load at a higher rate — say 1200 baud, which is available on some of the new packaged personal computers.

Unfortunately until now, if you have used an interface designed to work at 1200 baud or some other high data rate, you have tended to lose the compatibility of the Kansas City Standard. This can create problems, because hobbyists often want to exchange data and programs with each other.

The CI-1 cassette interface from E & M Electronics has been designed to get around these problems. It is a dual-mode interface, able to work in either the Kansas City format or in a high-speed mode capable of handling data at up to 1200 baud. At his rate the highest audio frequency recorded on the tape is 8kHz, which should be within the bandwidth of most cassette

recorders.

So using the CI-1 allows you to dump and load most of the time at up to 1200 baud for faster system operation, while still allowing you to generate and handle material conforming to the Kansas City Standard, when required.

The CI-1 is based on a phase-locked loop (PLL) encoding and decoding system, to provide tolerance to tape recorder speed fluctuations. The PLL and filter circuitry time constants are switched to change between the Kansas City and high speed modes of operation.

To make the interface compatible with just about any system, its computer and terminal ports can be wired

*The CI-1 interface as assembled from a kit. As supplied it doesn't include a case or power supply, just the basic PCB assembly.*



for either 20mA current loop or TTL logic level interfacing.

Other features of the interface include a "data present" LED, to make it easier to find the start of data records, and CMOS switching to simplify the mode wiring. The interface runs from a single 5V supply, drawing only 90ma maximum with current-loop interfacing (50ma with TTL interfacing).

The CI-1 interface is available as both an assembled and tested unit, ready for operation, and as a do-it-yourself kit. This is a little unusual, because PLL interfaces are usually a little critical when it comes to some of the key components. To get around any possible problems, E & M Electronics select and match the critical components for each kit, and supply them as a carefully identified set. They also offer a back-up ser-

vice, in case the kit builder should get into trouble.

Ed Monsour, the engineer behind E & M Electronics, sent me a sample CI-1 kit with the idea that I could find out at first hand how it goes together and performs.

Although the instructions supplied with the kit are a little brief, they are quite clear and I found no real difficulty in putting it together. The only trouble was a very minor one: connections to the PCB are via pins and push-on connectors, and PCB drilling tolerances made the connectors a little hard to push on properly. But they responded with a little care and perseverance.

My only other minor gripe is that the switches supplied with the kit have very short toggle levers. Presumably this is to prevent inadvertent operation, but some users like myself may prefer to have a longer toggle at least on the record/play switch. Still, this is easily fixed.

After following the setting-up procedure given in the instructions, the

completed CI-1 worked very well. In Kansas City mode it made recordings which were fully compatible with my existing interface, and vice-versa. And in the high-speed mode it performed as series of dumps and reloads at 1200 baud without an error.

In short, then, I found the CI-1 interface an excellent performer, and can recommend it to anyone seeking an interface which offers high-speed operation with a compatibility option.

The quoted prices of \$69.00 for the assembled unit and \$39.00 for the kit (both plus 15% sales tax if applicable) also seem very reasonable.

Further information on the CI-1 cassette interface is available from E & M Electronics Pty Ltd, 136 Marrickville Road, Marrickville, NSW 2204. Telephone (02) 51 5880.

# Using the PIPLA/PIPBUG2 ROM in your 2650 system

After some delay, Signetics has released the 2656/CP1002 ROM device containing its "PIPLA" line assembler program, together with an improved version of the PIPBUG monitor. Here are details on how the device can be used with our 2650 Mini Computer system.

by JAMIESON ROWE

If you built up our popular 2650 Mini Computer system, you'll probably be aware that there was provision for a mysterious 40-pin IC, on the expansion board described in the November 1978 issue. This was explained at the time as simply "a possible future addition", and until now we haven't been able to clarify the situation any further.

Actually I did give a clue to the identity of the mysterious device in the April 1979 issue, in the article describing a simple line assembler program for 2650 systems. As you may recall, I mentioned that the assembler was based on PIPLA, a program developed by Signetics in the USA to go into a "special ROM device" along with a modified and enhanced version of PIPBUG.

But now the full story can be told. The mysterious device in question is the CP1002, a custom-programmed version of Signetics' 2656 "system memory interface" (SMI) device, and it is finally available.

I first learned of the CP1002 back in April 1978, during a visit to the Signetics facility in California. In fact during the visit, the Signetics people very kindly gave me a pre-production sample of the device, in the expectation that it would be going into production shortly.

Shortly after my return, David Edwards and I were planning the expansion board for the 2650 Mini Computer, and in view of the likely release of the CP1002 we decided to allow space for it on the board. However after this was done we were advised by Philips that Signetics had struck unexpected trouble with the device, and its release would be delayed. By this stage it was too late to modify the PCB pattern, so we were forced to gloss over the matter.

Apparently Signetics struck more trouble than they expected, because as the months wore on the CP1002 still failed to appear. This was one of the

reasons that I finally decided to describe a modified version of the PIPLA line assembler, in the April 1979 issue.

Well, the problems must finally have been solved, as the CP1002 is here at last. So without further ado let's see what it contains, and how you can hook it into your 2650 Mini Computer.

As mentioned above, the CP1002 is actually a custom-programmed version of the Signetics 2656 SMI device. This is a mask programmed N-channel MOS LSI device, in a 40-pin package, and containing 2K bytes of ROM, a 128-byte static RAM, a clock oscillator, an 8-bit latch and 8 multi-purpose pins which may be programmed to serve as either I/O lines or memory block chip enable outputs.

In the case of the CP1002 version, the 2K ROM contains two useful programs. One is PIPBUG2, a modified and enhanced version of the familiar monitor program used in most small 2650 systems; the other is PIPLA, a small line assembler.

PIPBUG2 is similar to the original PIPBUG, but it offers some additional features. One is that it will operate at either 110 or 300 baud, as far as communication with the terminal is concerned. It is automatically synchronised to whichever of these rates is required, simply by sending in a "U" from the terminal keyboard after the CPU has been reset.

Another feature offered by PIPBUG2 is that it is capable of dumping a program in the binary format needed to program PROMs on a Data I/O PROM programmer. And there is a third feature: the ability to perform hexadecimal addition.

The only drawback of PIPBUG2 is that Signetics have made it quite different from the original PIPBUG in terms of subroutine calling addresses, etc. So if you have a swag of programs which make use of the subroutines in original PIPBUG, you'll have to modify them for

use with PIPBUG2. It isn't just a matter of changing the subroutine calls, either — some of the subroutines use different registers, and different parameters.

As for PIPLA, the line assembler, this is very similar to the line assembler I described in the April 1979 issue. The only differences are as follows:

1. PIPLA gives no initial identifying message.
2. PIPLA assumes an initial origin at 0C00, rather than 0440.
3. PIPLA has no facility to accept the DATA directive.
4. PIPLA does not strip the address of non-branching absolute address instruction to 13 bits, so that can make errors when assembling programs for pages other than page 0.

Of course a final difference is that PIPLA is meant to go with PIPBUG2. It uses subroutines from the latter, and thus is dependant upon it.

What it boils down to is this. The CP1002 provides you with PIPBUG2 and PIPLA, resident in ROM so that they're always ready to go. And together the two programs are a big improvement over the original PIPBUG, which you can consider them as replacing. But whether you'll want to replace your existing PIPBUG ROM with the CP1002 will probably depend upon how many programs you have that use the original PIPBUG subroutines. If you've got quite a lot, you may not find the idea too attractive.

For those who do want to use the CP1002, it can be connected into the 2650 Mini Computer quite simply. The details are shown in the diagrams. As you can see, the main thing is to add a 40-pin DIL socket to the previously unused space on the expansion board (78UP9). Most of the necessary connections are made by the PCB pattern, already. All you have to do to get the ROM section of the device in operation is to run a wire from pin 22 of the 2650, to supply the WRP signal to the CP1002's pin 17.

The CP1002 has its own internal memory block decoding, so that it automatically assumes the address range 000-87F. The ROM occupies the addresses 000-7FF, while the 128-byte RAM occupies 800-87F.

What this means is that to prevent bus conflict, no other memory devices can occupy the same memory range. As you won't need the original PIPBUG ROM any more, this will free the bottom 1K (address range 000-3FF). However you'll probably have to shift some of the RAMs out of the range from 400-7FF and 800-BFF, to higher blocks. Depending upon your system and the amount of RAM you have, this may be simply a matter of changing links at the output of the 74LS138 decoder on the CPU board.

As you can see, the CP1002 also provides for a crystal clock oscillator for the 2650 CPU. So if you haven't provided your 2650 with a crystal clock as yet, this can now be done by adding a 4.000MHz crystal, three resistors and a capacitor as shown. Space is already provided for these components, alongside the CP1002 socket on the expansion board.

The output of the clock oscillator appears at pin 10 of the CP1002, and is at 1MHz ready to connect directly to the clock input of the 2650 chip (pin 38). Needless to say you will have to remove the existing 74LS123 clock oscillator chip, to prevent it loading down the new clock signal. If you have used a socket for the 74LS123 this will simply be a matter of unplugging the IC from its socket. Otherwise you may have to unsolder the IC and remove it that way, although some may elect to simply cut the PCB trace connecting its output pin 5 to pin 38 of the 2650.

In the CP1002 version of the 2656, the eight "multi-purpose" pins are programmed as memory block select and extended I/O address enable outputs. Although these are unlikely to be

TABLE 1: 2656/CP1002 SMI enable outputs				
PIN	LABEL	FUNCTION	ADDRESS	M/I/O
34	X0	I/O enable	FF	0
35	X1	Mem. select	0C00-0CFF	1
36	X2	Mem. select	0D00-0DFF	1
37	X3	Mem. select	0E00-0EFF	1
9	X4	Mem. select	0F00-0FFF or 1F00-1FFF	1
8	X5	I/O enable	00-03	0
7	X6	I/O enable	04-07	0
6	X7	I/O enable	FF	0

of much use in the 2650 Mini Computer system, Table 1 shows the significance of eight signals. Note that four of them are memory chip enables for 256-byte blocks, while the other four are enable signals for extended I/O addresses.

When you have wired in the CP1002 and checked it out, you'll be ready to turn on the system and try it out. As with the original PIPBUG, PIPBUG2 starts at address 0000 and thus comes up immediately due to the power-up reset. But in this case it doesn't print its prompt asterisk (\*) immediately; instead it waits for you to key in a "U" from the terminal, in either 110 or 300 baud. This tells it which of the two rates you want, and it then locks onto that rate and sends out the prompt character to show that it's ready for business.

The commands for PIPBUG2 are the same as for its predecessor, except for the two extras. The format for the hex addition command is

H sp AAAA sp BBBB cr  
where H is the command character,

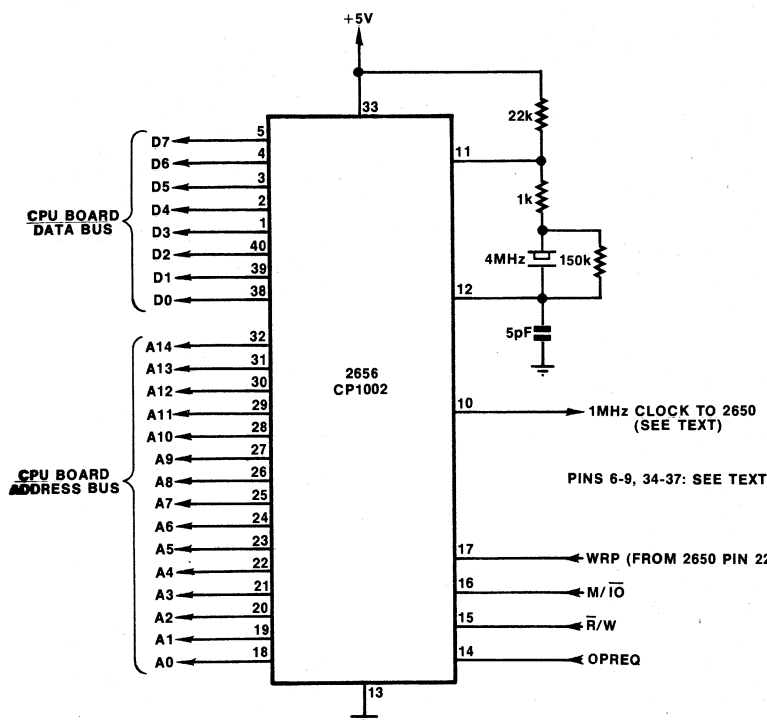
AAAA is one of the hex numbers to be added, BBBB is the second number, sp is a space and cr is a carriage return. Leading zeroes are not necessary when keying in the numbers.

Note that you can use this command to perform a hex subtraction by using it to tell you the 2's complement of the subtrahend first, then adding that to the diminuend. To get the 2's complement you first work out the 1's complement yourself, simply by complementing all bits individually. Then use the H command to add 1 to this figure, which will give you the 2's complement. Finally you then use the H command again to add this to the second number.

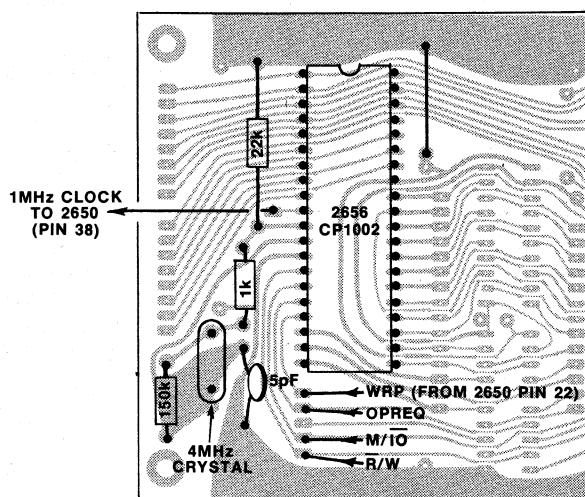
The format for the PROM programming dump command is

P sp A sp BBBB sp CCCC cr

where P is the command character, A is a parameter specifying the bits of each byte to be dumped, BBBB is the starting address in memory of the data to be dumped, and CCCC is the number of following words (i.e., one less than the word capacity of the PROM to be



2656-CP1002 PIPBUG 2/PIPLA ROM CONNECTIONS



ABOVE: The wiring required on the 2650 Mini Computer expansion board, in order to use the CP1002. The address, data and supply connections are provided already by the PCB.

LEFT: The schematic connections for the CP1002 ROM. No chip enable signal is required, as it contains its own address decoding. The crystal oscillator is optional.

## CP1002 "PIPLA" ROM

loaded). As before sp means a space, and cr means a carriage return.

The parameter A is used to specify the dumping format. There are three formats allowed; you can either dump all 8 bits of each memory byte, only the least significant 4 bits of each, or only the most significant 4 bits. The three modes correspond to the following values for parameter A:

0 — all 8 bits dumped

1 — only the least significant 4 bits

2 — only the most significant 4 bits

If options 1 or 2 are specified, the four bits of data are right justified and the upper four bits are dumped as zeroes.

The remaining command functions provided by PIPBUG2 are virtually identical to those of the original PIPBUG. Hence there is the "A" command to examine and alter memory, the "L" command to load from cassette or paper tape, the "D" command to dump to cassette or paper tape, the "S" command to see and set the registers, the "B" command to set a breakpoint, the "C" command to clear a breakpoint and the "G" command to transfer command to a user program. These are all used in exactly the same manner as those of the original PIPBUG.

As with the first version of PIPBUG, there are a number of utility sub-routines in PIPBUG2 which may be called by user programs. The most useful of these are described in Table 2. Note that as mentioned earlier, some of these sub-routines are significantly different from those in the original PIPBUG when it comes to use of registers, etc.

The PIPLA line assembler starts at hex 0400. As mentioned earlier it gives no initial identifying message and assumes a starting origin of 0C00 for the program to be assembled. So when you call it, the response is simply 0C00.

Apart from this, its operation is very similar to that of the modified assembler I described in the April 1979 issue. You can change the origin as desired with an ORG directive, store a string of ASCII characters with an ASCII directive, and return to PIPBUG2 with an END directive. The only directive not available is the DATA directive.

There is only one other point to remember. The input buffer used by PIPLA is only 24 characters long, compared with the buffer of about 60 characters used by the modified assembler. So you cannot have a long string in an ASCII directive, nor can you fit in comments after the operand field of an instruction line. But you can still have normal comment lines (identified by an asterisk as the first character), as long as they are shorter than 24 characters.

TABLE 2: User-accessible subroutines in PIPBUG2

LABEL	FUNCTION	CALL BY
CHIN	Inputs a character to R0 from the serial terminal	ZBSR *0009 (BB 89)
COUT	Outputs a character from R0 to the serial terminal	ZBSR *0007 (BB 87)
BIN	Reads two hex chars from the terminal, forms byte in R1	ZBSR *000D (BB 8D)
BOUT	Prints the byte in R1 as a two-digit hex number (Data in R0 is destroyed)	ZBSR *000B (BB 8B)
LKUP	Converts a hex char in R0 into a 4-bit number (returned in R0 also)	ZBSR 0026 (BB 26)
GNUM	Fetches a 4-digit number from the input buffer, stores in R1 and R2	ZBSR *000F (BB 8F)
STRT	Stores R1, R2 into 80D, 80E	ZBSR 0021 (BB 21)
INCRT	Increments contents of 80D, 80E	ZBSR 0017 (BB 17)
CRLF	Sends CR, LF to terminal	BSTA, UN 01A9 (3F 01 A9)
CHNG	Converts the byte in R0 into two hex chars returned in R1, R2	BSTA, UN 028D (3F 02 8D)
FORM	Outputs 3 spaces to terminal	BSTA, UN 0360 (3F 03 60)
GAP	Output 50 spaces to terminal	BSTA, UN 0364 (3F 03 64)

# Use your 2650 system to generate random Morse!

Trying to learn Morse code? The best way is to have an obliging "old timer" send you random groups of letters and numbers, so that you don't anticipate or "journalise". For those lacking an obliging friend here is the next best thing — a program which turns your 2650 Mini Computer into a random Morse generator.

by **RICHARD ROGERS, VK7RO**

4/439 Huon Road, South Hobart 7000

One of the common errors of beginners in copying Morse code is to "journalise", or write down the end of a word before it has been sent! Random code groups are an excellent practice material to help combat this tendency. Once you are able to copy random groups, plain language will seem easy. Also, with random code there are many more chances to hear the letters which occur infrequently in plain language.

The program described here was originally written for my Central Data 2650 system, but the program as listed has been modified to suit systems using the Pipbug monitor program, like the EA 2650 Mini Computer.

The program generates five-character groups consisting of four letters and one figure, eg ZF90B 8JLUY etc, at speeds ranging from 3 to 25 words per minute. The starting speed is selectable and the speed increases by one WPM every five minutes. The current speed is displayed on the VDU. Below 10WPM, the characters are sent at a 10WPM rate but the spaces between the characters are increased.

As written, the program generates a tone at the 2650 flag output. The tone frequency used is ignored by a 110 baud VDU and nothing is printed on the screen during the morse output. I use a loudspeaker in series with a 1000 ohm resistor, connected between the output of the flag buffer and earth, as a monitor.

The program may be changed to give a voltage suitable for controlling an external oscillator by changing the code at 04A3 from 76 to 74.

Some NOP's are provided within the program to facilitate the use of any other output port. For instance, the use bit 0 of output port D as the tone output, the following code changes are required.

05EB change from C0 C0 C0 C0 to 04 00 F0 C0

049D change from 74 40 C0 C0 to 04 01 F0 C0

04A3 change from 76 40 C0 C0 to 04 00 F0 C0

04AE change from 76 40 C0 C0 to 04 00 F0 C0

The program may also be modified to generate five character groups of

mixed letters, figures and punctuation by changing 04F8 from 18 26 04 1A to C0 C0 04 30.

My thanks to Ron Brown, VK7ZRO, for allowing me to test the program on his system.

```

0440 1F 05 E7 60 88 A8 90 40 28 D0 08 20 78 B0 48 E0
0450 A0 F0 68 D8 50 10 C0 30 18 70 98 B8 C8 7C 3C 1C
0460 0C 04 04 C4 E4 F4 FC 56 CE E2 32 7A 86 94 B4 B6
0470 4A 8C 54 00 00 E5 00 19 15 1A 0F 3B 2F 3B 2D 3B
0480 2B 3B 29 3B 27 3B 25 3B 23 17 3B 0F 3B 0D 3B 0B
0490 3B 1A D1 45 FE E5 00 18 6C 1B 5A 06 00 74 40 C0
04A0 C0 3B 14 76 40 C0 C0 3B 0E FA 72 17 06 00 76 40
04B0 C0 C0 3B 03 FA 7B 17 04 5C F8 7E 17 09 14 0D 64
04C0 D3 05 01 E5 17 1A 02 05 00 C9 07 8D 64 D3 CD 64
04D0 D3 17 00 0E 01 02 03 04 05 06 07 08 09 0A 0B 0C
04E0 0D 0E 0F 10 11 12 13 14 15 16 07 05 04 05 C8 29
04F0 0A 24 3B 3C C8 20 08 1E 18 26 04 1A C8 1B 0A 17
0500 3B 2E C8 13 C1 0D 64 43 CF 65 1A A7 01 14 08 06
0510 A4 01 C8 02 18 62 00 00 00 00 00 00 00 00 00 00
0520 04 0A C8 75 0A 72 3B 08 C8 6E C1 0D 64 5D 1B 58
0530 3F 04 8C 82 1A 03 E8 61 16 A8 5E 1B 77 3F 04 EA
0540 07 06 0F 65 19 C1 3F 04 75 3B 90 3B 8E FB 73 04
0550 01 88 07 E4 00 15 C8 02 1B 63 00 04 00 0E 25 97
0560 14 3F 02 84 1B 77 01 E0 00 00 00 00 77 0E 0B 79
0570 05 00 07 11 75 01 D1 B5 01 18 04 E9 6C 1A 06 77
0580 01 A9 66 77 01 06 02 0E 45 6A D0 CE 65 6A 5A 77
0590 FB 64 09 57 75 FF 17 00 0A 0A 0D 52 41 4E 44 4F
05A0 4D 20 4D 4F 52 53 45 20 44 45 20 56 4B 37 52 4F
05B0 0A 0D 30 33 20 2B 20 32 35 20 57 50 4D 0A 0A 0D
05C0 53 54 41 52 54 49 4E 47 20 53 50 45 45 44 3F 0A
05D0 0D 00 0A 0D 20 20 57 50 4D 00 E7 30 1E 00 1D E7
05E0 39 1D 00 1D 47 0F 17 76 40 75 FF C0 C0 C0 C0 06
05F0 00 3F 05 5D 3F 04 8C 12 1A 7A 3F 02 86 C3 3F 02
0600 84 3B 57 D3 03 D0 D0 83 CC 05 68 3F 02 86 C3 3F
0610 02 B4 3B 46 8F 05 68 CF 05 68 06 3E 3F 05 5D 0F
0620 05 68 E7 19 1D 00 1D E7 03 1E 00 1D E7 09 19 10
0630 A7 02 04 79 06 02 B2 FB 7D CC 05 5C 05 30 1B 1A
0640 CF 05 69 0C 05 66 CC 05 6A 0C 05 67 CC 05 6B 3F
0650 05 6C 04 89 CC 05 5C 0D 05 6B CD 04 9C D1 CD 04
0660 AB 0C 05 68 C1 D1 D1 B1 CC 05 54 20 CC 05 5A 3F
0670 05 3D 06 37 3F 05 5D 0D 05 68 E5 19 18 02 05 01
0680 CD 05 68 E5 0A 1A 0A E5 14 1A 04 85 0C 1B 02 85
0690 06 3F 02 69 1F 06 1A

```

At right is the full hex listing of the author's random Morse program. It starts at 0440.



# The S-100 Bus & how to interface a 2650 to it

Most computer enthusiasts have heard about the S-100 bus system, and that a wide variety of memory boards, floppy disc controllers, video interfaces, speech synthesisers and other fancy peripheral boards are made for it. But do you know how the S-100 bus works, and how it evolved? This article describes the basic S-100 system and tells you how to provide your 2650 Mini Computer with an S-100 interface.

by JAMIESON ROWE

Back in January and February 1975, the US magazine *Popular Electronics* described a build-it-yourself microcomputer project called the "Altair 8800". Based on the 8080 microprocessor, which had not long been released by Intel, the Altair had been designed by MITS, Inc, a firm in Albuquerque, New Mexico. In fact the authors of the *Popular Electronics* articles were two of the MITS engineers responsible for the design: H. Edward Roberts and William Yates. Following publication of the articles, MITS began selling the Altair in both kit and fully assembled form.

The Altair 8800 wasn't the first microcomputer described for home construction. The US magazine *Radio-Electronics* had described a machine called the "Mark-8" in their July 1974 issue, while here at *Electronics Australia* we had begun to describe our EDUC-8 design in the following month. But in the US in particular, the Altair became very popular — so popular, in fact, that it is generally regarded as having launched the US hobby computer industry.

Although the original Altair design used permanently wired multi-conductor ribbon cable to interconnect the various printed circuit boards (PCBs), MITS soon changed over to a motherboard and plug-in PCB system to permit more convenient expansion. The plug-in PCB cards were double sided and mated with 100-way edge connector sockets having two

rows of 50 contacts spaced on 0.125in (3.2mm) centres.

Not all of the 100 connections provided by the sockets were actually used for the Altair's interconnection "bus" lines. In fact only about 60 were used initially, the rest being left for future expansion. Sixteen lines were used for addresses, eight lines each for data into and out of the processor, and the remaining 28 lines for control signals and power supply rails.

As the popularity of the Altair design grew, other manufacturers hopped on the bandwagon with memory boards and a variety of peripheral interface boards, all designed to plug into the Altair's 100-way sockets and hook up to its interconnection bus. The "Altair bus" thus became a de facto interconnection standard, followed fairly closely by everyone who wanted to make plug-ins for the Altair.

Then alternative processor boards and complete computers started to appear. These were obviously designed to compete with the Altair computer, but used the same nominal interconnection bus so that they could take advantage of the variety of available plug-ins to offer the same degree of expansion flexibility.

It was not practical for competing computer manufacturers to continue calling the de facto interconnection standard the "Altair bus", so it became known as the "S-100 bus".

At this stage it should be noted that because the original Altair machine

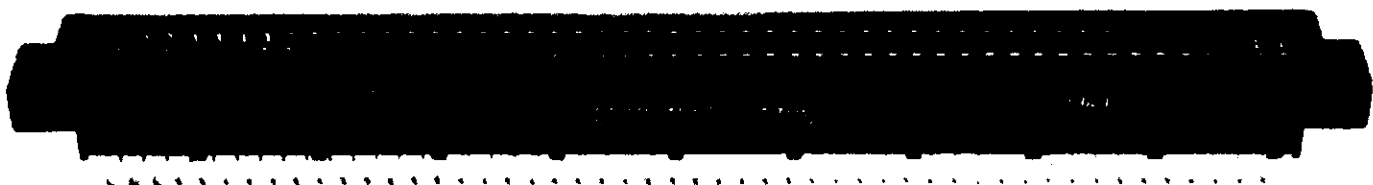
used an 8080 processor, many of the control signals on the Altair bus were basically 8080 control signals. This posed no problems as far as the first few competing machines were concerned, as they too used the 8080 processor. So for a while at least, the S-100 bus was basically a "pure" Altair/8080 standard.

But as time wore on, other processors started to appear, and many of these were "later generation" processors which neither required nor generated all of the control signals used by the 8080. As a result, manufacturers of these new processor boards were faced with either making the new processors "pretend" to be an 8080, or producing S-100 boards which ignored some of the control signals which had been used on the original Altair bus.

Predictably, some took one course and some the other. As a result the newly named S-100 bus began to diverge from the original "pure 8080" Altair standard. The divergence grew even more as those making memory and peripheral plug-in boards began to take advantage of some of the features offered by the newer processors and dedicated controller chips.

So what happened was that although the S-100 bus system had become an "industry standard", its effectiveness as a standard dropped significantly. Whereas it had been possible to plug virtually any board made for the Altair bus into an Altair machine and get it going almost immediately, people soon found that all boards made for the S-100 system were by no means equal. There could be all sorts of problems in trying to combine S-100 boards from different manufacturers, and some S-100 boards just couldn't be made to work together at all — either because of signal timing differences, or because some boards needed signals that the others didn't produce.

Nowadays, the S-100 bus system is still regarded in the USA as one of the



The 100-way edge connector socket used by S-100 plugins. It has two rows of 50 contacts, spaced on 3.2mm centres and

numbered 1-50 on one side, 51-100 on the other (running in the same direction). Courtesy Radio Despatch Service.



# TABLE 1: THE MAIN S-100 BUS SIGNALS

Pin	Signal	Explanation
1	+8V	Unregulated input to +5V regulators on plug-in cards.
2	+16V	Positive unregulated voltage supply.
3	XRDY	External Ready — ANDed with PRDY (pin 72) and connected to READY on the 8080. If XRDY and/or PRDY are pulled low, the CPU will enter a Wait or memory cycle extend state until both are high. XRDY is often used as a front panel control and can allow single stepping. PRDY is usually used to signal valid data from slow memory.
4	VI0	Vectored Interrupt 0 — A vectored interrupt system is used when very fast multiple interrupt response is required and is implemented with a special circuit card.
5	VI1	Vectored Interrupt 1
6	VI2	Vectored Interrupt 2
7	VI3	Vectored Interrupt 3
8	VI4	Vectored Interrupt 4
9	VI5	Vectored Interrupt 5
10	VI6	Vectored Interrupt 6
11	VI7	Vectored Interrupt 7
12	—	These pins not standardised.
13	—	
14	—	
15	—	
16	—	
17	—	
18	STAT DSB	Status Disable — A low on this line puts the status line buffers SMEMR, SINTP, SMI, SOUT, SHLTA, SSTACK, SWO, and SINTA into a high impedance state.
19	C/C DSB	Command/Control Disable — A low on this line puts the command/control line buffers PHLDA, PSYNC, PDBIN, PINTE, PWR, and PWAIT into a high impedance state.
20	UNPROT	Unprotect — A positive pulse resets the Protect flipflop on the currently addressed board so that it can accept data. (Compare with PROT, pin 70.)
21	SS	Single Step — Used by front panel. A high disables input buffer while panel drives bidirectional data bus.
22	ADD DSB	Address Disable — A low on this line puts the 16 address line buffers into a high impedance state.
23	DO DSB	Data Out Disable — A low on this line puts the 8 processor data output line buffers into a high impedance state.
24	$\phi 2$	Phase 2 clock — The master timing signal for the bus in 8080-based systems.
25	$\phi 1$	Phase 1 clock
26	PHLDA	Halt Acknowledge — Processor command/control output signal which goes high following a HOLD signal; it indicates that the data and address buses have gone to the high impedance state and the processor has entered the HOLD state after completion of the current machine cycle.
27	PWAIT	Wait — Command/control signal out which, when high, acknowledges that processor is in a Wait or extended memory cycle state.
28	PINTE	Interrupt Enable — Command/control signal out which indicates condition of interrupt Enable flipflop.
29	A5	Address Bit 5
30	A4	Address Bit 4
31	A3	Address Bit 3
32	A15	Address Bit 15
33	A12	Address Bit 12
34	A9	Address Bit 9
35	DO1	Data Out Bit 1
36	DO0	Data Out Bit 0
37	A10	Address Bit 10
38	DO4	Data Out Bit 4
39	DO5	Data Out Bit 5
40	DO6	Data Out Bit 6
41	DI2	Data In Bit 2
42	DI3	Data In Bit 3
43	DI7	Data In Bit 7
44	SM1	8080 status output signal which, when high, indicates that the current bus cycle is an op code fetch.
45	SOUT	Status output signal which, when high, indicates that the address bus contains the address of an output device and the data bus will contain the output data when PWR is active (low).

(Continued on next page)

for the signal concerned, together with a brief explanation of the signal's function. The information should be fairly self evident, but a few supplementary comments may help to make things clearer.

Note first that no signals are specified for pins 12-17 and pins 55-67 inclusive. This does not signify that these pins do not carry signals, or that they are ignored by S-100 boards and systems. Quite the contrary; in fact, many current S-100 systems do employ these pins to carry quite important signals. The problem is that use of the pins is not sufficiently standardised to allow each one to be given a fixed signal allocation.

For example pin 13 is used in various systems to carry interrupt request (IRQ), phase 3 shift clock (CK3), standby power (STDBY), pause status (PAUSE) or memory bank 8 select. Similarly pin 67 is used in various systems to carry signals such as phantom disable (PHANTOM), non-maskable interrupt (NMI), refresh disable (RFSHDSBL), memory disable (MDSBL), refresh (RFSH), video sample clock (SCLK) or address line 19 (A19).

So for some S-100 boards, these pins may carry signals which are essential for correct operation. But because the signals are not standardised, it is not really feasible to provide them in a generalised S-100 interface.

The next thing to note is that among the standardised signals, there is a certain amount of duplication and functional overlapping. For example XRDY-bar (pin 3) and PRDY-bar (pin 72) both perform the same function, while PINT-bar (pin 73) and VI0-VI7 (pins 4-11) overlap in their functions. These redundancies are largely the result of the ad hoc way in which the S-100 bus was developed.

It should also be noted that many of the S-100 control signals are basically those used by an 8080 microprocessor. As such these signals are often not particularly compatible with either more modern processors, or peripherals designed to go with them. It may be either difficult to derive the 8080-type signals from those actually generated, or difficult to use them once derived and fed along the bus, or both.

So, in providing an S-100 interface, you are faced with the choice of either making your processor "pretend" to be an 8080 and using the standard S-100 control signals, or ignoring these signals and using alternative control signals on some of the unstandardised bus pins.

The first approach will tend to give you somewhat greater compatibility with the wide range of available S-100 plug-ins. But it may also involve clumsy interfacing logic, and prevent you from taking full advantage of the features offered by a more modern processor. The second approach may tend to be more elegant and more powerful, but

tends to introduce hassles when you try to use certain S-100 boards. The choice is up to you.

Of course, some of the S-100 control signals are more important than others. Some signals are only needed if you plan to have a fancy front panel on your system — a feature which is not as popular nowadays as it was. Others are only used for things like a hardware-implemented single step facility, or stack management hardware external to the processor. If you don't want these facilities, or don't need them, then the signals can be ignored.

Perhaps the remaining general point that should be made about the S-100 bus is that as you can see, it uses two 8-bit data buses: one for data into the processor, and the other for data out of the processor. This is a carryover from the original Altair design, and is again a little clumsy by modern standards. In general only one of the two buses is ever in use at any instant, so it would be more elegant and efficient to have a single bidirectional bus.

But if you want to make your interface compatible with most of the S-100 plug-ins, you have to provide for the two separate data buses — clumsy though they may be. Of course you can always provide your own bidirectional bus as well, using eight of the unstandardised pins. Just make sure that the pins you use aren't needed by any of your S-100 plug-ins for special control signals.

Well then, let's get down to specifics. What's involved in providing an S-100 interface for your 2650 Mini Computer system?

Before going any further, I would like to stress that the remainder of this article consists basically of a set of suggestions, rather than the description of an interface that has been built up and tested. The circuit diagram given has not been tested, as this would have involved a considerable amount of time and effort which could not really be justified in view of the limited interest. But it has been prepared from a careful survey of S-100 literature and reference material, and I believe it to be fully practical.

Basically if you want to provide your 2650 system with an S-100 interface which provides each and every one of the various standardised control signals, it isn't easy. But on the other hand, some of the control signals turn out to be unnecessary in a 2650-based system, except in very rare circumstances.

The interface shown in the circuit diagram provides only the main control signals, but should be suitable for interfacing your 2650 system to most S-100 plug-ins.

Let's run through the circuit, starting from the bottom and working upward. First are the 16 address lines AD0-AD15, buffered by a pair of 81LS95 or similar Tri-state octal buffers. The inputs for

46	SINP	Status output signal which, when high, indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when PDBIN is active.
47	SMEMR	Memory Read — Status output signal which, when high, indicates that the data bus will be used to read memory data.
48	SHLTA	Halt Acknowledge — Status output signal which, when high, acknowledges that a HALT instruction has been executed.
49	CLOCK	Phase 2 clock inverted
50	GND	Signal and power ground
51	+8V	Same as pin 1
52	-16V	Negative unregulated voltage supply
53	SSW DSB	Sense Switch Disable — A low disables the data input buffers so the input from the sense switches may be strobed onto the bidirectional data bus.
54	EXT CLR	External Clear — A low clears I/O devices.
55	—	} These pins not standardised.
56	—	
57	—	
58	—	
59	—	
60	—	
61	—	
62	—	
63	—	
64	—	
65	—	
66	—	
67	—	
68	MWRITE	Memory Write — A high indicates that the current data on the Data Out Bus is to be written into the memory location currently on the address bus.
69	PS	Status of protect flipflop (low for protect).
70	PROT	Protect — A positive pulse sets the protect flipflop.
71	RUN	Run — A high indicates that the Run/Stop flipflop is set to RUN.
72	PRDY	Ready — See pin 3.
73	PINT	Interrupt Request — A low causes the processor to recognise an interrupt request at the end of the current instruction or while halted. If the CPU is in the Hold state or if the Interrupt Enable flipflop is reset, it will not honour the request.
74	PHOLD	Hold — A low requests the processor to enter the Hold state. It allows an external device to gain control of the address and data buses as soon as the current machine cycle is completed.
75	PRESET	Reset — A low causes the contents of the program counter to be cleared and the instruction register is set to 0.
76	PSYNC	Sync — The command/control signal out which, when high, identifies the beginning of an 8080 machine cycle.
77	PWR	Write — The command/control signal out which, when low, signifies the presence of valid data on the Data Out bus.
78	PDBIN	Data Bus In — The command/control signal out which, when high, requests data on the Di bus from the addressed memory or I/O.
79	A0	Address Bit 0
80	A1	Address Bit 1
81	A2	Address Bit 2
82	A6	Address Bit 6
83	A7	Address Bit 7
84	A8	Address Bit 8
85	A13	Address Bit 13
86	A14	Address Bit 14
87	A15	Address Bit 15
88	DO2	Data Out Bit 2
89	DO3	Data Out Bit 3
90	DO7	Data Out Bit 7
91	DI4	Data In Bit 4
92	DI5	Data In Bit 5
93	DI6	Data In Bit 6
94	DI1	Data In Bit 1
95	DI0	Data In Bit 0
96	SINTA	Interrupt Acknowledge — The status output signal which, when high, identifies the instruction fetch cycle(s) that immediately follow an accepted interrupt request presented on PINT.
97	SWO	Write/Output — The status output signal identifying a bus cycle which, when low, transfers data from processor to memory or I/O.
98	SSTACK	Stack — Status output signal which indicates, when high, that the address bus holds the pushdown stack address from the Stack Pointer and that a stack operation will occur on the current cycle.
99	POC	Power On Clear — Generated by PRESET or power on. Used to reset CPU and I/O devices.
100	GND	Signal and power ground

the buffers are taken from the address lines (already buffered) on the 2650 Mini Computer's expansion board. —

Note that the S-100 bus requires 16 address lines, whereas the 2650 system only has 15 lines available (AD0-AD14). The input of the 16th buffer is therefore tied permanently to ground.

One enable input of each of the 81LS95 address buffer devices is connected to a gate. This allows the buffers to be disabled, and the S-100 address lines to be floated in a high impedance state, either in response to the ADD DSB-bar signal (pin 22) or when the processor is halted. Other S-100 boards are thus able to take control of the address lines, for things like DMA (direct memory access) data transfers.

Moving upward, we find two more 81LS95 octal buffers, the first used to buffer the S-100 data out lines DO0-DO7, and the second to buffer the S-100 data input lines DI0-DI7. As with the address buffers, the data out buffers are controlled by another gate, so they can be disabled either in response to the DO DSB-bar signal (pin 23) or when the processor is halted.

In addition, both the DO and DI buffers are controlled separately by two of the outputs of an 82S103 device. This is a programmable gate array, which Signetics and Philips are making available pre-programmed with the logic functions necessary to generate eight key S-100 control signals from the existing 2650 control signals OPREQ, R-bar/W, M/IO-bar and WRP.

As you can see, besides the two data buffer control signals the device also produces the S-100 signals SOUT (pin 45), SINP (pin 46), SMEMR (pin 47), PWR-bar (pin 77), PDBIN (pin 78) and MWRT (pin 68). So it really takes some of the hassles out of making the 2650 "pretend" to be an 8080!

The preprogrammed version of the 82S103 is coded with the suffix CK1179, and is available from your normal parts supplier on order from Philips Industries. It should cost you less than \$10, including tax.

The programming chart for the 82S103/CK1179 is shown in Table 2, for the benefit of those who want to analyse the logic functions involved in producing the S-100 signals. Note that device inputs I4, I5 and IF are not used, and can be left unconnected; similarly the ninth device output F8 is not used. Note also that inputs I6 to IE inclusive are all effectively programmed to act as active-high enable inputs, so that they must all be taken to logic high level for any output to be enabled.

What this means is that these inputs may effectively be used to disable the S-100 interface, whenever the processor is dealing with the memory and I/O ports provided in the original 2650 system. This is done simply by connecting active-low enable signals from the existing 2650 system to some of the 82S103 enable inputs, as shown. The remaining enable inputs are simply tied

TABLE 2: 82S103/CK1179 PROGRAMMING CHART

INPUT VARIABLES																OUTPUT FUNCTIONS		
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	IA	IB	IC	ID	IE	IF	OUTPUT	POLARITY	NAME
H	H	—	—	—	—	H	H	H	H	H	H	H	H	H	—	F0	L	DO enable
H	L	—	—	—	—	H	H	H	H	H	H	H	H	H	—	F1	L	DI enable
—	H	L	—	—	—	H	H	H	H	H	H	H	H	H	—	F2	H	SOUT
—	L	L	—	—	—	H	H	H	H	H	H	H	H	H	—	F3	H	SINP
—	L	H	—	—	—	H	H	H	H	H	H	H	H	H	—	F4	H	SMEMR
H	H	—	H	—	—	H	H	H	H	H	H	H	H	H	—	F5	L	PWR
H	L	—	—	—	—	H	H	H	H	H	H	H	H	H	—	F6	H	PDBIN
H	H	H	—	—	—	H	H	H	H	H	H	H	H	H	—	F7	H	MWRT
—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	F8	—	—

I0 = OPREQ; I1 =  $\bar{R}/W$ ; I2 = M/ $\bar{T}$ O; I3 = WRP; I6—IE = enable inputs (active high)

high via pullup resistors.

So if you have already provided your 2650 system with 7k of RAM, in addition to the 1k PIPBUG ROM, and this memory is all in page 0, you obviously won't want any S-100 boards to respond when this part of memory space is being addressed. This is achieved quite simply by connecting the PAGE 0-bar enable signal (from the 74LS138 decoder on the expansion board) to say input I6 of the 82S103, as shown.

Similarly if you have already implemented the four 2650 non-extended I/O ports, you can disable the S-100 interface whenever these are being addressed simply by connecting the C-bar and D-bar signals (again from the 74LS138 on the expansion board) to inputs I7 and I8.

Inputs I9-IE inclusive are still available, and may be used to disable the S-100 interface for any other memory blocks or I/O addresses you may have already implemented. All you need to do is derive an active-low signal from each enable signal, and connect these to the spare inputs. Since there are six available inputs (apart from those used for PAGE 0-bar, C-bar and D-bar), this should provide enough flexibility for almost any situation.

Moving further up the circuit diagram, we find the circuitry for the remaining S-100 control signals.

The XRDY-bar and READY-bar signal lines are connected via a NAND gate to the OPACK-bar input of the 2650 processor, to allow memory cycles to be extended for slow memory. Note that the 2650 does not enter a wait state when this is done, unlike the 8080; in fact the 2650's "wait" state corresponds to the 8080 "hold" state. However, the function of the two S-100 "ready" lines should be unaffected, as these are basically used for memory cycle extension.

Don't forget that in the original 2650 Mini Computer, the OPACK-bar input of the 2650 (pin 36) was permanently earthed. So the copper track of the PC board will have to be cut, to disconnect this earth and allow the pin to be controlled. As the copper track concerned made other earth connections, a wire

link will need to be added to maintain these connections.

A flipflop is used in the interrupt logic, as the INTREQ-bar input of the 2650 must be held low until it is acknowledged by a high on the INTACK output. A link (L4) is shown, to allow you to select whether the PINT-bar or VIO inputs from the S-100 bus are used to set the flipflop and initiate interrupts. As may be seen the flipflop is reset by either INTACK going high, or the reset signal. The latter ensures that the flipflop is always in the reset state when power is applied to the system.

The dashed gate shown in the reset circuitry is actually the 74LS38 gate originally used as an inverter in the reset line of the 2650 Mini Computer. The second input (pin 4), which was originally tied to logic high, is now used to accept the S-100 PRESET-bar signal (pin 75). This allows S-100 plug-ins to reset the system if necessary.

Links L1, L2 and L3 are shown to indicate that you have a choice in deciding which signals to feed out on the S-100 clock lines  $\phi 1$  (pin 25),  $\phi 2$  (pin 24) and CLOCK-bar (pin 49). As the 2650 does not use the two-phase clock system of the 8080, the choice of signals fed on these lines will depend upon the requirements of the S-100 plug-ins you want to use.

If the plug-ins basically only use the  $\phi 2$  signal as a data strobe, for example (this is fairly common), you will probably find that the buffered OPREQ ("BOPREQ") signal will be most suitable. On the other hand one or other of the S-100 clock signal may be used as a source of synchronised high-frequency signals (say by a video interface board, as a dot shift clock and input to the timebase divider), in which case the 1MHz master clock signal may be more appropriate. Or you may need to provide both BOPREQ and the 1MHz clock, on different lines, for use by different boards. It will depend upon the S-100 boards you are using.

The only remaining point to note about the interface circuit concerns the power supply rails. By convention, S-100 plug-in boards have their own 5V regulators, and are supplied with an

unregulated (or only pre-regulated) input of 8V DC. So the main supply rail of the S-100 bus is the +8V line connected to pins 1 and 51, and referred to the ground pins 50 and 100.

The +16V and -16V rails shown on pins 2 and 52 are secondary supply rails, used rather less frequently by plug-ins requiring higher voltage for op amps, D-to-A converters and so on. These boards generally have their own 12V regulators, working from the unregulated 16V lines.

If you don't plan to use plug-ins which require the higher rails, you can forget the +16V and -16V power supplies. All you will need is an 8V power supply, capable of supplying the current needs of your S-100 plug-in boards. Needless to say, the ground reference of the 8V supply will need to be connected to the ground side of the existing 2650 system power supply.

Finally, a few suggestions about the physical side of your S-100 interface.

I would suggest that you don't try to design your own interface and motherboard. There are a number of good S-100 motherboards already available, at quite reasonable cost. Similarly there are quite a few S-100 "development boards", complete with gold-plated double sided edge connector pads, and designed specifically to allow you to wire up custom plug-ins.

As both these items are readily available, it seems to me that the easiest way to build your S-100 interface is to wire it up on one of the development boards, with one or more lengths of rainbow ribbon cable to connect the board into your existing 2650 system. The S-100 connections can be made directly to the appropriate edge connector pads, so that the interface board can then be plugged into a standard S-100 motherboard.

This way, you won't have to design or etch any custom PC boards; you'll be using readily-available standard boards. The interface will simply become another S-100 plug-in, which happens to have an "umbilical cord" back into your 2650 system. You also have the option of using a standard S-100 power supply, case and card cage if you wish.

# Improving the 2650 mini Line Assembler

If you have used the 2650 Mini Assembler described in the April 1979 issue, you'll know it is a little inflexible when you want to correct typing errors. Here are two small modifications which make it very much easier and faster to use.

by **A. M. KOLLOSCH**

Higginbotham Avenue, Armidale NSW 2350

After using the 2650 Mini Assembler for a while, I became a little irritated by its lack of any facility to let you correct minor typing errors as soon as you notice them, before the end of the line. As you'll know if you've used the assembler, you have to finish the line and either reset the original to step back and re-type the line (assuming the assembler doesn't throw you out), or restart the assembler and also reset the origin (if it has thrown you out). In both cases the lack of flexibility is quite inconvenient, as well as being tedious and time consuming.

To get around these problems I have developed two modifications for the assembler, which make it considerably faster and more convenient to use.

The first modification is to the line input routine.

## \*MODIFIED LINE INPUT ROUTINE

```

IACD 0700      LODI,R3      00
IACF E73C      COMI,R3      3C
IAD1 1C15B0     BCTA,Z      15B0
IAD4 3F236      BSTA,UN      0236
IAD7 E47F      COMI,R0      7F
IAD9 9303      BCFR,Z      1AE6
IADB 03         LODZ,R3
IADC 1871      BCTR,Z      1ACF
IADE 0428      LODI,R0      03
IAE0 B3A0      ZBSR      *0222
IAE2 A701      SUBI,R3      01
IAE4 1B69      BCTR,UN      1ACF
IAE6 0503      LODI,R1      03
IAE8 ED7AC9     COMA,R1      1AC9#
IAEB 1309      BCTR,Z      1AF6
IAED F979      BDRR,P      1AE8
IAEF CF7A02     STRA,R3      1A32#
IAF2 BBA0      ZBSR      *0220
IAF4 D859      BIRR,UN      1ACF
IAF6 CF0429     STRA,R3      0429
IAF9 CD042A     STRA,R1      042A
IAFC 0720      LODI,R3      00
IAFE 9BA5      ZERR      *0025
  
```

ABOVE: A disassembler listing of the modified line input routine, which lets you step back to correct typing errors.

RIGHT: A similar listing of the modifications to allow you to re-type lines that are thrown out by the assembler. It now types "? ERROR", and repeats the address.

put routine. Its effect is to let you step back along the line input buffer, using "delete" (rubout) characters. So if you spot a typing error before you have finished a line, you can step back to it and then type the rest of the line again before typing a carriage return.


Actually the modified routine is arranged to echo "backspace" characters to the terminal, instead of the incoming "delete" characters, so if your terminal can perform the backspace function it will step the cursor back to show you where you are go-

ing. With terminals which don't perform backspacing you'll have to count back yourself, but this is usually no problem.

A disassembler listing of the modified input routine is shown below. It is one byte longer than the existing routine, ending at X'1AFF instead of 1AFE.

The second modification is a little more elaborate. It involves an additional error handling routine, a modified starting sequence and a couple of subroutines, together with changes to all the error throwout addresses.

The idea of this modification is that instead of throwing you right back to PIPBUG when it finds an error, the assembler now prints a curt "? ERROR" message, and reprints the address of the line concerned so you can re-type it correctly.

The disassembler listing for this modification is also shown below. I think you'll find it worthwhile. 

## \*MODIFICATION FOR IMPROVED ERROR HANDLING

```

159E 00 02
15A0 0D240D     LODA,R1      040D
15A3 0E040E     LODA,R2      040E
15A6 C976      STRR,R1      159E
15A8 CA75      STRR,R2      159F
15AA 17         RETC,UN
15AB 0971      LODR,R1      159E
15AD 0A70      LODR,R2      159F
15AF 17         RETC,UN
15B0 75FF      CPSL      FF
15B2 740F      CPSU      0F
15B4 0503      LODI,R1      03
15B6 0D75C3     LODA,R1      15C3#
15B9 BBA0      ZBSR      *0020
15BB F979      BDRR,P      15B6
15BD BBAS      ZBSR      *0025
15BF 3B6A      BSTR,UN      15AB
15C1 1F160C     BCTA,UN      160C
15C4 524F52     ROR
15C7 524520     RE
15CA 203F      ?
160C 3BF0      BSTR,UN      *15FE
160E 3F15A0     BSTA,UN      15A0 GO TO NEW SUBR
1611 3BDC      BSTR,UN      *15EF
1613 02         LODZ,R2
1614 C1         STRZ,R1
1615 3BD8      BSTR,UN      *15EF
1617 042E      LODI,R0      2E
1619 3F02B4     BSTA,UN      02B4
161C 3BD3      BSTR,UN      *15F1
  
```

## \*MODIFIED START SEQUENCE

MODIFY THE ERROR THROWOUT ADDRESS TO X'15B0 AT THE FOLLOWING LOCATIONS:

```

162F-30 16BA-2B 1704-05 1745--46
17B0-81 1A93-94 1AD2-D3
  
```

# "Trace" routine helps debug 2650 programs

When you're trying to debug a tricky program in assembly language, a breakpoint isn't always the answer — you can generally only call it once. Here is a "trace" routine for 2650 systems which can be rather more helpful. You can call it any number of times, and each time it is called it prints out the contents of all processor registers.

by JAMIESON ROWE

Like most small microcomputer systems, the 2650 Mini Computer provides only one debug aid: a pair of breakpoints, which are software implemented by the PIPBUG monitor program. These can be quite handy, but there are many occasions when they just don't help enough.

For a start, each breakpoint can only be used once. When it is executed once, PIPBUG replaces it with the original instruction. So you can't use the breakpoints to track down bugs which are inside loops, for example — the breakpoint disappears first time you go around the loop!

The other main drawback is that the breakpoint "runtime" routine simply saves the processor register contents in RAM, and then transfers control back to the PIPBUG command loop. So if you actually want to examine the registers, you then have to use the PIPBUG "see and set the registers" (S) command. This can be very tedious and time consuming when you have to use the breakpoints over and over.

Most of these disadvantages can be avoided by using the "trace" routine described in this article. It was developed some months ago for this very purpose, and since then it has helped me considerably in tracking down elusive bugs.

Basically it consists of a subroutine which may be called any number of times, by temporarily patching appropriate BSTA or BSTR instructions into the program you are trying to debug. When it is called, it first saves the contents of all of the 2650 registers. Then it prints them all out on the terminal, to provide a "snapshot" of the current processor status. Then it restores all of the registers again, and does a return.

Before it prints out the register contents, it prints three spaces. This is to prevent confusion if the program you are testing already involves printing. After the spaces it prints the registers in the same order that they are provided by PIPBUG: R0, R1, R2, R3, R1', R2', R3', PSU and PSL. Finally it prints a carriage return and line feed, so that each "snapshot" is on a different line.

A full source listing of the trace routine is shown below, together with a sample of its operation. As listed the routine is located from 0440 to 0495, but it may be relocated anywhere in page 0 without changes. Note that it stores the processor registers in locations 0400-0408 — ie, the same locations used by PIPBUG for this purpose.

The sample tracing shown below was produced by patching a call into the author's Disassembler program, at the end of the PRINT MESSAGE subroutine (1D9C). The code at 1D9C was changed to 1B39, to branch to 1DD6 where a patch of 3F0440, 1F1D94 was located. As you can see this gives a "snapshot" after each character is printed from the line buffer.

```
*G1B00 1B00 1B02
1 001A0200003100E280
2 001A0201004200E280
3 001A0202003000E280
4 001A0203003000E280
5 001A0204002000E280
6 001A0205003700E280
7 001A0206003600E280
8 001A0207003600E280
9 001A0208003000E280
0 001A0209002000E280
1 001A020A002000E280
2 001A020B002000E280
3 001A020C002000E280
4 001A020D002000E280
5 001A020E002000E280
6 001A020F005000E280
7 001A0210005000E280
8 001A0211005300E280
9 001A0212005500E280
0 001A0213002000E280
1 001A0214002000E280
2 001A0215002000E280
3 001A0216002000E280
4 001A0217002000E280
5 001A0218002000E280
6 001A0219003600E280
7 001A021A003000E280
```

RIGHT: A full listing of the trace routine, complete with comments so that you can see how it works.

LEFT: A sample of the routine in operation. Here it was patched into the Disassembler program, so that a "snapshot" of the processor registers is printed after each character from the Disassembler's line buffer.

\*DIAGNOSTIC "TRACE" ROUTINE FOR  
\*2650 SYSTEMS. J. ROWE MARCH 1979

```
0440 CC0400 STRA,R0 0400 SAVE R0
0443 12 SPSU & PSU
0444 CC0407 STRA,R0 0407
0447 7660 PPSU 60 FORCE TO MARK,INHIBIT INT.
0449 13 SPSL SAVE PSL
044A CC0408 STRA,R0 0408
044D 7710 PPSL 10 FORCE TO BANK 1
044F CD0404 STRA,R1 0404 SAVE BANK 1 REGS
0452 CE0405 STRA,R2 0405
0455 CF0406 STRA,R3 0406
0458 7519 CPSL 19 FORCE TO BANK 0,CLR C & WC
045A CD0401 STRA,R1 0401 SAVE BANK 0 REGS
045D CE0402 STRA,R2 0402
0460 CF0403 STRA,R3 0403
0463 3F035B BSTA,UN 035B PRINT 3 SPACES
0466 07FF LODI,R3 FF SET R3 AS INDEX
0468 0609 LDDI,R2 09 & R2 AS COUNTER
046A 0F2400 LODA,R3 0400+ FETCH SAVED REG
046D C1 STRZ,R1 MOVE TO R1
046E 3F0269 BSTA,UN 0269 PRINT VIA BOUT SUBR
0471 FA77 BDRR,N 046A LOOP BACK TIL DONE 9
0473 3F008A BSTA,UN 008A THEN GIVE CRLF
0476 0D0401 LODA,R1 0401 RESTORE ALL REGS
0479 0E0402 LODA,R2 0402
047C 0F0403 LODA,R3 0403
047F 7710 PPSL 10
0481 0D0404 LODA,R1 0404
0484 0E0405 LODA,R2 0405
0487 0F0406 LODA,R3 0406
048A 0C0407 LODA,R0 0407
048D 92 LPSU
048E 0C0408 LODA,R0 0408
0491 93 LPSL
0492 0C0408 LODA,R0 0408
0495 17 RETC,UN & RETURN
```



# Micro Basic Programs for 2650 systems

## Temperature Conversion, Radio Log

Here are two short programs written in "Micro Basic" — the cut-down version of BASIC developed for small 2650 computer systems by reader Alan Peek. One program converts temperatures from one scale to another, while the other is a program to manage a radio amateur's contact log.

Following our review of Alan Peek's "Micro Basic" for small 2650 microcomputer systems, published in the April 1979 issue, it would seem that quite a few readers have obtained Mr Peek's interpreter and have been working with it. Already two readers have sent in programs they have developed, and as they seem likely to interest readers we are publishing details here.

The first program came from Mr Syd Brooks, of 6 Edgar Street, Ferntree Gully Victoria 3156. It is a simple little program which can convert between the Celsius, Fahrenheit, Kelvin and Rankin temperature scales. Mr Brooks has provided it with a little humour, to add to the interest, along with some checks and reprompts in the event of

an invalid entry when the program is being used.

The complete listing of the program is shown in Fig. 1, with a sample of operation in Fig. 2. As you can see, it is fairly self-explanatory.

```
G1
PROGRAM TO CALCULATE
DEGREES C,K,F OR R GIVEN ONE

WILL YOU GIVE C,K,F OR R? F
PRESS + OR -+
WHAT IS TEMP?212

C=100 K=373 F=212 R=672

WILL YOU GIVE C,K,F OR R?

1 P"PROGRAM TO CALCULATE"
2 P"DEGREES C,K,F OR R GIVEN ONE"
3 P
4 P"WILL YOU GIVE C,K,F OR R? " AA
5 TA=70,A:75,A:67,A:82 G1<
6 P"PRESS + OR -" AB TB=43 L1=X G3>
7 TB=45 L1=X G2>
8 G2<
9 P"WHAT IS TEMP" IE LEX*=E
10 TA=70 G4>
11 TA=82 G4>
12 TA=67 G4>
13 TA=75 G4>
14 LE460+=R,R=E
15 LE460-=F,F32-5*9/=C,C273+=K,E=R G3>
16 LE273+=K,K=E
17 LE273-=C,C9*5/32+=F,F460+=R,E=K
18 TK<0 P"THAT'S IMPOSSIBLE STILL!" G3>
19 TC<400 P"THAT'S COLD!" G2>
20 TC>1999 P"GUESS" G3
21 P
22 P"C="C," K="K," F="F," R="R
23 G3
24 $A=C,F,K OR R B=SIGN F=TEMP$
25 $C=CELSIUS F=FAHRENHEIT K=KELVIN $
26 $R=RANKIN $
27 E
```

*Fig. 1, at left is the full listing of temperature conversion program while Fig. 2 above is a sample of its operation.*

The second program came from Mr Horst Leykam, of 165 Victor Street, Dee Why NSW 2099. Mr Leykam is a radio amateur, with the call sign VK2BHF, and explains that he wrote the program in an effort to produce a more elegant and effective way of maintaining and referencing his contact log.

What this program does is maintain a log of contacts, with each contact represented by callsign, the name of the operator contacted, the date and the readability/signal strength details. It allows you to add to the log "file" when each contact is made, and then to have an automatic search made for previous contacts with the same callsign. This lets you "refresh" your memory regarding the name of the operator, and the previous times that you exchanged calls.

A file can be expanded until it fills the allocated memory buffer space (hex 1000—1FFF, or decimal 4096—8191). As each entry is made to a file, the program tells you how many bytes of memory are left — in decimal. This allows you to save a file at any time on cassette tape, using the normal PIPBUG dump command, and then re-load it into the buffer later using the L command.

The program can search for either the first file entry matching the supplied data (typically the callsign), or for all matching entries. It also allows you to alter an existing file entry, providing the change will fit into the same space. This allows you to correct a previous mistake, when it is discovered.

A full listing of the program is shown in Fig. 3, with a sample of its operation shown in Fig. 4. When the program starts up, it immediately asks you for a command. You have five options, each command being represented by a single digit (1-5) terminated by a carriage return.

1 — Is to enter a new file. Each line should be delimited by a space and an asterisk.

2 — Is to accept a data entry (say the callsign), and search through the current file for a match. The first line found to contain the data will be printed out.

```

1 P"ENTER COMMAND" P G35
2 L4096=Z P"ENTER NEW FILE" S44
3 AB TZ=8190 G33
4 MB>Z JZ TB:13 G1<
5 P"MEMORY LEFT="8190+Z-,"BYTES" LZ=Q G4<
6 P"ENTER DATA" P
7 L4070=C
8 AD
9 MD>C JC TD:13 G1<
10 P"SEARCHING" S44
11 P
12 L4096=Z
13 L4070=C TZ=Q G34
14 MB<Z MD<C
15 TB:D JZ TD:13 G2<
16 TD=13 G2>
17 JZ JC G3<
18 CZ
19 MB<Z
20 TB:42 CZ G1<
21 JZ
22 MB<Z
23 OB JZ TB:42 G1<
24 TF=2 G1
25 TF=3 L4070=C P G12<
26 TF=4 P"TYPE NEW LINE" CZ P
27 CZ
28 MB<Z TB:42 G1<
29 JZ MB<Z TB=42 G2>
30 AB MB>Z G1<
31 P"LINE FULL"
32 G1
33 P"MEMORY FULL" G1
34 P"END OF FILE="Z G1
35 L42=K MK>4095 IF
36 TF=1 G2
37 TF=2 G6
38 TF=3 G6
39 TF=4 G6
40 TF=5 LQ=Z P"ADD TO FILE" P"BYTES LEFT="8190+Z- P G3
41 MB<Z
42 TB:38 JZ G1<
43 LZ=Q G1
44 P"CALLSIGN NAME DATE RST" P R

```

```

G1
ENTER COMMAND
?5
ADD TO FILE
BYTES LEFT=3939
VK2XYZ FRED 23/3/89 594 *
MEMORY LEFT=3905 BYTES
ENTER COMMAND
?3
ENTER DATA
VK2XYZ
SEARCHING
CALLSIGN NAME DATE RST

VK2XYZ FRED 1/2/89 599 *
VK2XYZ FRED 23/3/89 594 *

END OF FILE=4285
ENTER COMMAND
?

```

*Fig. 3 above is the full listing of the amateur log program while Fig. 4, at left, is a sample of its operation.*

that the end of file pointer is set up properly. Otherwise it will be ignored.

Note that searching a full 4K byte file can take up to four minutes. This is mainly because the Micro Basic interpreter is rather slow in operation — although Mr Leykam notes that his program may well be capable of improvement (it was his first programming effort).

Finally, a note regarding Micro Basic itself for the benefit of those who may not have seen the earlier story. Micro Basic is a tiny version of BASIC, written by Alan Peek for small 2650 microcomputer systems. The editor/interpreter for Micro Basic squeezes into a mere 1.6K bytes of memory, so that it can even run in systems with only 4K of RAM (although Mr Leykam's program will require a system with 7K of RAM).

As you may have deduced from the program listings in this article, Micro Basic achieves this remarkable economy by the use of single-character commands, reverse Polish notation and an efficient way of packing the source program into memory. Its unorthodox approach takes a bit of getting used to, but the ability to program rapidly even in small systems makes it well worth persevering.

A cassette of the Micro Basic editor/interpreter (in PIPBUG format) complete with instructions and a full source listing is available from Alan Peek at 10 Gale Street, Woolwich NSW 2110.



- 3 — This is like command 2, except that the program will print out all lines in the current file containing the data.
- 4 — Is to allow altering an existing line in the file. Note that the new line must be the same length as the original.
- 5 — Is to add to the current file. Each new line should be delimited by a space and an asterisk as before.

If the file in memory is to be saved on cassette, the last character in the file should be an ampersand (&). This is required by the alternative start-up routine, in order to set up the end of file pointer when the file is re-loaded into memory. When a file has been re-loaded from cassette, start the program at line 41 instead of line 1, to ensure

# 'Execute' Program for the 2650 minicomputer

Newcomers to computing often wonder just what some of the jargon used means. This article is meant as a remedy to this situation, and attempts to explain with a practical example just what is meant by the term "execute".

by LUDI KRAUS

One of the most often used but most little understood terms bandied about by computer proponents, both professional and amateur alike is "execute". The newcomer to computing, on hearing the expression "let's execute the program", could be forgiven for thinking that this means "let's kill the program".

In fact, the dictionary definition of execute does encompass the meaning "to carry out capital punishment", ie to kill, but includes a host of other meanings as well. Thus we can speak of executing orders, plans or functions, meaning to carry out these duties or acts, or of executing a document such as a will.

In deed, the kill implication is the last meaning listed in my dictionary and is definitely not the meaning intended by our overheard programmer.

If we turn from a dictionary to a glossary of terms, as usually can be found at the rear of elementary programming manuals, we come across an alternative definition, such as the one given below:

**EXECUTE** — to fully perform a specific operation, such as would be accomplished by an instruction or a program.

Our newcomer could now draw the correct conclusion that when we talk of "executing a program", what we really mean is running, or, to be specific, letting the computer run a program.

"But," wails our beginner, "how do you run or execute, to use the jargon, a program?" Well, this requires a special purpose program, rather similar in concept to an assembler or an interpreter.

For the benefit of newcomers confused by those last two large words, an assembler is used to assemble programs, while an interpreter is used to interpret programs. Similarly, we will use an executioner to execute programs. (Enlightening isn't it?).

The program listing included with this article is a fully optioned executioner, intended for use with 2650 computer systems. It will operate with

both machine level languages, and with higher level language, such as Basic or Pascal. It is completely relocatable and is intended to reside in the topmost portion of the user RAM.

The program can also be used in ROM, although in this case its full advantages cannot be reaped. This is because it uses a unique form of error detection and correction, which involves a special section of the program, which modifies itself.

This special section, however, before it runs, first checks to see if it is ROM-resident. If it is, it neatly bypasses itself.

The error correction carries out a series of CRC (cyclic redundancy check) tests, incorporating Hamming code and BCC (binary condition code) tests. These checks and tests enable it to correct any possible errors in the remainder of the program.

This meant, in fact, that it was only necessary to debug the first section of the program, as the second section automatically debugged itself. This saved valuable time and meant that this article could be published one month earlier than anticipated.

In order to use the program with your system, first load the program you wish to execute into the area of RAM it

normally uses. If you have a ROM based system, this step is not necessary.

Next, load in the execute program. You can do this using either the A command of Pipbug, the hex input routines published in the March 1979 issue or the mini assembler published in the April 1979 issue. The program is 73 bytes long, and should be placed at the top of your memory.

If you have the 2650 Mini Computer with only 4k of RAM, start at address X'13AF. If you have implemented the 8k RAM expansion, your starting address should be X'1FAF if you have memory only in page 0, or X'2FAF if you have used the CPU RAM in page 1.

To run the EXECUTE program, simply type G 13AF XXXX CR, where XXXX is the HEX hex address of the first byte or location of the program to be executed. Control will return to Pipbug when the execution is complete. Please note that this may take some time, especially if your program includes a number of absolute and relative addressed indexed instructions.

A disassembly listing of the program has been included with this article, as an aid to those who do not have access to a disassembler. Unfortunately, no comments could be provided with the listing as that would give the game away. However, with the aid of the 2650 Microprocessor manual, novices should be able to work out the way in which the program operates.

(Editor's Note: Users are warned that this program does "execute" in the worst sense of the word.)

13AF 7640	PPSU	40	13D2 180E	BCTR,Z	13E2
13B1 7518	CPSL	18	13D4 50	RRR,R0	
13B3 0422	LODI,R0	22	13D5 50	RRR,R0	
13B5 24A5	EORI,R0	A5	13D6 50	RRR,R0	
13B7 D0	RRL,R0		13D7 3F02B4	BSTA,UN	02B4
13B8 C801	STRR,R0	13BB	13DA 0875	LODR,R0	13D1
13BA 1B73	BCTR,UN	13AF	13DC D800	BIRR,Z	13DE
13BC 6850	IORR,R0	13BE	13DE C871	STRR,R0	13D1
13BE 50	RRR,R0		13E0 1B6E	BCTR,UN	13D0
13BF 0A82	LODR,R2	*13C3	13E2 04C0	LODI,R0	C0
13C1 92	LPSU		13E4 CC040D	STRA,R0	040D
13C2 4A62	ANDR,R2	13A6	13E7 3B02	BSTR,UN	13EB
13C4 01	LODZ,R1		13E9 1B79	BCTR,UN	13E4
13C5 32	REDC,R2		13EB 0978	LODR,R1	13E5
13C6 7A7A	8SNR,N	13C2	13ED 0A77	LODR,R2	13E6
13C8 62	IORZ,R2		13EF DA02	BIRR,N	13F3
13C9 0900	LODR,R1	13CB	13F1 D900	BIRR,P	13F3
13CB 3F02DB	BSIA,UN	02DB	13F3 C970	STRR,R1	13E5
13CE 3B23	BSTR,UN	13F3	13F5 CA6F	STRR,R2	13E6
13D0 086A	LODR,R0	13BC	13F7 17	RETC,UN	