UNIVERSAL EPROM PROGRAMMER

Get to grips with the bits using our Universal EPROM programmer. Design and development by Mike Bedford.

EPROMs provide a convenient means of storing commonly used software, but many microcomputer users are deterred from using this media due to the difficulties of erasure and programming. EPROM programmers are both commercially available and have been featured as constructional articles in some magazines, but the former are relatively expensive and the latter tend to be limited as regards types of EPROM supported. For these reasons, it was considered appropriate to develop a universal EPROM programmer which could be built by the amateur for a modest sum. The programmer presented here will support the following EPROMs: 2758, 2716, 2516, 2732, 2732A, 2532, 2764, 2564, 27128 and 27256. This list includes every single supply version of the 27-Series and 25-Series devices up to and including 256k bit capacity, which seems likely to be the largest EPROM which will be produced as further development will probably be of EAROMs and EEPROMs. The programmer is intended for use in conjunction with a 6502 computer system and appropriate software is presented. Although the hardware is designed to support the 27256, this one EPROM is not as yet supported by the software due to lack of preliminary data on this device, the chip itself having not been released by the manufacturer at the time of writing. To complete the requirements for programming EPROMs, some hints are given on building an erasure unit.

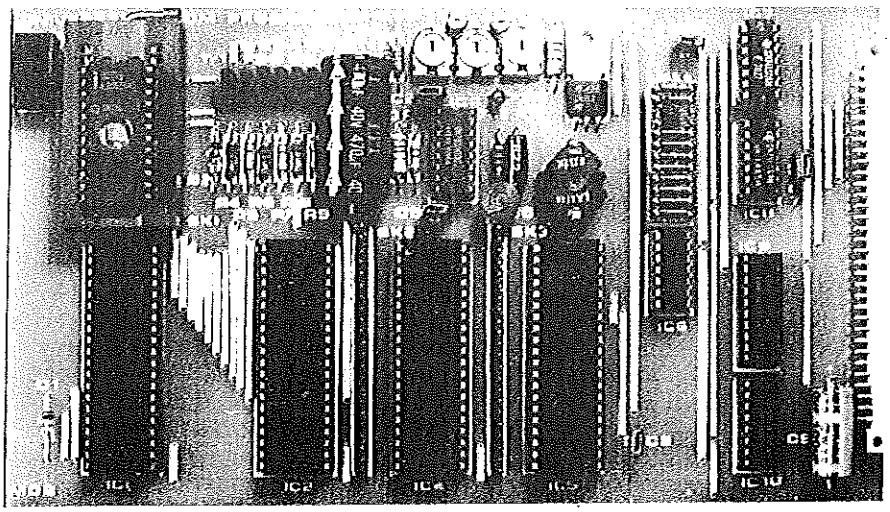
Hardware

The hardware has been designed for a Tangerine Microtan system, the physical dimensions of the circuit board being selected such that it will plug directly into the system rack. Non-Tangerine users should not be deterred, however, as electronically it should be quite simple to interface this card to any computer using the 6502 or 6809 processors. No special power supplies are required as all voltages required for programming are derived from a single +5v supply.

Although the PCB has sufficient space for a zero insertion force socket to be mounted on board, it is probably more convenient, especially if the card is to be rack mounted, to mount this socket in a separate console, making a connection with a length of ribbon cable. Note that a switch should be provided on the console to isolate V_{pp} and V_{cc} on pins 1 and 28 respectively. This is necessary since inserting or removing an EPROM with these supplies present may result in its destruction.

If the console is not used, switch SW1 should be mounted on board to isolate these supplies. On EPROMS which have V_{pp} and V_{cc} on other pins, these supplies may be isolated under program control. It will also be noticed that only a single zero insertion force socket has been provided on PCB. This decision was taken on grounds of economy and some constructors may prefer to add a 24-pin socket in addition. If a single 28-pin socket is used, 24-pin devices should be inserted into the bottom part of the socket, ie leaving pins 1, 2, 27 and 28 empty. Table 1 shows the pin outs of all EPROMs which are supported.

The programmer applies a TTL level to all pins with the exception of pin 1 which is connected to V_{pp}, pin 14 (0V) and pin 28 (+5v). V_{pp} is program selectable to +25 V, +21 V or +5 V and may also be applied to pins 22 and 23 as an alternative to a TTL level. To summarise this information, the programmer may be considered as a glorified 29-bit output port. This being the case, and since the circuitry required to implement the above would not completely fill a 8" x 4½" PCB, it seemed appropriate at the cost of only two more ICs to add four additional 8-bit output ports to make maximum use of the board space available. These ports are completely independent from the



The Universal EPROM Programmer. Note that this prototype differs slightly from the final version — in particular, the component numbering is completely different!

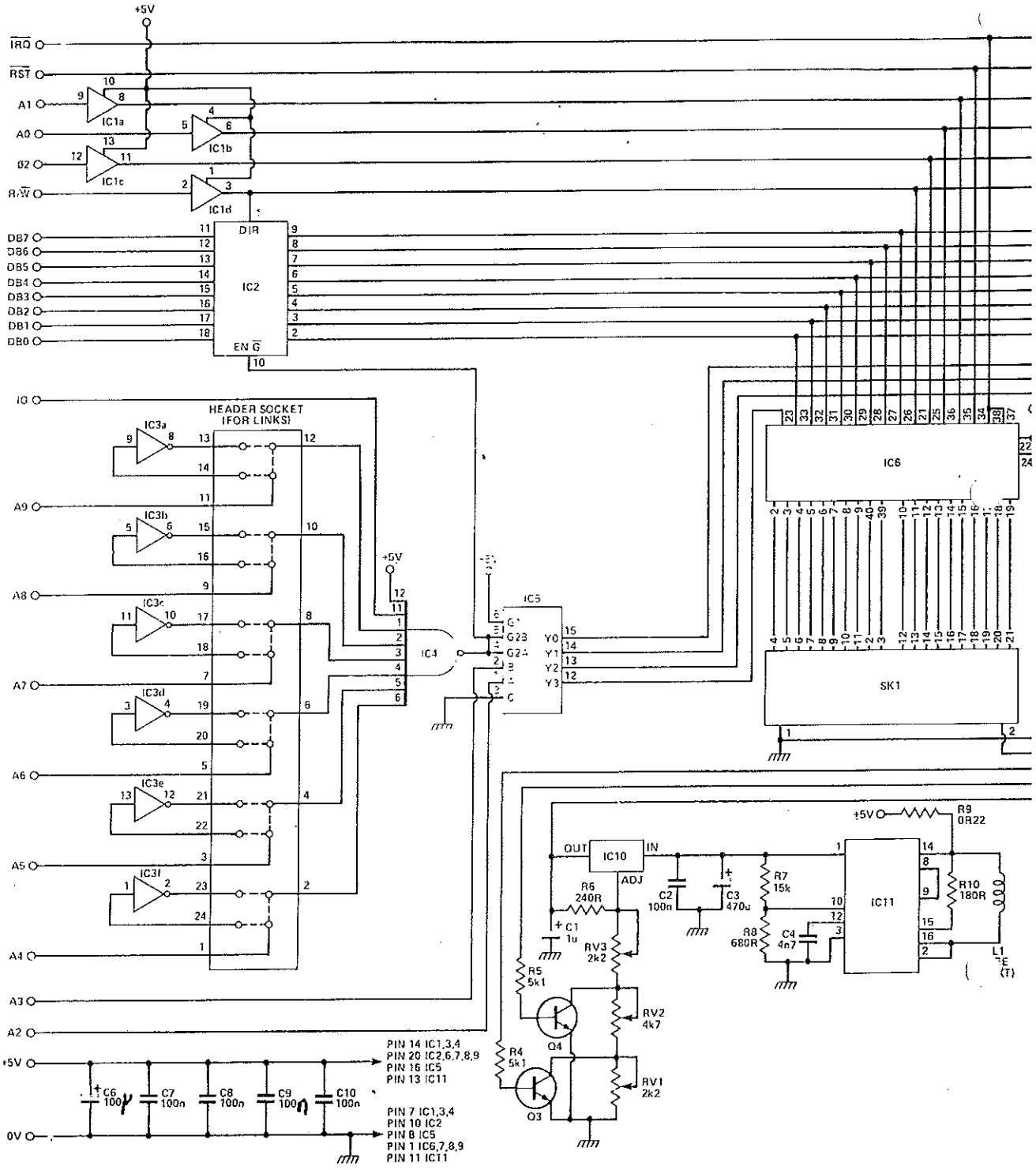


Fig. 1 Circuit diagram of the Universal EPROM Programmer.

programmer.

Software

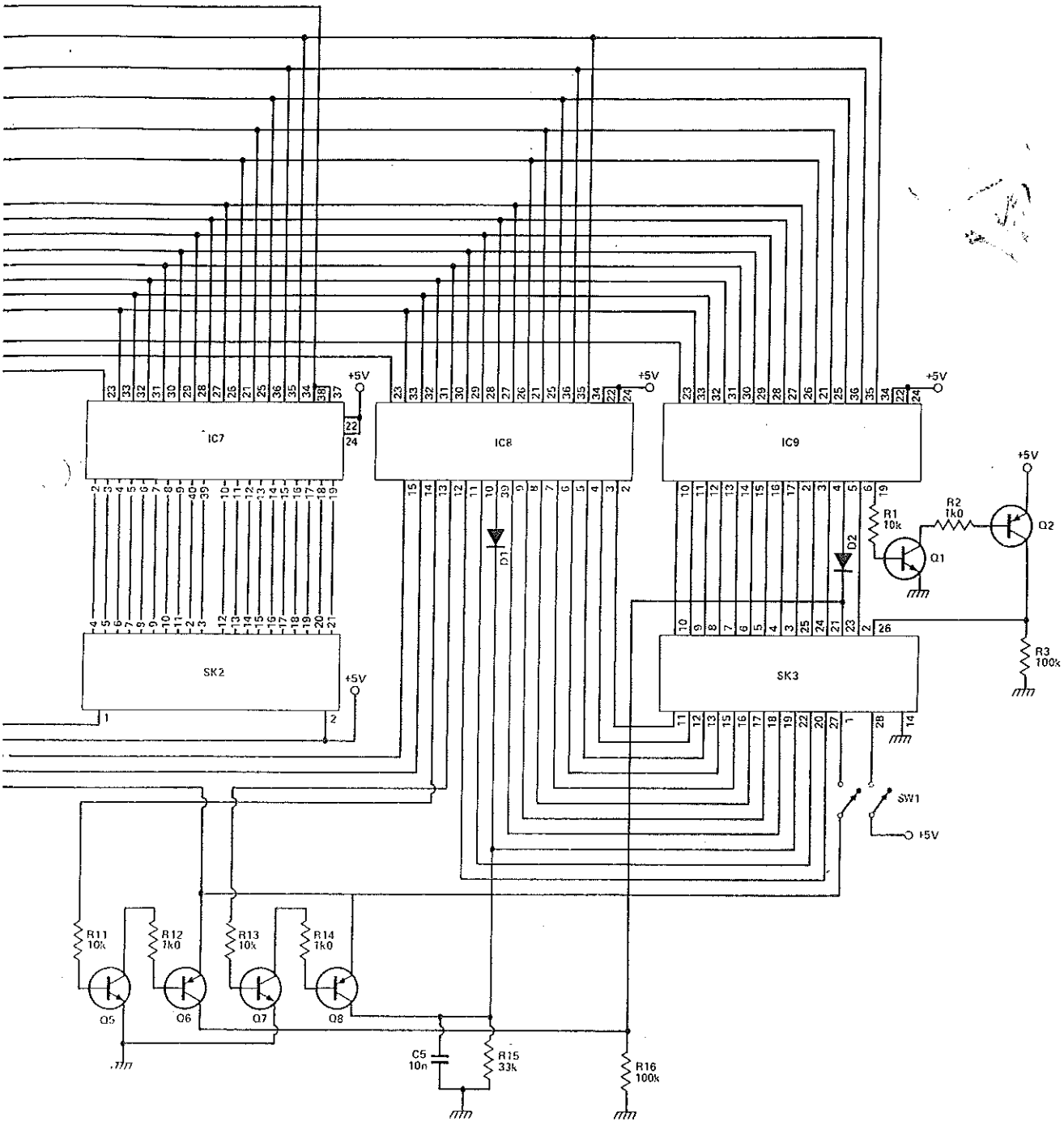
In order to be a useful development tool, the following functions are the minimum requirements of an EPROM programmer:

1. Read data from an EPROM into computer memory;
2. Compare the contents of an EPROM with the contents of computer memory;
3. Test an EPROM for erasure;
4. Program an EPROM from data in computer memory, verifying each byte as it is written.

In fact, some additional functions have also been added and these will be described under the section on using the EPROM programmer.

Table 2 is a mode selection table for all supported devices, showing the read, program, verify and standby modes, these being the modes necessary to implement the

The design is based around four 6821 20-bit PIOs. The circuitry comprising IC3, IC4 and IC5 provides the interface to the Tangerine bus and, in conjunction with the links, allows the board to be configured to occupy a 16-byte block within the 1k I/O area. IC1 and IC2 are used to buffer various signals in order that no more than one TTL load is presented to any bussed input. Of the four 6821s, IC6 and IC7 provide the four independent I/O ports. These being connected to the outside world via SK1 and SK2, whereas IC8 and IC9 are used to drive the EPROM programmer. Most of the signals required to drive the pro-

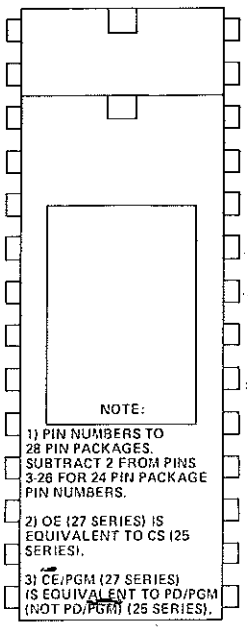


HOW IT WORKS

grammer are TTL levels and are taken directly from the two 6821s to the ZIF socket, SK3. Pin 26 on SK3 is slightly different in that although it is a pure TTL signal on all 28 pin devices, it is the V_{cc} supply on all EPROMs in 24-pin packages and hence requires a much greater current capacity than is available from a port on a 6821. Q1 and Q2 are therefore used to switch the +5 V supply to SK3 pin 26 under the control of IC9/CB2 (pin 19). A similar technique is used to switch V_{pp} onto SK3 pins 22 and 24 using transistor pairs Q5/Q6 and Q8 respectively. Since these two pins on SK3 are also required to present TTL levels under different conditions, they are also connected to 6821 ports, isolating these signals by use of Germanium diodes D1 and D2 respectively.

It should be noted that when a 6821 is required to drive a transistor, A 'B' port is used, these having a greater current sourcing capacity than 'A' ports, and when a port needs to be isolated by a diode, an 'A' port is used as these give a full +5 V high signal so that, even allowing for the voltage drop across the diode, a good TTL high is presented. The V_{pp} supply is generated from the +5 V supply using IC11, a 78S40 switching regulator IC, in connection with timing components L1 and C4. Although this component should be capable of regulating V_{pp} to within the required limits, experiments showed that it is advisable to select R7 and R8 such that IC11 would output about +30 V and use a separate LM317MP regulator to give the required voltage. Since V_{pp} may need to be +25 V, +21 V or +5 V, Q3 and Q4 are used, to switch out portions of the resistor chain between the regulator adjust terminal and 0 V hence altering the output voltage. These two transistors are connected to IC8 PB4 and PB5 (pins 14 and 15) hence allowing V_{pp} to be changed under program control.

27 256	27 128	25 64	27 64	25 32	27 32/ 32A	27/ 25 16	27 58	IC/ PIN
VPP	VPP	VPP	VPP					-/1
A12	A12	$\overline{CS1}$	A12					-/2
A7	A7	A7	A7	A7	A7	A7	A7	1/3
A6	A6	A6	A6	A6	A6	A6	A6	2/4
A5	A5	A5	A5	A5	A5	A5	A5	3/5
A4	A4	A4	A4	A4	A4	A4	A4	4/6
A3	A3	A3	A3	A3	A3	A3	A3	5/7
A2	A2	A2	A2	A2	A2	A2	A2	6/8
A1	A1	A1	A1	A1	A1	A1	A1	7/9
A0	A0	A0	A0	A0	A0	A0	A0	8/10
D0	D0	D0	D0	D0	D0	D0	D0	9/11
D1	D1	D1	D1	D1	D1	D1	D1	10/12
D2	D2	D2	D2	D2	D2	D2	D2	11/13
GND	GND	GND	GND	GND	GND	GND	GND	12/14



IC PIN	27 58	27/ 25 16	27/ 32 32A	25 32	27 64	25 64	27 128	27 256
-/28					VCC	VCC	VCC	VCC
-/27					$\overline{PGM}$	$\overline{CS2}$	$\overline{PGM}$	A14
24/26	VCC	VCC	VCC	VCC	N/C	N/C	A13	A13
23/25	A8	A8	A8	A8	A8	A8	A8	A8
22/24	A9	A9	A9	A9	A9	A9	A9	A9
21/23	VPP	VPP	A11	VPP	A11	A12	A11	A11
20/22	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$/ VPP	PD/ $\overline{PGM}$	$\overline{OE}$	PD/ $\overline{PGM}$	$\overline{OE}$	$\overline{OE}$
19/21	AR	A10	A10	A10	A10	A10	A10	A10
18/20	$\overline{CE}$/ PGM	$\overline{CE}$/ PGM	$\overline{CE}$	A11	$\overline{CE}$	A11	$\overline{CE}$	$\overline{CE}$
17/19	D7	D7	D7	D7	D7	D7	D7	D7
16/18	D6	D6	D6	D6	D6	D6	D6	D6
15/17	D5	D5	D5	D5	D5	D5	D5	D5
14/16	D4	D4	D4	D4	D4	D4	D4	D4
13/15	D3	D3	D3	D3	D3	D3	D3	D3

Table 1 Pin-outs of all the EPROMs our programmer will process.

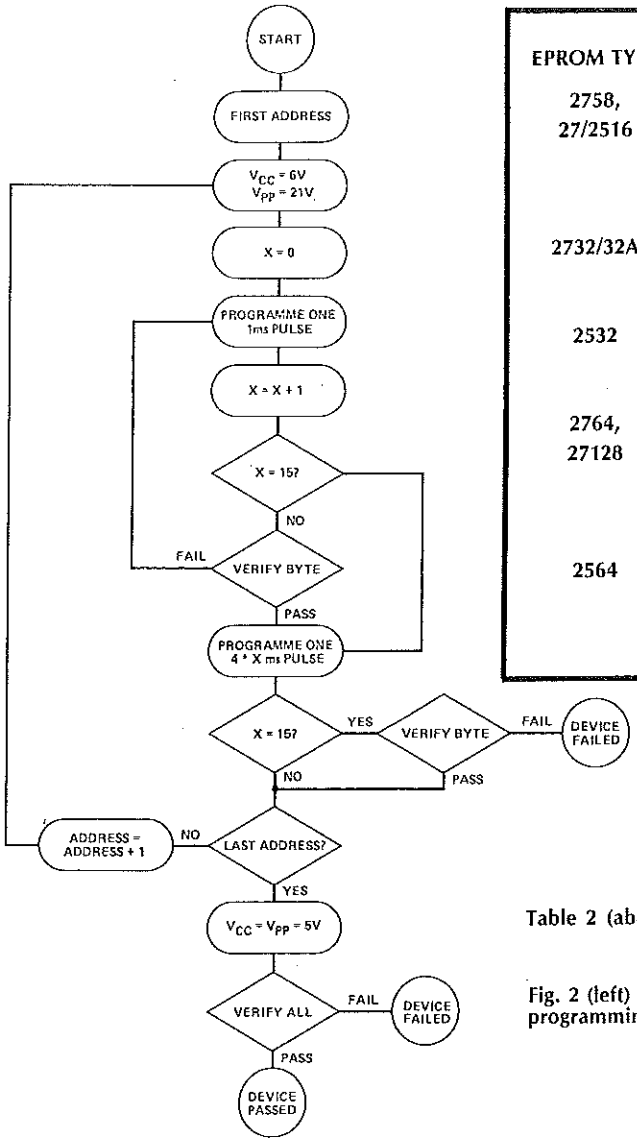


Fig. 2 (left) Flowchart for intelligent programming mode.

EPROM TYPE	FUNCTION	PIN	MODE READ	STANDBY	PROGRAMME	VERIFY
2758, 27/2516	*AR	19	VIL	VIL	VIL	VIL
	$\overline{CE}$/PGM	18	VIL	VIH	VIL→VIH	VIL
	$\overline{OE}$	20	VIL	X	VIH	VIL
	VPP	21	+5 V	+5 V	+25 V	+25 V
2732/32A	$\overline{CE}$	18	VIL	VIH	VIL	—
	$\overline{OE}$/VPP	20	VIL	X	**VPP	—
2532	PD/$\overline{PGM}$	20	VIL	VIH	VIH→VIL	—
	VPP	21	+5 V	+5 V	+25 V	—
2764, 27128	$\overline{CE}$	20	VIL	VIH	VIL	VIL
	$\overline{OE}$	22	VIL	X	X	VIL
	$\overline{PGM}$	27	VIH	X	VIH→VIL	VIH
	VPP	1	+5 V	+5 V	+21 V	+21 V
2564	PD/$\overline{PGM}$	22	VIL	VIH	VIH→VIL	—
	$\overline{CS1}$	21	VIL	X	VIL	—
	$\overline{CS2}$	27	VIL	X	VIL	—
	VPP	1	+5 V	+5 V	+25 V	—

NOTES:
X = DON'T CARE
* AR IS ONLY ON THE 2758
** VPP IS +25 V ON THE 2732, +21 V ON THE 2732A

Table 2 (above) Mode selection table.

functions mentioned. It may be noticed that some devices can perform both read and verify. Functionally, these modes are the same but verify is carried out with V_{pp} high, hence simplifying the program/verify process. From this table it may be seen that there is a great deal in common between the various EPROMs, hence simplifying the design of software.

To get the full picture, the timing diagrams of each device should be scrutinised, but for the general user this information is probably not relevant. To summarise, however, EPROMs generally require about 50 ms to program each byte giving total programming times of from around 50 seconds for a 2758 to 13 minutes for a 27256. You may consider these times rather long, especially for the larger devices and although not implemented in the software presented here, there is an alternative programming method referred to as the *intelligent programming mode* which can reduce programming times by a factor of six for the 2764, 27128 and 27256. Figure 2 is a flow diagram of the functions which need to be performed in order to implement this method of prorgramming. Notice that V_{cc} needs to be increased to +6V for this process to be carried out and that there is no provision in the hardware to select this value of V_{cc} under program control. This need present no great problem, however, to the user wishing to implement this mode, as +6v may be switchable from the programming console, perhaps by deriving this voltage from the system +12v supply.

Any additions to, or modifications of the software will need to be made in the light of the information presented in Tables 3 and 4. These may be described as a programmer's view of the EPROM programmer, Table 3 showing the address of each register and Table 4 indicating which bits of the various 6821 ports connect to which EPROM pins or perform the various control functions required.

Construction And Alignment

Although the circuit of the programmer is of sufficient complexity that if it were to be produced commercially it would probably be double sided, it is not so complex that a single sided board would be impossible to design. It was considered that cost would be of prime importance to an amateur building a one-off project, and on these grounds it was artworked as a single-sided board. As a result of this, a number of insulated wire links need to be inserted prior to fitting the components. No special instructions are required on the fitting of components with the exception of socket SK3 and switch SW1. If a separate programming console is to be used, then an ordinary low profile DIL socket should be used as SK3, and two wire links should replace SW1. On the other hand, if the programmer is intended to be self contained on a single board, a zero insertion force DIL socket should be used as SK3 and switch SW1 is needed. Once the construction is complete, the links need to be configured to place the board at the desired address. The offset from the start of the I/O area is 16 times the binary number represented by the links. The best way to illustrate exactly how the links are used is probably graphically; Fig. 3 shows a few examples.

The other part of the circuit which requires setting up is associated with V_{pp} generation. This is very important as EPROMs will be destroyed if V_{pp} is more than 0V5 too high. The best way to check this

Table 3 Register addresses.

FUNCTION	6821 NUMBER	REGISTER	OFFSET FROM BASE
	IC9	DDRA/ORA	00
	IC9	CRA	01
REGISTERS	IC9	DDRB/ORB	02
FOR	IC9	CRB	03
EPROM	IC8	DDRA/ORA	04
PROGRAMMER	IC8	CRA	05
	IC8	DDRB/ORB	06
	IC8	CRB	07
	IC7	DDRA/DRA	08
REGISTERS	IC7	CRA	09
FOR	IC7	DDRB/ORB	10
GENERAL	IC7	CRB	11
PURPOSE	IC6	DDRA/ORA	12
I/O	IC6	CRA	13
PORTS	IC6	DDRB/ORB	14
	IC6	CRB	15

Table 4 Port functions.

6821 NUMBER/ PORT	SK3 PIN NO.	PROGRAMMER FUNCTION
IC9/PB0	10	A0
IC9/PB1	9	A1
IC9/PB2	8	A2
IC9/PB3	7	A3
IC9/PB4	6	A4
IC9/PB5	5	A5
IC9/PB6	4	A6
IC9/PB7	3	A7
IC9/PA0	25	A8
IC9/PA1	24	A9
IC9/PA2	21	A10 (EXCEPT 2758), AR (2758)
IC9/PA3	23	A11 (2732/32A/64/128/256), A12 (2564)
IC9/PA4	2	A12 (2764/128/256), $\overline{CS1}$ (2564)
IC9/CB2	26	A13 (27128/256), VCC (2758/16/32/32A, 2516/32)
IC8/PA0	11	D0
IC8/PA1	12	D1
IC8/PA2	13	D2
IC8/PA3	15	D3
IC8/PA4	16	D4
IC8/PA5	17	D5
IC8/PA6	18	D6
IC8/PA7	19	D7
IC8/CA2	22	$\overline{OE}$ (2758/16/38/32A/64/128/256,2516), PD/$\overline{PGM}$ (2532/64)
IC8/PB0	20	A11 (2532/64), $\overline{CE}$/PGM (2758/16, 2516), $\overline{CE}$ (REMAINDER)
IC8/PB1	27	$\overline{PGM}$ (2764/128), $\overline{CS2}$ (2564), A14 (27256)
IC8/PB2	22	VPP (2732/32A)
IC8/PB3	23	VPP (2758/16, 2516/32)
IC8/PB4	—	VPP +5V SELECT*
IC8/PB5	—	VPP +21V SELECT*
		*NOT (VPP+5) AND NOT (VPP+21V) = +25V SELECT

is with a voltmeter probe on the output of IC2, with IC8 removed from its socket. Apply +5 V to pin 14 of the IC8 socket and adjust RV3 until +5 V is recorded on the test meter. Now replace +5 V on pin 14 by +5 V on pin 13 of IC8 socket and adjust RV2 for +21 V on the meter. Finally, remove +5 V from pin 13 of IC8 socket and adjust RV1 for a potential of +25 V at IC10 output. Setting up is now complete.

Using The Programmer

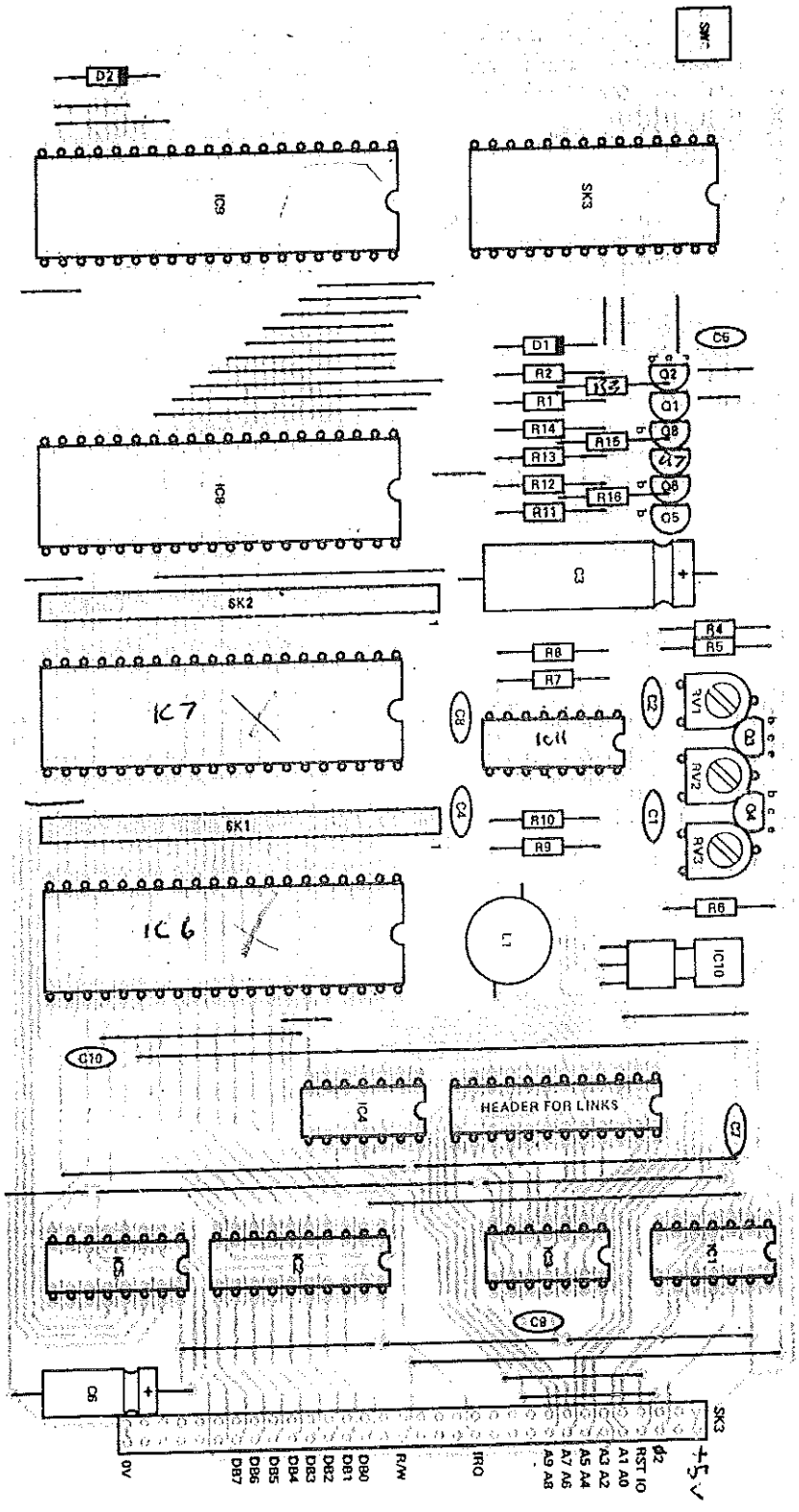
The first stage in using the EPROM programmer is to load and run the support package software which is written in BASIC and assembly code. The user will first be prompted for EPROM type, valid responses being 2758, 2716, 2516, 2732, 2732A, 2532, 2764, 2564 and 27128. A request for a base address will then be made and the user should respond with a four-figure hexadecimal number. The base address is the address relative to which all references to computer internal memory are made and this means that the user does not need to be concerned with the actual absolute addresses in computer memory. The reasons for being able to select the base address are two fold. Firstly, to fit in with the memory map of any computer on which the software may be run and, secondly, to allow more than one set of data to be maintained at one time by using different base addresses. A table showing all the commands available will the be printed before the *? prompt appears on the screen. When this prompt indicates that the program is waiting for input of a command, it is safe to insert or remove EPROMs from the ZIF socket, provided, of course, that the isolating switch is in the OFF position for 28 pin devices.

The following describes the function of the EPROM programmer support package commands. It should be noted that in each case, either the whole word or the initial letter may be used:-

(N)EW, (B)ASE These two commands cause the user to be prompted for a new EPROM type or a new base address respectively, hence allowing these two parameters to be altered without exiting and re-running the program.

(T)EST This performs a test on the EPROM, reporting whether or not it has been erased.

(R)EAD, (P)ROGRAMME, (V)ERIFY These commands cause the user to be prompted for start and finish addresses which should be entered as two four-figure hexadecimal numbers separated by a comma. These addresses define the portion of the EPROM to be used and also the portion of computer internal memory into which data will be written for a read operation or from which data will be read for program or verify operations. The computer addresses are offset from the base address. In program and verify, any discrepancies between EPROM and computer memory will be reported. Such discrepancies will be printed, one per line, stopping after every 16 lines. Pressing return at this point will return to the *? prompt, whereas pressing any other key will



Overlay diagram of the programmer.

PARTS LIST

RESISTORS (all ¼W 5% unless stated)

R1, 11, 13	10k
R2, 12, 14	1k0
R3, 16	100k
R4, 5	5k1
R6	240R
R7	15k
R8	680R
R9	0R22 wire wound
R10	180R, 1W
R15	33k
RV1, 3	2k2 min horizontal preset
RV2	4k7 min horizontal preset

CAPACITORS

C1	1u 35 V axial electrolytic
C2, 7, 8, 9, 10	100n ceramic
C3	470u 35 V axial electrolytic
C4	4n7 polyester
C5	10n ceramic
C6	100u 6V3 axial electrolytic
C10	10n ceramic

SEMICONDUCTORS

IC1	74LS126
IC2	74LS245
IC3	74LS04
IC4	74LS30
IC5	74LS138
IC6, 7, 8, 9	MC6821 (or similar)
IC10	LM317M
IC11	78S40
Q1, 3, 4, 5, 7	BC184L
Q2, 6, 8	BC214L
D1, 2	OA91

MISCELLANEOUS

L1	34 turns 24 swg wire on RM6 pot core (AL = 250)
SK1, 2	0.1" pitch 22-way male molex connectors (or shorter ones made up to required length)
SK3	28 pin zero insertion force socket (or ordinary socket is console is used, see text)
SW1	two pole single throw switch (omit if using console)

One 16-way and one 86-way DIL headers and sockets; edge connector (2 × 32-way A+B DIN Euro connector, male angled pins, for Tangerine) PCB, wire, etc.

continue the programming or verification.

(L)IST, (M)ODIFY These commands probably duplicate facilities available in the computer monitor but are included here to allow minimum changes to be made to computer memory or data to be checked without the need to exit from this package. LIST will request start and finish addresses and will list on the screen the addresses and data of the portion of memory requested, offset from the base address. The listing will stop after every 16 lines at which point pressing return will exist to the *? prompt, whereas pressing any other key will continue with the listing.

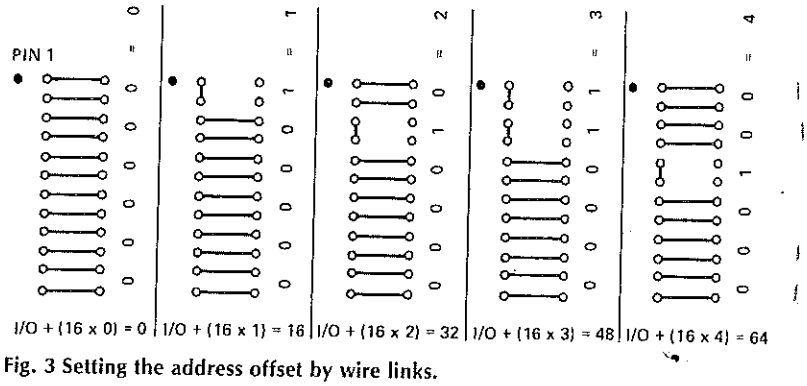


Fig. 3 Setting the address offset by wire links.

MODIFY will prompt for a single address, the contents of the base address offset by the value given being displayed on the screen. Entering an X at this point will return to the *? prompt leaving the data in that location unchanged, whereas entering a two-figure hexadecimal number will cause the relevant address to be updated with the data entered.

(H)ELP lists all the commands available and reminds the user of the currently selected EPROM type and base address.

(E)XIT Causes the program to terminate.

In the above commands, any wrongly entered information will result in self-explanatory error messages. The one error message which, perhaps, requires a word of explanation is TYPE/RANGE INCOMPATIBLE. This error is a result of entering a start and finish address to any command which defines a range larger than the capacity of the selected EPROM type.

Erasing EPROMs

EPROMs are erased by exposure to ultra-violet light through the transparent window on the top of the package. Small commercially available erasing units with capacities of up to six chips cost typically in the £40-£50 region. If the requirements for erasing EPROMs are considered, it becomes obvious that an erasing unit can be constructed for considerably less than the price of commercial equipment. The requirements stated by EPROM manufacturers to erase such a device are a 20 to 30 minute exposure of 2357 A (253.7 nm) wavelength ultra-violet light at an intensity of 12000 uW/cm^2. The Philips TUV 15 W tube emits UV at the required wavelength and at an intensity of 37 uW/cm^2 at about one inch from the tube, the distance metre from the tube, which corresponds to about 12000 uW/cm2 at a distance of one inch, as used in most EPROM erasers.

The tube costs in the region of £10 and will fit into an ordinary 15" fluorescent light fitting, it provides the basis for a relatively inexpensive unit which could accommodate about ten ICs.

A few words of caution are appropriate at this point. Ultra-violet radiation, and in particular shortwave UV as emitted by the TUV 15 W, is harmful to both the eyes and the skin. It is therefore essential to build the tube into a light-tight cabinet, ideally with a micro switch fitted under the lid to isolate the supply when opened, which will prevent UV light from coming into contact with skin or eyes.

EPROMs may also be erased by UV of longer wavelength (3000 A - 4000 A) although longer exposure times will probably be required. Since 'black light' tubes of the type used for disco lighting emit at about 3500 A and are more easily available than short-wave UV tubes, it may be worthwhile experimenting with this type of light source.

One final point on the topic of erasing EPROMs is that both sunlight and ordinary fluorescent tubes emit some radiation in the 3000 A - 4000 A region with the result that prolonged exposure to these light sources will result in erasure. For this reason it is recommended that an opaque adhesive label is used to cover the windows of programmed EPROMs.

Next month, we'll describe the software for the unit. **ETI**

BUYLINES

There should be relatively few problems in obtaining components for this project. The 0.22R resistor can be found in several suppliers' lists, including Watford's. The RM6 potcore is available from RS Components.
our PCB service — see page 77 for details.

UNIVERSAL EPROM PROGRAMMER

To use our Universal EPROM programmer, you've got to have the software to drive it. Mike Bedford fills us in on what's needed.

The logical choice of programming language for a software package which is required to perform critical timing and which contains large frequently peated loops, is assembler. On the other hand, the obvious choice of language for a package which is intended to run on a variety of different personal computers is BASIC. The software presented here is a compromise between the two: a BASIC program which performs the I/O but which calls an assembler subroutine for the time critical or time consuming tasks.

The assembler routine starts at address 1C00, but this may need to be relocated in order to fit in with the memory map of some systems. If this routine is relocated, the variable MC on line 290 of the BASIC program will have to be changed to the decimal start address of the routine. Another portion of the BASIC program which may require tailoring to a particular system is line 310. The variable PA on this line contains the start address of the EPROM programmer hardware as selected by the links on the board. This address should also be updated in the assembler subroutine on line 23 which equates IC9PIA to the start address.

Microtan 65 BASIC uses the statement I=USR(X) to call a machine code subroutine, having first POKE'd the low order byte of the M/C address to 34 and having POKE'd the high order byte to 35. This is done on lines 4040-4060, 5030-5050, 6040-6060 and 7040-7060 of the BASIC program and may require modification on other machines.

Finally, the programming timing loop in the assembler routine assumes a processor clock frequency of the 750KHz as used on the Microtan. The value loaded into register Y on line 143 of the routine will have to be modified accordingly for other clock speeds (use hexadecimal 27 for 1 MHz).

As far as entering the program is concerned, the main BASIC program is rather long and it would be advisable to enter it in relatively small portions, saving it after the addition of each new section. This suggestion is made for two reasons: firstly it is difficult to concentrate for sufficiently long to enter the whole program at once without making errors; and, secondly it would be extremely frustrating if the computer were to crash for some reason after having typed in over 200 lines of code!

The assembler listing is rather long, and will only be of interest to readers wishing to modify the software. For this reason we haven't reproduced it here, but a copy may be obtained by sending a large stamped addressed envelope (or international reply coupon) to the ETI office — please mark the outer envelope "PROGRAMMER LISTING". Most users will find it easiest to enter the hex code directly.

Once the program and subroutine have been entered and recorded on cassette, it will be worthwhile investing some time carefully checking through the program. It is quite possible that a mistake may cause more than the appearance of the all too familiar SYNTAX ERROR on the screen: an error in the software could easily turn an EPROM programmer into an EPROM destroyer!

Sample Run

On page 39 is a reproduction of

1C00	4C	4C	1C	00	00	00	00	00	00	00	00	00	00	3C	3C	3C
1C10	3C	3C	3C	34	34	34	18	18	18	10	10	18	12	10	12	3C
1C20	3C	3C	34	34	3C	3C	3C	3C	08	08	08	05	25	08	22	00
1C30	22	01	01	01	01	01	08	02	08	02	06	06	06	06	06	05
1C40	06	05	06	01	01	01	01	01	00	01	00	01	20	87	1D	AD
1C50	0B	1C	C9	02	D0	03	4C	BC	1C	A9	30	8D	25	BC	A9	00
1C60	8D	24	BC	A9	34	8D	25	BC	AE	0C	1C	BD	0D	1C	8D	23
1C70	BC	A9	00	8D	20	BC	A9	3C	8D	25	BC	BD	16	1C	8D	26
1C80	BC	20	9C	1D	20	62	1D	D0	03	4C	13	1D	20	17	1D	AD
1C90	25	BC	49	08	8D	25	BC	20	9C	1D	AD	24	BC	8D	0A	1C
1CA0	A2	00	AC	0B	1C	F0	10	30	07	C9	FF	F0	BB	4C	13	1D
1CB0	C1	35	F0	B4	4C	13	1D	81	35	4C	68	1C	A9	30	8D	25
1CC0	BC	A9	FF	8D	24	BC	A9	34	8D	25	BC	AE	0C	1C	BD	0D
1CD0	1C	8D	23	BC	A9	00	8D	20	BC	BD	1F	1C	8D	25	BC	BD
1CE0	28	1C	8D	26	BC	20	9C	1D	20	62	1D	F0	26	20	17	1D
1CF0	A2	00	A1	35	8D	24	BC	AE	0C	1C	BC	3A	1C	B9	20	BC
1D00	5D	31	1C	99	20	BC	A0	1D	A2	FF	CA	D0	FD	88	D0	FA
1D10	4C	CB	1C	20	87	1D	60	AD	05	1C	8D	22	BC	BD	43	1C
1D20	F0	09	AD	06	1C	8D	20	BC	4C	52	1D	AD	20	BC	0D	09
1D30	1C	8D	20	BC	AD	06	1C	29	10	F0	08	AD	20	BC	09	08
1D40	8D	20	BC	AD	06	1C	29	08	F0	08	AD	26	BC	09	01	8D
1D50	26	BC	AD	06	1C	29	20	F0	08	AD	23	BC	09	08	8D	23
1D60	BC	60	E6	35	D0	02	E6	36	EE	05	1C	D0	03	EE	06	1C
1D70	AD	06	1C	29	E7	8D	09	1C	AD	05	1C	CD	07	1C	D0	06
1D80	AD	06	1C	CD	08	1C	60	A6	35	A4	36	AD	03	1C	85	35
1D90	AD	04	1C	85	36	8E	03	1C	8C	04	1C	60	A0	80	88	D0
1DA0	FD	60														

Fig. 1 Hex dump of the machine code — see text for details of how to obtain the assembler listing, should you need it.

```
LIST

 10 REM...EPROM SUPPORT PACKAGE
 20 REM...M D BEDFORD MARCH 83
 30 DIM C$(9)
 40 DATA "NEW","BASE","HELP","READ"
 50 DATA "PROGRAMME","VERIFY","TEST"
 60 DATA "LIST","MODIFY"
 70 FOR N=1 TO 9
 80 READ C$(N)
 90 NEXT
100 DIM H$(15)
110 DATA "0123456789ABCDEF"
120 READ HH$
130 FOR N=0 TO 15
140 H$(N) = MID$(HH$,N+1,1)
150 NEXT N
160 DIM T$(9)
170 DATA "2758","2716","2516","2732"
180 DATA "2732A","2532","2764","2564"
190 DATA "27128"
200 FOR N=1 TO 9
210 READ T$(N)
220 NEXT N
230 DIM S(9)
235 DATA 1024,2048,2848,4096,4096
240 DATA 4096,8192,8192,16384
250 FOR N=1 TO 9
260 READ S(N)
270 NEXT
280 REM...START ADDRESS OF M/C ROUTINES
290 MC=7168
300 REM...START ADDRESS OF PROGRAMMER
310 PA=48168
320 HR = INT(MC/256)
330 LR = MC-256*HR
340 POKE PA+1,49
370 POKE PA,255
380 POKE PA+3,48
390 POKE PA+2,255
400 POKE PA+7,48
410 POKE PA+6,255
700 PRINT CHR$(12)
710 PRINT "EPROM PROGRAMMER SUPPORT PACKAGE"
720 PRINT:PRINT
730 GOSUB 1000
740 GOSUB 2000
750 GOSUB 3000
755 GOSUB 13000
760 PRINT
770 INPUT "X";A$
780 IF A$="E" OR A$="EXIT" THEN STOP
790 FOR N=1 TO 9
800 IF A$=C$(N) OR A$=LEFT$(C$(N),1) THEN 820
810 NEXT N
820 ON N GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9000,900
830 PRINT
840 GOTO 770
900 REM...NO SUCH COMMAND
910 PRINT "COMMAND INVALID"
920 RETURN
1000 REM...(N)EW
1010 INPUT "EPROM TYPE";A$
1020 FOR EN=1 TO 9
1030 IF A$=T$(EN) THEN 1065
1040 NEXT EN
1050 PRINT "TYPE INVAILD"
1060 GOTO 1010
1065 POKE MC+12,EN-1
1070 RETURN
2000 REM...(B)ASE
2010 INPUT "BASE ADDRESS";A$
2020 GOSUB 10000
2030 IF A=99999 THEN 2010
2040 BA=A:BA$=A$
2050 RETURN
3000 REM...(H)ELP
3010 PRINT:PRINT
3020 PRINT "COMMANDS AVAILABLE ":PRINT
3030 PRINT "(N)EW";TAB(16);"(B)ASE"
3040 PRINT "(R)EAD";TAB(16);"(T)EST"
3050 PRINT "(P)ROGRAMME";TAB(16);"(V)ERIFY"
3060 PRINT "(L)IST";TAB(16);"(M)ODIFY"
3070 PRINT "(H)ELP";TAB(16);"(E)XIT"
3080 PRINT
3090 PRINT "TYPE = ";T$(EN);TAB(16);"BASE = ";BA$
3100 RETURN
4000 REM...(R)EAD
4010 POKE MC+11,0
4020 GOSUB 11000
4030 GOSUB 14000
4040 POKE 34,LR
4050 POKE 35,HR
4060 I=USR(X)
4070 GOSUB 13000
4080 RETURN
5000 REM...(P)ROGRAMME
5005 POKE MC+11,2
5010 GOSUB 11000
5020 GOSUB 14000
5030 POKE 34,LR
5040 POKE 35,HR
5050 I=USR(X)
5060 GOSUB 6015
5070 GOSUB 13000
5080 RETURN
6000 REM...(V)ERIFY
6010 GOSUB 11000
6015 LN=-1
6020 POKE MC+11,128
6030 GOSUB 14000
6040 POKE 34,LR
6050 POKE 35,HR
6060 I=USR(X)
6070 B=256*PEEK(MC+6)+PEEK(MC+5)
6080 IF B>FA THEN 6238
6090 A=PEEK(BA+B)
6100 GOSUB 12000
6110 B$=A$
6120 A=B
6130 GOSUB 12000
6140 C$=A$
6150 A=PEEK(MC+10)
6155 GOSUB 12000
6160 LN=LN+1
6170 IF LN<15 THEN 6210
6180 GET Z$
6190 IF ASC(Z$) = 13 THEN 6230
6200 LN=0:PRINT
6210 PRINT C$;"   EPROM = ";MID$(A$,3,2);"   MEMORY = ";MID$(B$,3,2)
6220 GOTO 6040
6230 GOSUB 13000
6240 RETURN
7000 REM...(T)EST
7010 POKE MC+11,1
7020 SA=0:FA=S(EN)-1
7030 GOSUB 14000
7040 POKE 34,LR
7050 POKE 35,HR
7060 I=USR(X)
7070 IF 256*PEEK(MC+6)+PEEK(MC+5)>FA THEN 7100
7080 PRINT "EPROM NOT ERASED"
7090 GOTO 7110
7100 PRINT "EPROM ERASED"
7110 GOSUB 13000
7120 RETURN
8000 REM...(L)IST
8010 GOSUB 11000
8020 PRINT
8030 LN=-1
8040 FOR N=BA+SA TO BA+FA
8050 LN=LN+1
8060 IF LN<15 THEN 8100
8070 GET A$
8080 IF ASC(A$) = 13 THEN 8170
8090 LN=0:PRINT
8100 A=N-BA
8110 GOSUB 12000
8120 B$=A$
8130 A=PEEK(N)
8132 Z$=" "
8134 IF A>32 AND A<128 THEN Z$=CHR$(A)
8140 GOSUB 12000
8150 PRINT B$;TAB(7);MID$(A$,3,2);TAB(12);Z$
8160 NEXT N
8170 RETURN
9000 REM...(M)ODIFY
9010 INPUT "ADDRESS";A$
9020 GOSUB 10000
9025 IF A=99999 THEN 9010
9030 B=A
9040 A=PEEK(A+BA)
9050 GOSUB 12000
9060 PRINT "VALUE = ";MID$(A$,3,2);TAB(16);"NEW VALUE = ";
9070 INPUT A$
9075 IF A$="X" THEN 9110
9080 A$="0"+A$
9090 GOSUB 10000
9095 IF A=99999 THEN 9060
9100 POKE B+BA,A
9110 RETURN
10000 REM...HEX DECODE
10010 IF LEN(A$)<>4 THEN 10070
10020 A=0
10030 FOR N=1 TO 4
10040 FOR I=0 TO 15
10050 IF H$(I) = MID$(A$,N,1) THEN 10080
10060 NEXT I
10070 A=99999:GOTO 10110
10080 A=A+I*16^(4-N)
10090 NEXT N
10100 GOTO 10120
10110 PRINT "INVALID"
10120 RETURN
11000 REM...START, FINISH ADDRESSES
11010 INPUT "START, FINISH ADDRESSES";A$,B$
11020 GOSUB 10000
11025 IF A=99999 THEN 11010
11030 SA=A
11040 A$=B$
11050 GOSUB 10000
11055 IF A=99999 THEN 11010
11060 FA=A
11070 IF SA<=FA THEN 11100
11080 PRINT "RANGE INVALID"
11090 GOTO 11010
11100 IF FA-SA+.99 <=S(EN) THEN 11130
11110 PRINT "RANGE/TYPE INCOMPATIBLE"
11120 GOTO 11010
11130 RETURN
12000 REM...HEX ENCODE
12010 A$=""
12015 JJ=16*16*16
12020 FOR J=3 TO 0 STEP -1
12030 I=INT(A/JJ)
12040 A=A-I*JJ
12050 A$=A$+H$(I)
12055 JJ=JJ/16
12060 NEXT J
12070 RETURN
13000 REM...ZEROISE PIA'S
13002 POKE PA+7,52
13004 POKE PA+6,48
13010 POKE PA+1,52
13020 POKE PA,0
13050 POKE PA+5,48
13060 POKE PA+4,255
13070 POKE PA+5,52
13080 POKE PA+4,0
13090 POKE PA+3,52
13100 POKE PA+2,0
13110 RETURN
14000 REM...M/C PARAMETERS
14010 IF SA>0 THEN 14030
14020 HI=255:LO=255:GOTO 14050
14030 HI=INT((SA-1)/256)
14040 LO=SA-1-256*HI
14050 POKE MC+5,LO
14060 POKE MC+6,HI
14062 IF SA+BA>0 THEN 14070
14064 HI=255:LO=255:GOTO 14090
14070 HI=INT((SA+BA-1)/256)
14080 LO=SA+BA-1-256*HI
14090 POKE MC+3,LO
14100 POKE MC+4,HI
14110 HI=INT((FA+1)/256)
14120 LO=FA+1-256*HI
14130 POKE MC+7,LO
14140 POKE MC+8,HI
14150 RETURN
15000 END
OK
```

Fig. 2 The main BASIC program.

a printout obtained by running he EPROM programmer support package on a Tangerine Microtan system. Note that a base address of 2000 has been selected — this being the lowest reasonable-size area of RAM on the system, the BASIC program occupying about 6K and the machine code routine being located at 1C00.

In answer to the question about EPROM type, a response of 2716 was given. A 2716 EPROM was inserted into the ZIF socket when the first *? prompt was printed and this was tested for erasure using the (T)EST command. The program indicated that the device was not erased. At this point, the entire contents of the 2716 (0000—07FF) was transferred into computer memory using the (R)EAD command before listing the contents of a portion of this data by use of the (L)IST command. The (M)ODIFY command was then used to modify location 0007 before attempting to re-programme this single byte in the EPROM using the (P)ROGRAMME command. It will be noticed that this was unsuccessful, a fact indicated by the verification message. This should come as no surprise in view of the fact that an attempt to re-programme an un-erased device had been made and that programming can only set high bits low (ultra-violet erasure being required to set low bits high).

At this point the 2716 was replaced by a 2732 and the programmer was instructed of this change by use of the (N)EW command. Its entire contents (0000—0FFF) were read into memory and listed as before, by use of the (E)XIT command. Note that the (L)IST command gives both the hexadecimal value of each byte and, where appropriate, the ASCII symbol.

```
EPROM PROGRAMMER SUPPORT PACKAGE


EPROM TYPE? 2716
BASE ADDRESS? 2000


COMMANDS AVAILABLE :

(N)EW           (B)ASE
(R)EAD          (T)EST
(P)ROGRAMME     (V)ERIFY
(L)IST          (M)ODIFY
(H)ELP          (E)XIT

TYPE = 2716     BASE = 2000

*? T
EPROM NOT ERASED

*? R
START, FINISH ADDRESSES? 0000,07FF

*? L
START, FINISH ADDRESSES? 0000,0008

0000   12
0001   13
0002   14
0003   15
0004   16
0005   17
0006   18
0007   19
0008   1A

*? M
ADDRESS? 0007
VALUE = 19      NEW VALUE = ? AF

*? P
START, FINISH ADDRESSES? 0007,0007
0007   EPROM = 09   MEMORY = AF

*? N
EPROM TYPE? 2732

*? R
START, FINISH ADDRESSES? 0000,0FFF

*? L
START, FINISH ADDRESSES? 0200,0210

0200   E9
0201   00
0202   8E
0203   52   R
0204   26   &
0205   48   H
0206   89
0207   14
0208   26   &
0209   50   P
020A   89
020B   16
020C   26   &
020D   40   @
020E   89

020F   10
0210   03

*? E

 BREAK IN  780
OK
```

ETI

UNIVERSAL EPROM PROGRAMMER Revisited

We really started something with our Universal EPROM Programmer in the August and September issues. Requests for the assembler listing and for advice on how to modify the program to run on other machines continue to pour in. In an attempt to stem the tide, Mike Bedford offers a few further thoughts and the listing itself.

As mentioned in the 2nd part of the Universal EPROM Programmer article, the software has been written in such a way as to make it compatible with all 6502 based computers with the absolute minimum of changes. However, it is inevitable that some slight modifications will be required when transferring software, however similar the two machines may be. A large proportion of the software package is written in BASIC which, being a high level language, means that it should be possible to transfer it directly to another machine without any changes at all. This isn't quite the case as there are a number of dialects of BASIC, generally differing in those areas which are extensions to the original BASIC. Possible areas in which the BASIC program might need modifying have already been mentioned and it is now appropriate to indicate how the assembler subroutine may be modified to run on different machines.

Since all the keyboard and display I/O is carried out in the BASIC program hence camouflaging any hardware differences in these areas, the only changes which may be required to the assembler routine are due to differences in the memory maps and the clock speeds of the various machines. Apart from users wishing simply to transfer the software to another machine, others may be interested in enhancing the package to carry out additional functions, for example, implementing the intelligent programming algorithm or adding new devices such as the 27256, 68732 and 68764 EPROMS which the hardware could support. It is therefore quite likely that a number of readers will require a much greater insight into the workings of the assembler routine than could be glimpsed from the hexadecimal dump given in the second part of the article. For this reason it is the intention here to reproduce a full assembler listing together with a simple "guided tour" to give some idea of how it works, hence enabling changes to be carried out without too much of the anguish often associated with trying to understand someone else's assembler program.

A few comments about the assembler are appropriate before going on to describe the structure of the program. Although all assemblers carry out essentially the same task there are often slight differences in the syntax. This program was written on a PDP-11 computer using a cross-assembler:

a). The .PROCESSOR directive on line 13 is used to inform the assembler of which processor is in use. For 6502 only assemblers this line should be omitted.

b). The .ORG directive on line 14 sets the start address of the code. Syntax on other assemblers may be ★=IC00H or .=IC00H.

c). This assembler uses a suffix of H to indicate a hexadecimal number. Some other assemblers use a prefix of $. Ie IC00H may need replacing by $IC00.

d). In this assembler the .DEFINE directive is used to equate values to literals. In many other assemblers the word .DEFINE should be omitted to leave an equation (eg IC1PIA = OBC20H) and in others the EQU directive is used.

e). The .BYTE directive reserves a byte of memory and assigns an initial value to it. The more common version of this directive is DFB or DEFB.

The assembler syntax having been clarified, we can now go on to investigate the workings of the assembler subroutine and we shall make a start by looking at the parameter storage and data areas.

LOADR and HIADR are initialised by the BASIC program to one less than the first RAM address to be accessed. On exiting from the subroutine these locations will contain one more than the last address accessed. LOADE and HIADE are a similar address pair for the EPROM address. LOADF and HIADF are an address pair which are set by the BASIC program to one more than the last RAM address which should be used. RDATA is used to pass the

value of the data read from the EPROM back to the calling program, this being required in verify mode. MODE is set by the BASIC program to indicate whether read, verify, test or program function is required. Finally TYPE is set up by the main program to indicate the type of EPROM in use and will contain a number in the range 0 for 2758 to 8 for 27128.

As we now start to consider the data tables we shall be directing our attention to lines 47 to 64 of the program. In each of the seven lists here, there are 9 values of which one will be selected according to the value of TYPE. READS3 and READS6 contain the initial values of IC1CRB and IC2PIB respectively prior to which toggling bit 3 of IC2CRA would cause a read of the EPROM. PROGS5 and PROGS6 are similar tables of data for setting initial values of IC2CRA and IC2PIB prior to programming. PTBIT and PTBYTE indicate which bit of which PIA register requires to be toggled in order to carry out the programming of an EPROM. As a matter of interest, the reason that there are not tables of initial conditions for program and read for all PIA registers which connect to control pins on the EPROM is that some are set to the same initial value independently of EPROM type and some are set to the same value for both read and program. Finally A11A12 indicates whether the particular EPROM has A11 on pin 23 and A12 on Pin 2 or A11 on pin 20 and A12 on pin 23, these being the two options for the 27-series and 25-series devices.

Having described the parameter storage and data areas it should be relatively straightforward to see how the assembler routine works. The value of MODE is used to select one of the two main routines either READ for reading, verifying or testing or PROG for programming. In either of these two

```
1                       ;*************************************************
2                       ;*                                               *
3                       ;* EPROM PROGRAMMER SUPPORT PACKAGE              *
4                       ;* READ/PROGRAMME ROUTINE                        *
5                       ;* M. D. BEDFORD    MARCH 1983                   *
6                       ;* THIS ROUTINE IS CALLED FROM A MAIN BASIC PROGRAM *
7                       ;* AND WILL SUPPORT 2758, 2716, 2516, 2732, 2732A *
8                       ;* 2532, 2764, 2564 AND 27128 DEVICES.           *
9                       ;*                                               *
10                      ;*************************************************
11                      ;
12                      ;
13                              .PROCESSOR      R6500
14                              .ORG            1C00H
15                      ;
16                      ;ZERO PAGE LOCATIONS USED
17                      ;
18                              .DEFINE         ZPLOAD=35H
19                              .DEFINE         ZPHIAD=36H
20                      ;
21                      ;ADDRESSES OF 6821 REGISTERS USED
22                      ;
23                              .DEFINE         IC1PIA=0BC20H
24                              .DEFINE         IC1CRA=IC1PIA+1
25                              .DEFINE         IC1PIB=IC1PIA+2
26                              .DEFINE         IC1CRB=IC1PIA+3
27                              .DEFINE         IC2PIA=IC1PIA+4
28                              .DEFINE         IC2CRA=IC1PIA+5
29                              .DEFINE         IC2PIB=IC1PIA+6
30                              .DEFINE         IC2CRB=IC1PIA+7
31                      ;
32                      ;
33  1C00  4C4C1C                JMP     START
34                      ;
35                      ;
36  1C03  00            LOADR:  .BYTE   0       ;LO RAM START ADDRESS
37  1C04  00            HIADR:  .BYTE   0       ;HI RAM START ADDRESS
38  1C05  00            LOADE:  .BYTE   0       ;LO EPROM START ADDRESS
39  1C06  00            HIADE:  .BYTE   0       ;HI EPROM START ADDRESS
40  1C07  00            LOADF:  .BYTE   0       ;LO EPROM FINISH ADDRESS
41  1C08  00            HIADF:  .BYTE   0       ;HI EPROM FINISH ADDRESS
42  1C09  00            HIADM:  .BYTE   0       ;HI ADDRESS EPROM MASKED
43  1C0A  00            RDATA:  .BYTE   0       ;DATA READ FROM EPROM
44  1C0B  00            MODE:   .BYTE   0       ;0-R,FFH-V,80H-T,2-P
45  1C0C  00            TYPE:   .BYTE   0       ;0-2758 ... 8-27128
46                      ;
47                      ;INITIAL CONDITIONS FOR READ - IC1CRB,IC2PIB
48                      ;
49  1C0D  3C            READS3: .BYTE   3CH,3CH,3CH,3CH,3CH,3CH,34H,34H,34H
50  1C16  18            READS6: .BYTE   18H,18H,18H,10H,10H,18H,12H,10H,12H
51                      ;
52                      ;INITIAL CONDITIONS FOR PROG - IC2CRA,IC2PIB
53                      ;
54  1C1F  3C            PROGS5: .BYTE   3CH,3CH,3CH,34H,34H,3CH,3CH,3CH,3CH
55  1C28  08            PROGS6: .BYTE   8,8,8,5,25H,8,22H,0,22H
56                      ;
57                      ;BITS OF BYTES TO TOGGLE FOR PROGRAMME
58                      ;
59  1C31  01            PTBIT:  .BYTE   1,1,1,1,1,8,2,8,2
60  1C3A  06            PTBYTE: .BYTE   6,6,6,6,6,5,6,5,6
61                      ;
62                      ;PIN POSN OF A11/A12    1 : A11-PIN 23, A12-PIN 2
63                                              0 : A11-PIN 20, A12-PIN 23
64  1C43  01            A11A12: .BYTE   1,1,1,1,1,0,1,0,1
65                      ;
66                      ;
67  1C4C  20871D        START:  JSR     ZPSWAP  ;SWAP RAM ADDR/ ZERO PGE
68  1C4F  AD081C                LDA     MODE    ;
69  1C52  C902                  CMP     £2      ;PROGRAMME ?
70  1C54  D003                  BNE     READ    ;NO - MUST BE READ ETC.
71  1C56  4CBC1C                JMP     PROG    ;IF SO JUMP
72                      ;
73                      ;READING CODE
74                      ;
75  1C59  A930          READ:   LDA     £30H    ;CONFIGURE
76  1C5B  8D25BC                STA     IC2CRA  ;IC2PIA
77  1C5E  A900                  LDA     £0      ;AS
78  1C60  8D24BC                STA     IC2PIA  ;INPUT
79  1C63  A934                  LDA     £34H    ;FOR
80  1C65  8D25BC                STA     IC2CRA  ;READ
81  1C68  AE0C1C        LOOPR:  LDX     TYPE    ;DEVICE TYPE INDEX
82  1C6B  BD0D1C                LDA     READS3,X ;SET
83  1C6E  8D23BC                STA     IC1CRB  ;UP
84  1C71  A900                  LDA     £0      ;INITIAL
85  1C73  8D20BC                STA     IC1PIA  ;CONDITIONS
86  1C76  A93C                  LDA     £3CH    ;FOR
87  1C78  8D25BC                STA     IC2CRA  ;READ
88  1C7B  BD161C                LDA     READS6,X ;
89  1C7E  8D26BC                STA     IC2PIB  ;
90  1C81  209C1D                JSR     DELAY   ;SHORT DELAY
91  1C84  20621D                JSR     INCADD  ;NEXT RAM/EPROM ADDRESS
92  1C87  D003                  BNE     CONTR   ;IF NOT LAST - CONTINUE
93  1C89  4C131D                JMP     EXIT    ;ELSE GO AWAY
94  1C8C  20171D        CONTR:  JSR     SETADD  ;PUT ADDRESS ON PINS
```

Table 1 The assembler listing. Note that:–

1). All pound signs (£) in this listing should be read as hash signs (#). It is purely a function of the printer used that these have been reproduced incorrectly.

2). The assembler listing refers to the two 6821 PIA's as IC1 and IC2 as was the case on the first prototype. IC1 should now be read as IC9 and IC2 as IC8. This in no way alters the operation of the software.

```
 95   1C8F  AD25BC          LDA     IC2CRA      ;SET
 96   1C92  4908            EOR     £8          ;CHIP ENABLE
 97   1C94  8D25BC          STA     IC2CRA      ;LOW
 98   1C97  209C1D          JSR     DELAY       ;SHORT DELAY
 99   1C9A  AD24BC          LDA     IC2PIA      ;GET DATA FROM EPROM
100   1C9D  8D0A1C          STA     RDATA       ;AND STORE IT
101   1CA0  A200            LDX     £0          ;ZERO INDEX REGISTER
102   1CA2  AC0B1C          LDY     MODE        ;READ, TEST OR VERIFY ?
103   1CA5  F010            BEQ     RR          ;READ
104   1CA7  3007            BMI     VV          ;VERIFY
105   1CA9  C9FF            CMP     £0FFH       ;MUST BE VERIFY  - FF ?
106   1CAB  F0BB            BEQ     LOOPR       ;IF SO ROUND AGAIN
107   1CAD  4C131D          JMP     EXIT        ;ELSE NOT ERASED - EXIT
108   1CB0  C135    VV:     CMP     (ZPLOAD,X)  ;COMPARE WITH RAM
109   1CB2  F0B4            BEQ     LOOPR       ;IF OK LOOP
110   1CB4  4C131D          JMP     EXIT        ;ELSE BAD EPROM-EXIT
111   1CB7  8135    RR:     STA     (ZPLOAD,X)  ;STORE DATA IN RAM
112   1CB9  4C681C          JMP     LOOPR       ;ROUND AGAIN
113                 ;
114                 ;PROGRAMMING CODE
115                 ;
116   1CBC  A930    PROG:   LDA     £30H        ;CONFIGURE
117   1CBE  8D25BC          STA     IC2CRA      ;IC2PIA
118   1CC1  A9FF            LDA     £0FFH       ;AS
119   1CC3  8D24BC          STA     IC2PIA      ;OUTPUT
120   1CC6  A934            LDA     £34H        ;FOR
121   1CC8  8D25BC          STA     IC2CRA      ;PROGRAMME
122   1CCB  AE0C1C  LOOPP:  LDX     TYPE        ;EPROM TYPE
123   1CCE  BD0D1C          LDA     READS3,X    ;SET
124   1CD1  8D23BC          STA     IC1CRB      ;UP
125   1CD4  A900            LDA     £0          ;INITIAL
126   1CD6  8D20BC          STA     IC1PIA      ;CONDITIONS
127   1CD9  BD1F1C          LDA     PROGS5,X    ;FOR
128   1CDC  8D25BC          STA     IC2CRA      ;PROGRAMME
129   1CDF  BD281C          LDA     PROGS6,X    ;
130   1CE2  8D26BC          STA     IC2PIB      ;
131   1CE5  209C1D          JSR     DELAY       ;SHORT DELAY
132   1CE8  20621D          JSR     INCADD      ;NEXT RAM/EPROM ADDRESS
133   1CEB  F026            BEQ     EXIT        ;IF LAST THEN EXIT
134   1CED  20171D          JSR     SETADD      ;PUT ADDRESS ON PINS
135   1CF0  A200            LDX     £0          ;ZERO INDEX REGISTER
136   1CF2  A135            LDA     (ZPLOAD,X)  ;GET DATA FROM RAM
137   1CF4  8D24BC          STA     IC2PIA      ;PUT DATA ON PINS
138   1CF7  AE0C1C          LDX     TYPE        ;EPROM TYPE
139   1CFA  BC3A1C          LDY     PTBYTE,X    ;BYTE TO TOGGLE
140   1CFD  B9208C          LDA     IC1PIA,Y    ;LOAD IT
141   1D00  5D311C          EOR     PTBIT,X     ;TOGGLE BIT
142   1D03  99208C          STA     IC1PIA,Y    ;PUT IT BACK AGAIN
143   1D06  A01D            LDY     £1DH        ;CARRY
144   1D08  A2FF            LDX     £0FFH       ;OUT
145   1D0A  CA      DEL:    DEX                 ;50 MS
146   1D0B  D0FD            BNE     DEL         ;DELAY
147   1D0D  88              DEY                 ;FOR
148   1D0E  D0FA            BNE     DEL         ;PROGRAMME
149   1D10  4CCB1C          JMP     LOOPP       ;ROUND AGAIN
150                 ;
151   1D13  20871D  EXIT:   JSR     ZPSWAP      ;SWAP BACK ZERO PAGE
152   1D16  60              RTS                 ;EXIT BACK TO BASIC
153                 ;
154                 ;
155                 ;ROUTINE TO PUT ADDRESS ON APPROPRIATE EPROM PINS
156                 ;
157   1D17  AD051C  SETADD: LDA     LOADE       ;WRITE LOW ORDER BYTE
158   1D1A  8D22BC          STA     IC1PIB      ;STRAIGHT TO EPROM
159   1D1D  BD431C          LDA     A11A12,X    ;25 OR 27 SERIES ?
160   1D20  F009            BEQ     ADD25       ;BRANCH IF 25-SERIES
161   1D22  AD061C          LDA     HIADE       ;GET HI ORDER BYTE
162   1D25  8D20BC          STA     IC1PIA      ;WRITE STRAIGHT TO EPROM
163   1D28  4C521D          JMP     ADD13       ;JUMP TO A13 CODE
164   1D2B  AD20BC  ADD25:  LDA     IC1PIA      ;GET IC1PIA
165   1D2E  0D091C          ORA     HIADM       ;OR IN MASKED HI BYTE
166   1D31  8D20BC          STA     IC1PIA      ;PUT IT BACK AGAIN
167   1D34  AD061C          LDA     HIADE       ;GET HI ORDER BYTE
168   1D37  2910            AND     £10H        ;TEST A12
169   1D39  F008            BEQ     ADD11       ;IF UNSET GO TO A11 CODE
170   1D3B  AD20BC          LDA     IC1PIA      ;A12 IS SET - GET IC1PIA
171   1D3E  0908            ORA     £8H         ;OR IN A12
172   1D40  8D20BC          STA     IC1PIA      ;AND PUT IT BACK AGAIN
173   1D43  AD061C  ADD11:  LDA     HIADE       ;GET HI ORDER BYTE
174   1D46  2908            AND     £8H         ;TEST A11
175   1D48  F008            BEQ     ADD13       ;IF UNSET GO TO A13 CODE
176   1D4A  AD26BC          LDA     IC2PIB      ;A11 IS SET - GET IC2PIB
177   1D4D  0901            ORA     £1H         ;OR IN A11
178   1D4F  8D26BC          STA     IC2PIB      ;PUT IT BACK AGAIN
179   1D52  AD061C  ADD13:  LDA     HIADE       ;GET HI ORDER BYTE
180   1D55  2920            AND     £20H        ;TEST A13
181   1D57  F008            BEQ     ADDEXT      ;IF NOT SET RETURN
182   1D59  AD23BC          LDA     IC1CRB      ;A13 IS SET - GET AC2PIB
183   1D5C  0908            ORA     £8H         ;OR IN A13
184   1D5E  8D23BC          STA     IC1CRB      ;PUT IT BACK AGAIN
185   1D61  60      ADDEXT: RTS                 ;RETURN
186                 ;
187                 ;
188                 ;INCREMENT ADDRESS FOR BOTH RAM AND EPROM, WRITE
```

routines the initial conditions are set up and a loop is executed for each address to be read or programmed. Each time round the loop the appropriate PIA bit is toggled to carry out the required function to the EPROM. In the case of testing the read data is compared with FF (the expected value for an erased EPROM) and in verifying the read data is compared with data in RAM. In either of these two modes, if the test fails the subroutine exits and this is detected in the BASIC program by the fact that the address returned in LOADR and HIADR is less than the expected final RAM address.

The general overview being complete, a few specific points on the tailoring of the assembler routine to suit other machines will now be covered.

a). The origin of the routine should be selected so as not to conflict with the area of RAM used by the BASIC program and the area to be used for data storage (as selected by the BASE command). The origin is set on line 14.

b). The address of the EPROM programmer hardware as selected by the on board links should be equated to IC1PIA on line 23. It should be noted that the addresses in a). and b). must agree with the addresses of the M/C code routine and hardware as assigned in the BASIC program.

c). Lines 143 to 148 are a couple of nested loops which implement the 50ms pulse required for programming EPROMs. These loops will only generate a 50ms delay when running on a processor with a 750 kHz clock such as the Tangerine Microtan. For different clock frequencies the value of 1D loaded into register Y on line 143 would require to be changed proportionally. For example, if the subroutine were to be run on a machine with a 1MHz clock a value of 1D (hex) ★ 1000/750 = 27 (hex) should be used.

As a final point, although the assembler routine has been written for a 6502 processor and as such will not run on any other processor, the hardware is compatible with the increasing number of computers using the 6809 processor. For this reason, some owners of 6809 based machines may like to try their hand at converting this routine to 6809 code. Although, it wouldn't result in the most efficient 6809 code, the easiest way to do this would be to translate it virtually on an instruc-

tion by instruction basis, a task which shouldn't be beyond the capabilities of those with a moderate knowledge of 6809 programming.

The following list contains all those errors which have come to light since the project was published.

On the circuit diagram (page 46, August 1983), C6 is 100μ and C9 is 100n, not the other way round as given. Some bars were omitted from note three of Table 1 on page 48: it should have read "$\overline{CE}$/PGM (27 series) is equivalent to PD/PGM (not PD/$\overline{PGM}$) (25 series)." The penultimate sentence of the first paragraph on page 50 should read "... adjust RV1 for a potential of +25 V at *IC10* output." On the overlay, IC7 is between SK2 and SK1, IC6 is beteen SK1 and C10, IC11 is between R7 and R10, R3 is between R2 and Q2, and Q7 is between Q6 and Q8. A link is missing between IC7 and SK1, and the unidentified pins at the right hand end of SK3 are the +5V line. Finally, C10 appears twice in the parts list but only the first entry is correct, and the second DIL socket should of course, be 8, not 80, way.

Table 1 continued (see notes overleaf).

```
189                              ;HIADM -IE HIADE WITH A11 AND A12 MASKED OUT
190                              ;AND SET Z CONDITION CODE IF LAST ADDRESS DONE
191                              ;
192  1D62  E635     INCADD: INC     ZPLOAD      ;LO ORDER RAM ADDRESS
193  1D64  D002             BNE     INCROM      ;SKIP IF NO CARRY
194  1D66  E636             INC     ZPHIAD      ;ELSE INC HI RAM ADD
195  1D68  EE051C   INCROM: INC     LOADE       ;LO ORDER EPROM ADDRESS
196  1D6B  D003             BNE     INCEXT      ;SKIP IF NO CARRY
197  1D6D  EE061C           INC     HIADE       ;ELSE IN HI EPROM ADD
198  1D70  AD061C   INCEXT: LDA     HIADE       ;GET HI ORDER EPROM
199  1D73  29E7             AND     #0E7H       ;MASK OUT A11, A12
200  1D75  8D091C           STA     HIADM       ;STORE IT
201  1D78  AD051C           LDA     LOADE       ;GET LO EPROM ADDRESS
202  1D7B  CD071C           CMP     LOADF       ;LAST ADDDRESS ?
203  1D7E  D006             BNE     INCRTS      ;IF NOT EXIT
204  1D80  AD061C           LDA     HIADE       ;ELSE TEST HI ORDER
205  1D83  CD081C           CMP     HIADF       ;LAST ADDRESS ?
206  1D86  60       INCRTS: RTS                 ;RETURN
207                              ;
208                              ;
209                              ;THIS ROUTINE SWAPS ZPLOAD WITH LOADR
210                              ;AND ZPHIAD WITH HIADR
211                              ;
212  1D87  A635     ZPSWAP: LDX     ZPLOAD      ;GET LO ZERO PAGE
213  1D89  A436             LDY     ZPHIAD      ;AND HI ZERO PAGE
214  1D8B  AD031C           LDA     LOADR       ;GET LO RAM
215  1D8E  8535             STA     ZPLOAD      ;WRITE TO LO ZERO PAGE
216  1D90  AD041C           LDA     HIADR       ;GET HI RAM
217  1D93  8536             STA     ZPHIAD      ;WRITE TO HI ZERO PAGE
218  1D95  8E031C           STX     LOADR       ;ZERO PAGE TO LO RAM
219  1D98  8C041C           STY     HIADR       ;ZERO PAGE TO HI RAM
220  1D9B  60               RTS                 ;RETURN
221                              ;
222                              ;
223                              ;SHORT DELAY TO ACCOUNT FOR CAPACITOR SLOWING
224                              ;SIGNALS ON PIN 22 (20)
225                              ;
226  1D9C  A080     DELAY:  LDY     #80H        ;LOOP COUNTER
227  1D9E  88       DELAY1: DEY                 ;DECREMENT
228  1D9F  D0FD             BNE     DELAY1      ;LOOP IF NOT ZERO
229  1DA1  60               RTS                 ;RETURN
230                              ;
231                              .END
```

ETI

UNIVERSAL EPROM PROGRAMMER

THE SEQUEL TO THE SEQUEL

Some projects just won't lie down — and the EPROM programmer published last year was one of them!

One inevitable fact about projects published in electronics magazines is that although they are believed to be 100% functional at the time they are printed, it is obviously not possible to test them as extensively as if they were developed in a true commercial environment. This fact explains how a particular device may often be built as an ETI project for a faction of the cost of a similar commercial product. For this reason, we very much appreciate feedback from readers about any difficulties they are experiencing with published projects.

In particular we would like to express our gratitude to Graham Davies for the helpful comments he has made with regard to some problems he was having with the EPROM programmer. As a result of this correspondance we are now able to publish the following amendment to the assembler routine which appeared in January 84.

The 50mS programming pulse required to program EPROMs is initiated by lines 138 to 142 of the assembler routine and the code on lines 122 to 130 is relied upon to turn it off by re-setting up the initial conditions after executing the delay loops and jumping back to LOOPP.

The problem with this method is that part of the initialisation code resets IC1PIA to zero on lines 125,126. Since this register contains some high order address bits as well as control lines and on some EPROM types this zeroing takes place before turning off the pulse there is a short time when the programming condition still exists and yet the address has been modified to a value in the range 00 to FFH. Although the duration of this condition is nowhere near the 50mS required to program a location it has been found that the cumulative effect of this happening a number of times can be to overwrite the first 255 bytes of the device.

Although this could probably be cured by changing the order of some of the instructions in lines 122-130, it was considered that a 'play it safe' approach of ensuring that the programming pulse is turned off before jumping back to LOOPP should be adopted. This is done by duplicating lines 138-142 between lines 148 and 149. The modified section of assembler program is shown below together with a new hex dump.

A different problem has been mentioned in connection with using the programmer on machines other than the Microtan. The BBC machine and some others, especially those with disc operating systems, generate

```
1C00   4C 4C 1C 00 00 00 00 00 00 00 00 00 00 3C 3C 3C
1C10   3C 3C 3C 34 34 34 18 18 18 10 10 18 12 10 12 3C
1C20   3C 3C 34 34 3C 3C 3C 3C 08 08 08 05 25 08 22 00
1C30   22 01 01 01 01 01 08 02 08 02 06 06 06 06 06 05
1C40   06 05 06 01 01 01 01 01 00 01 00 01 20 96 1D AD
1C50   0B 1C C9 02 D0 03 4C BC 1C A9 30 8D 25 BC A9 00
1C60   8D 24 BC A9 34 8D 25 BC AE 0C 1C BD 0D 1C 8D 23
1C70   BC A9 00 8D 20 BC A9 3C 8D 25 BC BD 16 1C 8D 26
1C80   BC 20 AB 1D 20 71 1D D0 03 4C 22 1D 20 26 1D AD
1C90   25 BC 49 08 8D 25 BC 20 AB 1D AD 24 BC 8D 0A 1C
1CA0   A2 00 AC 0B 1C F0 10 30 07 C9 FF F0 BB 4C 22 1D
1CB0   C1 35 F0 B4 4C 22 1D 81 35 4C 68 1C A9 30 8D 25
1CC0   BC A9 FF 8D 24 BC A9 34 8D 25 BC AE 0C 1C BD 0D
1CD0   1C 8D 23 BC A9 00 8D 20 BC BD 1F 1C 8D 25 BC BD
1CE0   28 1C 8D 26 BC 20 AB 1D 20 71 1D F0 35 20 26 1D
1CF0   A2 00 A1 35 8D 24 BC AE 0C 1C BC 3A 1C B9 20 BC
1D00   5D 31 1C 99 20 BC A0 1D A2 FF CA D0 FD 88 D0 FA
1D10   AE 0C 1C BC 3A 1C B9 20 BC 5D 31 1C 99 20 BC 4C
1D20   CB 1C 20 96 1D 60 AD 05 1C 8D 22 BC BD 43 1C F0
1D30   09 AD 06 1C 8D 20 BC 4C 61 1D AD 20 BC 0D 09 1C
1D40   8D 20 BC AD 06 1C 29 10 F0 08 AD 20 BC 09 08 8D
1D50   20 BC AD 06 1C 29 08 F0 08 AD 26 BC 09 01 8D 26
1D60   BC AD 06 1C 29 20 F0 08 AD 23 BC 09 08 8D 23 BC
1D70   60 E6 35 D0 02 E6 36 EE 05 1C D0 03 EE 06 1C AD
1D80   06 1C 29 E7 8D 09 1C AD 05 1C CD 07 1C D0 06 AD
1D90   06 1C CD 08 1C 60 A6 35 A4 36 AD 03 1C 85 35 AD
1DA0   04 1C 85 36 8E 03 1C 8C 04 1C 60 A0 80 88 D0 FD
1DB0   60
```

Fig. 1 The modified hex dump.

regular interrupts in which zero page locations may be overwritten. Since the software presented for use with the programmer uses two zero page locations, 35H and 36H, if either of these were to be accessed in an interrupt routine, then things will obviously go wrong.

The solution here is to re-write portions of the assembler routine to access the data RAM area by some addressing mode which does not require zero page locations. One possible method is to use self-modifying code, or in other words, arrange for the reads and writes to the data RAM area to be made by absolute addressing, altering the op-codes of the instructions to access the next location each time the INCADD routine is executed.

In practice this would involve the following:

1. Remove all references to ZPLOAD, ZPHIAD and the ZPSWAP routine.
2. Change line 108 to VV: CPM VV
   111 to RR: STA RR
   136 to PP: LDA PP

```
138   1CF7  AE0C1C          LDX   TYPE      ;EPROM TYPE
139   1CFA  BC3A1C          LDY   PTBYTE,X  ;BYTE TO TOGGLE
140   1CFD  B920BC          LDA   IC1PIA,Y  ;LOAD IT
141   1D00  5D311C          EOR   PTBIT,X   ;TOGGLE BIT
142   1D03  9920BC          STA   IC1PIA,Y  ;PUT IT BACK AGAIN
143   1D06  A01D            LDY   #1DH      ;CARRY
144   1D08  A2FF            LDX   #0FFH     ;OUT
145   1D0A  CA        DEL:  DEX             ;50 MS
146   1D0B  D0FD            BNE   DEL       ;DELAY
147   1D0D  88              DEY             ;FOR
148   1D0E  D0EA            BNE   DEL       ;PROGRAMME
149   1D10  AE0C1C          LDX   TYPE      ;TOGGLE BIT BACK
150   1D13  BC3A1C          LDY   PTBYTE,X  ;
151   1D16  B920BC          LDA   IC1PIA,Y  ;
152   1D19  5D311C          EOR   PTBIT,X   ;
153   1D1C  9920BC          STA   IC1PIA,Y  ;
154   1D1F  4CCB1C          JMP   LOOPP     ;ROUND AGAIN
155
```

3. Insert the following code at the start of the routine i.e. line 66.

```
LDA    LOADR
STA    VV+1
STA    RR+1
STA    PP+1
LDA    HIADR
STA    VV+22
STA    RR+22
STA    PP+22
```

4. Change the start of the INCADD routine to the following:

```
INCADD:  INC   VV+1
         INC   RR+1
         INC   PP+1
         INC   RR+1
         BNE   INCROM
         INC   VV+2
         INC   RR+2
         INC   PP+2
INCROM: (as before . . .)
```

As a final point, although this doesn't affect the operation of the program, two comments are incorrect in the assembler listing published in January 1984. The following are the correct versions of the comments:

line 44 : ;0-R, 80H-V, 1-T, 2-P

line 105 : ;MUST BE TEST – FF ?

Fig. 2 The modified section of the assembler program. **ETI**