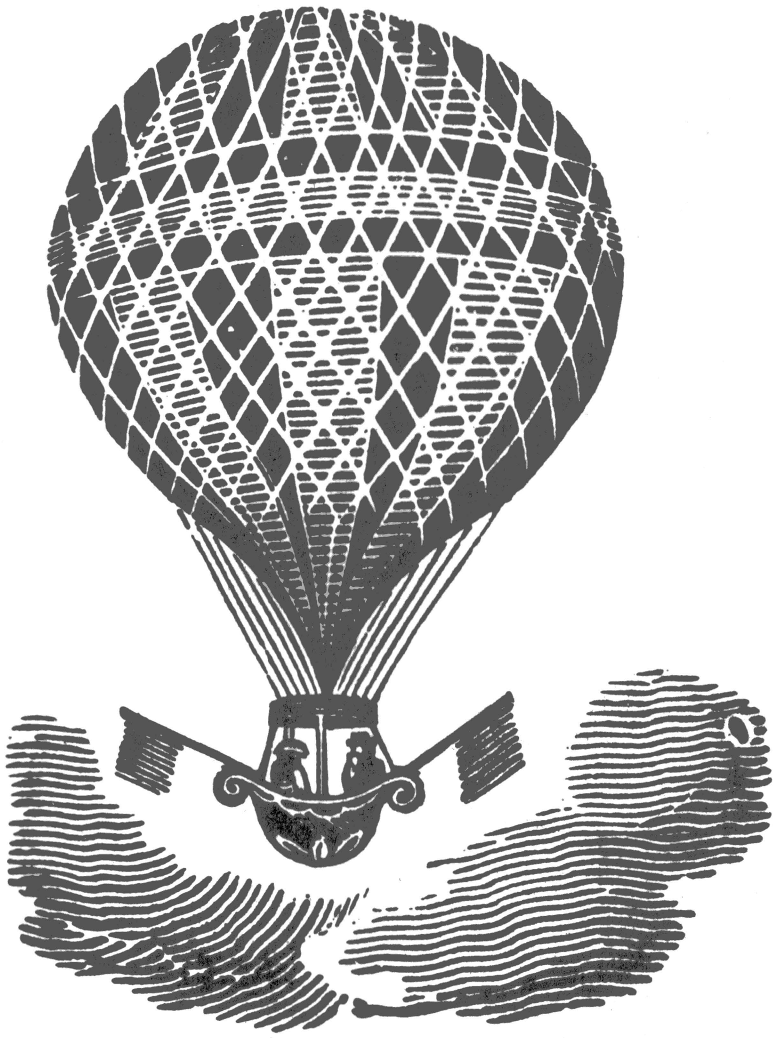
OSI/UK User Group Newsletter

Vol.2 No.4 June 1981



Speeding things up

600 baud cassette interface ☐ relocating the assembler
disk data separators ☐ and our usual miscellany

Editorial

Time for some changes?
As usual, things have been happening quietly in the background of the OSI scene. Again, we're late with this issue — reasons for this anon — and again the 'official distributor' status of our friends at American Data has been coming under scrutiny. With their exclusive European distributor contract coming up for renewal at the end of July, AmData decided to review their prices 'in line with the falling pound', we're told. The result: Superboards at a mere £209, a C1 at around £350, and so on. We picked up many a comment from dealers that OSI kit is unsaleable at these kind of prices. Bob Crook, Alan Davies and company at OS(UK) are supposed to be aiming at the business market, but this new trick has priced the C3 above the PDP-11! At the same time, Chris Cary has put the price of the UK101 *down*, to £120 for the kit version.

However, all is not so gloomy as it may sound. Urgent representations from the dealers seem to have had the desired effect, since the new price list seems to have been withdrawn at present; and Jim Cross, Ohio's sales manager, even came over to the MicroSystems show for discussions with dealers about the tangled state of affairs over here. The grapevine suggests that AmData's contract will *not* be renewed, but we will still be able to keep OS(UK)'s facilities, as part of Ohio Scientific itself rather than a 'pretend' version — to give us, at last, proper manufacturer backup in Britain. Whether this will prove true we have yet to see; I certainly hope it will.

Further whispers from the grapevine state that there will be *no* immediate replacement, and no new support facilities, for the Superboard *et al.*; the nearest thing to it will be the C5-DT, the baby member of OSI's IBS-Net business system. OSI aren't actually dropping their hobbyist side, but we suspect that without a few prods from us it will be left to fade away on its own.

Which is not what we want to happen to this User Group, and which is why I am having to hand over the editorship of this Newsletter. As explained at the end of this issue, I can no longer afford the time to run the editorial side alone; Richard and George, along with Dave Caine, Bob Bonser and others, will be taking this over during the next few issues, so I can deal more effectively with production and distribution. We will, incidentally, be at the *PCW* Show, September 10th–12th — stand T5 — we look forward to seeing you there!

Editor/production/documentation
Tom Graves: 19a West End, Street, Somerset BA16 0LQ; Street (0458) 45359 (early evenings *only*, preferably Tuesdays or Wednesdays)
Hardware/disk systems; new members
George Chkiantz, Richard Elen: 12 Bennerley Road, London SW11 6DS

For those who are new to the Group, we exist to provide an information service and exchange on all matters on Ohio Scientific and related systems — UK101 and others. Membership is £10 a year, mostly for six issues of this Newsletter; the 'year' begins December. We also handle technical queries and the like; but as mentioned above, it's becoming increasingly difficult to answer them promptly!

BASIC Notes

The WAIT instruction
A few notes from *Brian James* of Salford University:

WAIT I,J,K The WAIT statement is intended for programs which involve the use of some interfaced device connected to the computer. It is used when the statements following it must only proceed after some event signalled by the interfaced device. (See the earlier discussion of WAIT in Vol.2 No.2 — *Ed.*) The signal from the interfaced device is sent to a register at the address *I* referred to in the WAIT statement, the signal being connected to one of the data lines D0 to D7. There may be a total of eight signals associated with address I, such as the eight individual lines of a PIA.

The action of WAIT I,J,K is to take the contents (*C*) of address I and exclusive-OR them with *K* which should contain the initial pattern expected at the address I. This means that the individual bits of *C* are compared with the bits in the same position of *K* and each give a bit value of 0 where *C* and *K* match and a value of 1 when they do not match (see the exclusive-OR truth-table below). For example:

```
                    C  = 11000111
                    K  = 00001111
would give          RR = 11001000
(where R is the resultant bit setting)
and                 C  = 00001001
                    K  = 00001001
would give          R  = 00000000
```

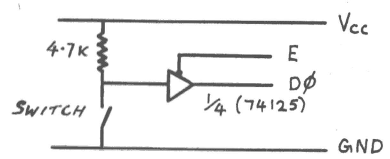
so that if the binary pattern set in *K* exactly matches the contents of the address I, namely *C*, we will get an output of 0 in *R*. This result is then ANDed with *J*, for example if *J*=8 which is 00001000 we can inspect whether D3, the fourth bit has changed, and if *J*=9 which is 00001001 we can inspect D3 or D0. The value of *J*, in decimal, indicates which bits are to be inspected, and the value of *K*, in decimal, gives the binary pattern of the expected initial setting of *C*, the contents of the address I.

The statement WAIT I,J,K will prevent further execution of the program until the pattern at I changes as determined by the values of *J* and *K*. For example if we want to wait until either D0 or D3 change from 1 to 0, or D5 changes from 0 to 1, the binary pattern in *K* should be set to *xx*0*x*1*xx*1 (where *x* = 'don't care') the setting is done in decimal as 9 (or in this case any number with 0 in D5, and 1 in both D3 and D0). The binary pattern in *J*, to select the data bits to be examined, should be set as 00101001, which is 41 in decimal. If we wish to test for *both* D0 and D3 changing from 1 to 0 we should have to use more than one WAIT statement, WAIT I,1,1 to wait for D0 to change and WAIT I,8,8 to wait for D3. The WAIT statement prevents continuation of the program until the result of its action is non-zero.

The interfaced device at address I is most conveniently buffered to the microprocessor data bus by a transparent buffer such as a 74125. (Note that the 74125 buffer has tri-state outputs). The output enable *E* of the buffer is driven by the decoded address I. The 74125 may be used to buffer logic levels from A-to-D converters where signals such as 'busy' and 'data valid' have to be examined, and also the setting of switches in circuits such as that shown in the diagram below.

C	K	R
0	0	0
0	1	1
1	0	1
1	1	0

Truth-table for X-OR function



Circuit to detect switch position

Also on WAIT, a few notes from *Jack Pike*:
The only time I have used WAIT other than when testing it was when trying to input BASIC programs as strings, and hit delimiter problems. I have enclosed a BASIC program using WAIT (shown below -- *Ed.*) which reads a BASIC program directly into RAM without tokenising it, and then will read it back out onto tape. The plan was to be able to modify the program whilst in RAM, mainly so I could automate transferral of BASIC programs written for other machines. As soon as I have my RS232 interface working I will have access to a large range of BASIC programs that need few mods to be able to run on a Superboard. I have not yet written the conversion program, there is just too much to do!

The program uses WAIT to input a BASIC program from tape into RAM from $800 on, and to output the program again to save it (on hitting the space-bar). It was developed to get round the problems of having characters like , ; " and @ in string input. Only characters 0–31 are masked (line 40) from the input. They cause a new line to be initiated (when "line-input" flag I=0). The array P contains pointers to the start of each new line in RAM (up to 100 lines). Lines 100 on output the program for saving on tape.

Obviously the application of this program is a bit specialised. It was developed to check the feasibility of this type of input. Thus although it "works", it is not necessarily efficient or robust.

Practical Electronics UK101 interface system — a review

Pat J. Gillen (G4GVW)

Commencing in their January 1981 issue, *Practical Electronics* magazine has been running a series of articles on the subject of interfacing the UK101 single-board computer to the outside world. The UK101 is based on the Superboard and the author of the series, D.E. Graham, has intimated that the interfaces described in the articles are Superboard compatible.

Interfacing of UK101 and Superboard is a subject of considerable interest to many owners of these computers as in their basic form these boards do not have accessible ports for interfacing to external facilities apart from the provisions made for cassette etc.

Although there are now quite a number of expansion boards on the market offering interfacing facilities, there is evidence to suggest that the *PE* series has aroused much interest. With this in mind, I thought it would be useful to provide Group members with a brief review of one constructor's experiences and opinions of part of the interface system.

Documentation

The series of articles describing the interfaces is written by D.E. Graham and accompanied by a considerable amount of illustrative and tabular information. Mr. Graham's text is extremely interesting and he goes to considerable trouble to fully explain the principles behind his approach. Whether intending to construct the interfacing modules or not, the UK101 (or Superboard) owner wishing to improve his knowledge of the principles of computers applied to control and data acquisition would learn much from the series of articles. For the owner who does construct the modules, the articles form the basis of a documentation file which will be far superior to that provided for the main board.

At the time this review was written, (April) the series had described the construction and application of two modules. The first of these is the subject of this review and is described as the *decoding module*. The second module in the series is designed to 'plug in' to the first module.

The decoding module

The module was designed to provide address decoding, user port and power supply facilities for a wide range of external control devices and expansions. Included in the text are a number of such ideas together with short software routines to demonstrate their operation. Decoded lines are provided for particular application to further interfaces to be described in later *PE* articles but more than sufficient information is given for the experienced constructor to let his (or her) imagination run riot.

Construction

I am usually loath to purchase kits, preferring to use components to hand for most constructional purposes. However, due to business pressures and a reasonable price being quoted for the kit from *Technomatic Ltd* I decided to purchase the kit to save time on the project.

All the parts provided were of acceptable quality and well packed to guard against transit damage. The printed-circuit board is double-sided but does not have plated-through holes. Connection between top and bottom tracks is accomplished in the usual way by soldering pins (not provided) or suitably sized wires in the appropriate locations.

All ICs are socketed and plugs are provided for the DIL sockets which are used for connecting to the ports. Also provided are two edge connectors for making connection with parts of the board. Herein lies the first snag! The edge connectors provided do not have fixing lugs and, to compound the problem, someone forgot to provide any fixing holes on the PC board. I prefer to mount circuit boards firmly in place when finished, and have seen some unfortunate mishaps to lovingly-constructed sub-assemblies left dangling in their inter-connecting wiring. However, these problems have been overcome by a suitable 'bodge'. I feel that the board would have been a lot better cosmetically had the edge connectors been located on a common edge or, better still, have been replaced by DIL connectors. Construction was completed in an evening and all the usual checks and tests completed before installing in a Superboard case. A few evenings later, I found the time to carry out some operational tests of the module. As they say at Houston, "All systems were go!!!"

So, what will it do?

For the more experienced, the provision of ports to the UK101 and Superboard adds facilities which are sadly lacking and for which applications are obvious. For the less knowledgeable, however, it might be useful if we dwell on a brief summary of the opportunities opened up.

By attaching quite simple circuits to the ports and writing very simple software routines, it is possible to control external devices such as lamps, relays, sound generators and almost anything for which an electrical control can be devised. Conversely, the computer can be controlled or provided with data by external devices. Examples which can be a lot of fun include joystick controls for games, switches, light-sensitive devices and, again, anything which can be made to provide a suitable electrical signal. In my case, for instance, the ports connect to an interface which takes Morse Code as an audio signal from my transmitting and receiving equipment, resolves the audio into a stream of logic which is then decoded by the computer and displayed on the VDU. The software also provides for the encoding of keyboard input into Morse and its output to a relay which in turn keys the transmitter.

Conclusion

For those who are not afraid to pick up a soldering iron, the construction of your own kit has a lot to commend it. You may not necessarily save money, but you will gain satisfaction from the process and the knowledge gained. In the case of the interface with which this review has been concerned, I feel able to recommend its construction to those who are still (as most of us) learning our way around the world of computers. Despite the one or two untidinesses described, it is relatively easy to construct and gains much from the excellences of the articles in *Practical Electronics*. The *Technomatics* kit, while not being a bargain, is worth buying — particularly if you do not have a good components shop in town. Unlike some kits in my experience, all the parts were provided, all were of acceptable quality and ... *very important* ... they did fit the board.

In due course, I will probably construct further items from Mr. Graham's series, and may be able to send in a further review.

In the meantime, how about other radio amateurs in the Group making themselves known and sending in a few notes on their applications of OSI and UK101 kit?

[*Ed.* — in case anyone doesn't yet know, the UK101 is best described as a 'photocopy' of the Series I Superboard — in most places you have to look hard to find the differences, not the similarities! The Series II Superboard is essentially the same, the differences being restricted to the video handling and the on-board implementation of RS232 and modem lines. So the *PE* articles, which relate mostly to the 40-pin output socket on the right-hand side of the board, should apply to the Superboard and C1 series machines with no alteration at all. The larger machines in the OSI range will probably be unable to use this expansion, although Martin Kay's Zen bus-to-bus interface board may be of use here (see Vol.1 No.4)].

Adding a disk drive: the data separator

Richard Elen

One curious fact about OSI's disk systems is that they rely almost entirely on software to derive timing signals, etc. This no doubt derives from the fact that OSI designed their disk systems some time ago, before modern LSI disk-controller chips were available. One of the functions generally carried out by these ICs is that of separating data and clock signals read from disk. As a result, many modern drives do not include a data separator on the drive, although early ones generally did (and now that much drive electronics is being replaced by other LSI chips, some *very* new drives include data separation once again).

The result of all this is that if you 'buy in' your drive to upgrade your OSI system, i.e. you don't get the drive via OSI, you may be missing a data separator. On some OSI systems, the drive is fitted with a little add-on board to handle this function (notably MPI and Shugart 5¼in floppies). If you've obtained your drive elsewhere, and intend to interface it with a 470, 505, 610 or 630 board, you will encounter one of three possibilities:

1. *A data separator is fitted.* In this case, simply use the separated data and clock lines in your interface.
2. *A data separator option is supplied on the board, but unpopulated.* In this case, it is most sensible to construct the separator in the space provided, following the instructions and circuit described by the manufacturer. Separators differ in design, so we can give little guidance here. Make sure you have the manuals; if you can't work out the on-board requirements, you will have to build this circuit.
3. *No option is fitted.* In this case, you will have to build your own, as shown here.

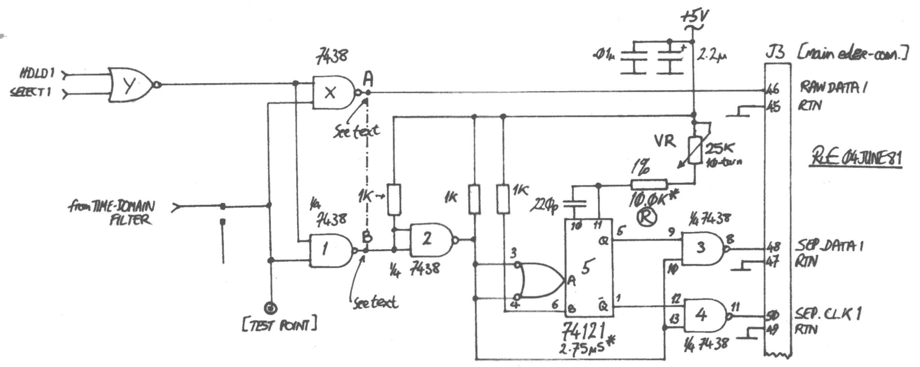


Fig. 1 TYPICAL DATA SEPARATOR circuit [BASF]

* 10.0KΩ [2.75μS] for 8in Drives
22KΩ [6μS] for 5¼in Drives

Fig.1 shows a typical data separator circuit which may be added either on the disk drive itself or on the controller board. It may be constructed on a small piece of Veroboard and placed in any convenient location where it can pick up the raw data line and five volts (plus ground).

The circuit described is that used on BASF 8in drives until quite recently. Later models followed the course mentioned above: they don't even have the option available. BASF are now developing and supplying drives with an LSI controller: this has the separator included in the LSI and you don't need to do any work at all.

As an aside, BASF drives are a very good investment if you're thinking of buying one new. They are very reliable, and thus don't seem to exist on the used equipment market. I have been running an 8in double-sided drive for some months (see last issue): it is beautifully made, and *exceptionally* quiet in operation. None of your usual clatter! A good supplier of these drives, both 8in and 5¼in, is Melkuist Ltd of 35a Guildford Street, Luton LU1 2NQ. They use them in their studio automation system because of their reliability, and because they are so quiet they can be used in the studio control room without making a distracting racket. Mention the User Group if you buy one from them.

In the circuit shown, the raw data signal is derived by NANDing the gated result of the HeadLoad and Select signals with a line from the Time-Domain filter. As the separator is an option on the BASF drives of this type, they fit one gate (labelled X) as standard, to supply the raw data line, and parallel it with another gate (1). This enables all the signals, separated and raw, to be available when the option is fitted, but still allows raw data to be output even if there isn't a separator component on the board. If you use this circuit, gates X and Y will already exist (or their equivalent circuitry), and Gate 1 (¼ 7438) can be omitted, connecting points A and B together to drive the separator. In other words, you only need to construct the circuit from point B onwards, feeding point B with simple raw data. Note that all the NAND gates (X and 1 through 4) are open-collector varieties: hence the 1K pullup resistors. Note that resistor R is a 1% component for stability.

As 5¼in and 8in drives require different time constants in the 74121 monostable, resistor R should by 10.0K for 8in drives (2.75 microseconds) and 22.0K for 5¼in ($6\mu S$).

The circuit shown is not the world's most amazing data separator. Many disk-controller hardware designs utilise a phase-locked loop separator, which is more stable and can cope with speed variations in the disk with greater reliability. However, the monostable-based design works faultlessly in my case, and anyway, OSI's disk controller circuitry makes great use of monostables for received data timing: a PLL separator would represent a bit of electronic overkill! This leads to an interesting speculation: it should be possible to redesign OSI's receive data monostable to become a data separator with a few mods on the controller board, and we're looking into it. At present, though, until we have investigated more fully, we recommend the approach described here — we know that it works!

With reference to the edge-connector pinouts described by the manufacturer of your drive, it will be possible to deduce the correct lines on the ribbon-cable which are to take the separated data and clock signals. The pin numbers on the diagram refer to *standard 8in drive* edge-connector pinouts, as used by Siemens, Shugart and BASF among others. *Minifloppy drives are different:* refer to the manufacturer's diagrams.

Setting up
When the separator has been constructed and installed, and the other interface lines have been connected, it will be necessary to set up the data separator. This is normally done with a special set-up alignment disk, which is hard to find and expensive. Unless you can get hold of one, follow the 'alignment algorithm' described below:

Data separator alignment algorithm
1. Rotate the 10-turn 25K trimmer fully *anticlockwise* to the end of its travel.
2. Power up the system and insert a disk with operating system.
3. Hit BREAK and 'D' to boot the system. (It will fail to boot).
4. Rotate the trimmer control one half-turn *clockwise*.
5. Re-boot the system (BREAK and 'D').
6. IF booting fails, GOTO STEP 4.
   ELSE:
7. Rotate the trimmer one *quarter-turn* clockwise.
8. Hit BREAK and 'D' to re-boot.
9. IF boot is successful, GOTO STEP 7.
   ELSE:
10. Reset the trimmer to a position midway between the positions at which it *first* and *last* booted successfully. Reboot to test this setting.
11. If boot is successful, the separator is now aligned.

Note: If the system *never* boots successfully, check the wiring of the separator and interface and try again from step 1. If it *always* boots successfully (most unlikely) then set the trimmer to the middle of its travel.

After alignment, check that the boot is generally reliable, re-running this procedure if it isn't. If the system never appears to 'settle down', recheck the wiring thoroughly once again.

Relocating the Assembler

Tony Parsons

I have made the assumption that a person wishing to relocate the assembler will also have sufficient memory to accommodate a relocated Extended Monitor, in my case $42AF.

The Extended Monitor's 'Relocate' facility is used initially to move the assembler. (The program calls the appropriate routines). The program then uses two tables to tidy up. The "Cat 1" job is to correct all the changes the Relocate made but should not have. The "Cat 2" job is to change locations the Relocator should have but didn't, like hidden addresses etc.

Note 1. Address $45F7 is the relocated ExMon's equivalent of the original $0B48. You will have to insert the appropriate address for your configuration.

Note 2. As for Note 1: Original address for this routine is $0DBA.

The next step after complying with the two notes above, is to calculate the difference in Hex between the original location of the assembler and the proposed new location. From this the equivalent of the old start address $1300 should be derived, together with the updated locations for source file begin, end and 'Next Byte' stores, the original locations being $12C9, $12CB and $12FE.

Run the program!
The prompt will be 'R'.
Reply XXXX=0240,1391
—just as in ExMon's relocate function where XXXX is the new first location, on 'Return' the program will execute. When done it jumps to $FE00.

You should next insert the required values for source file locations.As an aid I have listed the new values for my machine.

First location $4AAF
Difference $486F (Between start new & start old) ie 4AAF-0240.
Entry Point $5B6F (1300 + 486F)
Source File start $5B38 LO $5B39 HI
Source File end pointer $5B3A $5B39
Next location $5B6D $5B6E

My source file begins $200 and ends $42AE. Whilst it is not essential (I think all bugs have been removed) it is advisable to install the self healing patch at the relocated equivalent of address $029C.

Using my case again as an example we have at $4B0B ($029C+ $486F)

```
$4B0B   JMP $4972      ;inserted bit
$4B0E   INY
$4B0F   LDA ($24),Y
etc.

$4972   INY
$4973   LDA ($24),Y
$4975   CMP #$14
$4977   BCS $498D
$4979   DEY
$497A   LDA ($24),Y
$497C   CLC
$497D   ADC #$6F       ;Lo Byte of difference
$497F   STA ($24),Y
$4981   INY
$4982   LDA ($24),Y
$4984   ADC #$48       ;Hi Byte of difference
$4986   STA ($24),Y
$4988   LDA #$3F
$498A   JSR $FFEE      ;Print '?' if an address that Parsons
$498D   DEY            ;missed has just been corrected
$498E   LDA ($24),Y
$4990   TAX
$4991   JMP $4B0E
```

You should now have a working assembler in a new location. Test it by deleting the old one from memory and thoroughly exercising the new one with deliberate errors etc.

Dealer Notes

More dealers...
Trickling down the grapevine has come information on some of the new dealers recruited by OS(UK). One, *Tomorrow MicroSystems*, based at 1 Queen Street, Hadleigh, Suffolk (0473 823698), has now become the 'retail wing' of OS(UK) itself; quite what that means we're not sure, but it seems that they themselves are recruiting subsidiary dealers, one being Steve Morrall of *Online Design* in Portsmouth (0705 738153). Another new dealer is *Tamsys* at 12a Sheet Street, Windsor (Windsor 56747) — the contact there is Philip Bowe. All of these we met briefly at MicroSystems, on the Ohio stand there.

We've heard of a few more in the Home Counties area — *Kram* being one of them — but we don't know who or where they are! Let us know, so that we can publish assorted details!

...one less...
The trials and tribulations of *Beaver Systems* have been confusing a number of people — not least Steve Hanlan, who ran the company! Originally based in Thame, the company changed both address and name at the beginning of this year — the address because their rented offices were sold, the name because an automatic-test-equipment firm informed them that they had prior and sole rights to the name 'Beaver'. Next door to us in Street, and under the new name of *Avalon Computers*, things went back to normal for a while. Then along came DJ Systems, of "The Last One" fame, needing a contract programmer to handle implementation work on the Sharp and Ohio systems. Exit Steve. And, effectively, exit the new-fledged Avalon, for Steve is now down at Ilminster full-time. He'll be at the PCW show, selling versions of The Last One; and that will be the final appearance of Avalon. Apart from TLO enquiries, Steve is referring all software matters to *Premier Publications*, the Croydon software house; for hardware enquiries refer to your local dealer.

...and perhaps none at all?
We are actually a little worried about Ohio in general, and OS(UK) in particular. At the same time as Comp have put their price down on the UK101 (to £120, or perhaps even £99, we're told, for the kit version), the American original has gone *up*, with the fall of the £, to an unsaleable £200-odd — a slight difference! As mentioned in the editorial to this issue, the other hardware prices have gone up by the same amount, making even the big C3 systems look over-priced. Since OSI sales depended on the fact of their being low-priced, this makes things a trifle difficult over here. The lack of applications software has not helped matters either — 65U may be a very good operating system for data-base work, and much faster than yer typical CP/M, but there simply isn't any software about; and OSI's infamous documentation (now rapidly improving, we're glad to report — there are now some usable BASIC manuals about) has not exactly made it easy to develop your own. Despite the relative crudeness of CP/M, it *is* becoming a standard; we wonder if OSI's main sales of the C3 series in the States have been more for their use as CP/M machines than for 65U. In any case, if those ridiculous prices are not revised, it's going to be bye-bye, OSI, we suspect...

Superboard as development system
One of the more annoying aspects of the Ohio scene is the limited range of software tools available. One of the more interesting of new developments, then, is one which combines almost all of the available development tools into a single add-on board. Our member Martin Spalton, who now runs *MCS Electronics*, has

been selling an improved version of both the Assembler and ExMon in a set of four 2716-type EPROMs for some time now (see his advert below); he's now added a compatible PROM-programmer/EPROM board to his range, giving (when ExMon and Assembler are installed) all the basic development tools for machine-code work on a single add-on board, ready to run on power-on. The board will sell as a kit for about £60, Martin tells, us, with ExMon and the Assembler at £6 and £20 extra respectively; we're told that Mutek will also be selling a ready-built version for Martin later this year. We have one on order at present, and intend to include a review of it shortly.

Small ads

BASIC programmer's toolkit for UK101, C1/Superboard, C1E and C2 BASIC-in-ROM systems. 2K of machine code on tape giving 14 new commands including RENUMBER, DELETE, SEARCH, TRACE and OLD. Compatible with all monitors (CEGMON only for C2). Only £10 inclusive (£9 for User Group members), with full instructions. State computer type and memory size. Send SAE for more details. An EPROM version could be produced if there is sufficient interest. S.A. Smith, 36 Woodvale Ave, London SE25 4AE.

Garbage-collector revision now available in mask programmed ROM. Direct plug-in replacement for original BASIC-3 — no soldering. Suitable for Superboard and UK101. Special price to group members £7. Please enclose suitable SAE. R.W. Stibbons, 3 Mansfield Drive, Hayes, Midx, UB4 8DZ.

Superboard II for sale: 8K RAM computer in excellent working order, switchable 1MHz or 2MHz with separate power supply, metal case. Complete with manuals and games tapes. £210 ono. Phone Medway (0634) 252845.

User Group Notes

PCW Show

As mentioned elsewhere in this issue, we will be at this year's PCW Show, 10–12 September (Thursday to Friday), at the Cunard Hotel in Hammersmith, next to the A4 flyover and Hammersmith 'tube stations. We only have a small stand, of course, in the 'hobbyist' area downstairs; but we felt it would be the easiest way of meeting up with as many of our members as possible. If anyone would like to help us on the stand for any of the days, get in touch with us as soon as possible! In any case, we've been able to get 'Club vouchers' from the show organisers, giving 50p off the entrance fee; you should find one in this copy of the Newsletter. We should have some spares left over; again, let us know. And see you there, we hope!

Instant Machine Code!

Extended Monitor and Assembler in EPROM

The Ohio/Compukit *Extended Monitor* is now available in EPROM, located at $9800. This greatly enhances the machine-code facilities of the standard machine; if you have CEGMON, it adds LOAD and SAVE in both Hex Dump and Checksum formats, not to mention the Disassembler, Search and all the other ExMon facilities.

An improved version of the *Assembler* is also available on a set of three EPROMs addressed from $8000 to $97FF. New features include: user definable source-file space (so that if the program being assembled is to run from, say, page-3, then the source file can be placed further up memory — permitting 'A3' assembly to memory while retaining the source file in RAM); listings and assemblies can be halted by CTRL-S for viewing and restarted by G; and as published by the User Group, line numbers can be suppressed during load from tape, to permit simple merging of library subroutines.

***Prices:* ExMon — £6.00 Assembler — £20.00** New 2716's — £4.50 No VAT. Please include 50p P&P.

Note: For copyright reasons, your original tape of ExMon and/or Assembler should be included with your order.

EPROM Memory/Programming Board

The prime functions of this printed circuit board are *a)* to provide eight 24-pin EPROM sockets as a memory expansion board and *b)* to provide an EPROM programmer; both of which will handle 2516, 2716, 2532 and 2732 4/8Kbyte, 5V devices.

☐ **PCB** The PCB measures 100mm × 275mm and has a single sided 40-pin edge connector making it suitable for Eurocard type rack mounting. It is double-sided with plated through holes, tinned, solder masked, with gold plated edge connectors and silk screen legend. A total of 28 IC sockets are used.

☐ **Buffers** Everything on the board is fully buffered: address lines, R/$\overline{W}$ and Ø2 use 8T97s; data lines use a 74LS245. IRQ and NMI interrupt lines are not used and not buffered.

☐ **Memory** Eight 24-pin EPROM sockets are provided and these are addressed as four 4K blocks. As supplied they are decoded at $8000, $9000, $C000 and $E000 and connected for 2716, but by changing links and tracks on the board may be addressed anywhere in memory space. Most combinations of memory chips may be used, including the pin-compatible Hitachi RAMs; if 8 × 2732-type is required a 13-input NAND gate will be needed to replace the 8-input NAND chip-enable gate supplied as standard.

☐ **Programmer** The EPROM programmer is fully addressed at $F780 but may be addressed elsewhere if required. It is software controlled, using a 6821 PIA for its bidirectional data registers and a 12-bit counter chip which, when clocked by the PIA, supplies addressing for the EPROM being programmed. This feature, combined with the software supplied, makes the programmer very flexible. The hardware requirements of different EPROM types are selected by 'personality plugs' — pre-wired 14-pin DIL headers or, optionally, a single multi-way DIL switch. The V_{pp} programming voltage derived from the power supply provided is software controlled and regulated to 25 volts. A standard low-profile socket (sufficient for hobbyist use) is used as the programming socket, but there is sufficient clearance for a Textool-type zero-insertion-force socket if required.

☐ **Output** The board also provides a 40-pin output socket to allow further expansion to the system. All the address and data lines, R/$\overline{W}$ and Ø2 are buffered; the data-direction signal is also taken care of. This feature permits 'daisy-chaining' of other expansion boards attached to the OSI/UK101 expansion socket, or for other users without a racking system.

***Provisional price:* £60.00** in kit form; exact price dependent on whether software is supplied as listings, on tape or in EPROM.

MCS Electronics An EPROM programming service
9 Willowfields, Hilton, nr Derby, DE6 5GU. *Tel:* 028 373 3802

Make your computer

NAG...

Just one of the many uses to which you can put the fantastic Mutek

Voice Synthesiser

This remarkable unit adds speech output capabilities to almost any computer with a parallel or serial output port.

Infinite Vocabulary

For less than the cost of most limited-vocabulary synthesisers the Mutek **VoxBox** offers full phoneme synthesis using the building blocks of speech. This enables simple and easy generation of programmed speech and sentences of any length.

Fits Most Computers

Within minutes of receiving the **VoxBox** you will be able to generate intelligible speech using the comprehensive documentation supplied with the unit. For most computers the interfacing is just a matter of plugging in — and the relevant connector is supplied as part of the package.

Listen before you buy!

We can't honestly say that the quality is as good as human speech — but why not call *Bath (0225) 743289* after 6pm and listen to our computer talking through **VoxBox**. You may place a credit-card order or a request for further information after the message.

Models currently available:

VB-1 **VoxBox** for all PETs **£59**
VB-2 **VoxBox** for Acorn Atom **£59**
(these units complete with cable and connector)
VB-3 **VoxBox** for 6821-type PIA parallel port **£57**
(complete with cable and data for connection)
VB-4 RS-232 **VoxBox** with serial interface* **£83**
(**note:* CTS signal must be available)
VB-5 RS-232 **VoxBox** for Superboard/UK101 .... **£83**
(with interface cable and data for connection)
The VB-4/VB-5 units include Mutek's serial to parallel conversion board — an optional built-in Centronics printer port is available for these units at **£5.00** extra.
VC-1 PET demonstration cassette ... **£2.50**
VC-2 Atom demonstration cassette **£2.50**
Versions available for other makes of computer — please call for details.

All units are ready-built, fully guaranteed and complete with case and speaker, mains PSU and lead and comprehensive manual. *The manual is also available separately at £1.50* (credited against order).
Please include VAT at the appropriate rate and £2.00 for carriage for any number of units.

Mutek Quarry Hill, Box, Wilts. Tel: Bath (0225) 743289

```
10               *=$5D00
20               JSR $A86C        ; cr/lf
30               LDA #$52
40               JSR $FFEE        ; 'R'
50               JSR $45F7        ; get relocate parameters (see note 1)
55               SEC
60               LDA $DA          ; Calc diff in 1st locations
70               STA $26          ; Store in 20, 21 pair
80               SBC #$40
90               STA $20
100              LDA $DB
110              STA $27
120              SBC #$02
130              STA $21
140              JSR $4869        ; Call relocate (see note 2)
150   CAT1       JSR CLOFST       ; Clear off SGT pair 22, 23: X=#$FF
160   NEXTS      INX
170              LDA CAT1TB, X    ; Get next factor from CAT1 Offset table
180              BNE SUB
190              INC $23
200              LDA $23
210              CMP #$14
220              BNE NEXTS
230              BEQ CAT2
240   SUB        JSR CALPTR       ; Add derived offset to program's start
250              JSR SUBSTO       ;Subtract diff from pair pointed to by 24, 25
260              JMP NEXTS
270   CAT2       JSR CLOFST
280   NEXTA      NOP
290              INX
300              LDA CAT2TB, X
310              BNE ADD
320              INC $23
330              LDA $23
340              CMP #$14
350              BNE NEXTA
360              LDA #$0F         ;Adjust pointer to primary add table
370              STA $23
380              LDA #$49
390              STA $22
400              JSR CALPTR
410              LDY #$00
420              CLC
430              LDA #$80
440              ADC $20
450              STA ($24), Y
460              LDA #$07
470              LDY #$06
480              ADC $21
490              STA ($24), Y
500              JMP $FE00
510   ADD        JSR CALPTR
520              JSR ADDSTO
530              JMP NEXTA
540   CLOFST     LDA #$00
550              STA $22
560              STA $23
```

```
570              TAX
580              DEX
590              RTS
600   CALPTR     STA $22
610              CLC
620              ADC $26
630              STA $24
640              LDA $23
650              ADC $27
660              STA $25
670              RTS
690              SEC
700              LDA ($24), Y
710              SBC $20
720              STA ($24), Y
730              INY
740              LDA ($24), Y
750              SBC $21
760              STA ($24), Y
770              RTS
780   ADDSTO     LDY #$00
790              CLC
800              LDA ($24), Y
810              ADC $20
820              STA ($24), Y
830              INY
840              LDA ($24), Y
850              ADC $21
860              STA ($24), Y
870              RTS
880   CAT1TB     .BYTE $00, $9E
890              .BYTE $00, $3A, $46, $8D
900              .BYTE $00, $97, $BA, $F2
910              .BYTE $00, $47, $4F, $56, $5C, $69, $73, $7F, $83, $A2
920              .BYTE $A7, $AF, $C2, $D6, $DE, $E6, $EC, $FA, $FF
930              .BYTE $00, $14, $2D, $35, $94, $B3, $CF, $E4, $FF
940              .BYTEe $00, $06, $12, $1F, $2F, $6F, $F1
950              .BYTE $00, $0F
960              .BYTE $00, $00, $C8, $F2, $F5
970              .BYTE $00, $28, $87, $93, $99, $A7, $AB, $B1, $E6, $EF
980              .BYTE $00, $C0, $F3
990              .BYTE $00, $39, $48, $53, $63, $78, $A6, $DA, $E0, $E6
1000             .BYTE $00, $1A, $4A, $4E, $92, $9A, $C2, $EA, $FE
1010             .BYTE $00, $3E, $60, $6C, $6F, $72
1020             .BYTE $00, $03, $15, $18
1030             .BYTE $00, $9C
1040             .BYTE $00, $00, $00, $00,
1050  CAT2TB     .BYTE $00, $9D
1060             .BYTE $00, $00, $9C, $D8
1070             .BYTE $00, $21, $24, $27, $2A, $2D, $4E, $9A, $BE
1080             .BYTE $00, $39, $40, $42, $44, $46, $48, $4A, $4C, $4E
1090             .BYTE $50, $52, $54, $56, $58, $5A, $5C, $5E, $60, $62
1100             .BYTE $64, $66, $68, $6A, $6C, $6E, $70, $72, $74, $76
1110             .BYTE $78, $7A, $7C, $7E, $80, $82, $84, $86, $88, $8A
1120             .BYTE $8C, $8E, $98, $E3
1130             .BYTE $00, $08, $0B, $0E, $69, $81
1140             .BYTE $00, $0E, $18, $1B, $4B, $D2, $E1
```

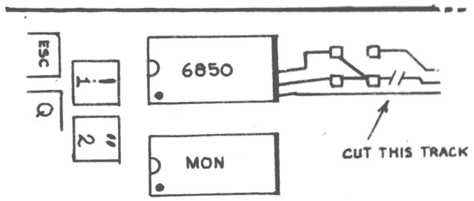
```
1150             .BYTE $00, $00, $35, $3C, $54, $EE
1160             .BYTE $00, $16, $79, $E5
1170             .BYTE $00, $45, $92, $E8
1180             .BYTE $00, $52, $55, $7A, $83, $9C, $B8, $BB
1190             .BYTE $00, $F4
1200             .BYTE $00, $39, $58, $6E, $71, $7E, $8E, $DD, $EE, $FA
1210             .BYTE $00, $05, $08, $14, $1D, $CE, $F9
1220             .BYTE $00, $02, $17, $8D
1230             .BYTE $00, $07, $18
1240             .BYTE $00, $00, $00
```

600 BAUD cassette interface

George Chkiantz

Users may be interested in the following simple modification to double the speed of the cassette interface. This applies to all versions of the 600 board (e.g. all Superboards Series 1 or 2, UK101).

All that is needed is a single pole two-way switch mounted somewhere (the hardest job). Cut the track between pins 3 and 4 of U14 (the ACIA) and TxClock (just above the ACIA). Connect pins 3 and 4 of U14 to the centre pole of the switch. Connect one side of the switch to the original Tx clock (i.e. re-joining the track you have just cut) for 300 Baud operation, and connect the other side of the switch to U57 pin 11, for 600 Baud.



Cut the track marked #

How it works: Tx Clock is derived from the divider chain through U57, which is arranged to divide by 13, and subsequently through U63 which divides by 2. By moving Tx Clock's take off point from U63/9 to U57/11, data will be clocked out of the ACIA at twice the previous rate, although the carrier frequencies will remain the same. Adjustment of the monostable U69 via R57 may be required if it is 'on the edge' but no subsequent adjustment should be needed on switching from 300 to 600 Baud.

With the above method, speeds of 1200 may be attempted (by connecting U14 pins 3 and 4 to U57/12), but reliability may suffer as U57 does not produce symmetrical waveforms at this point. In order to achieve higher interface speeds, higher carrier frequencies or, preferably, a different encoder must be used, such as the one published by ETI or the working version of this sold by Mutek.

These higher speeds bring problems relating to the tape recorder and also to BASIC.

1. The quality of the tape recorder used must be such that its frequency response extends to beyond 10KHz with few drop outs and low wow and flutter. The tape must be of sufficiently good quality for the above objectives to be realised and the heads kept clean and demagnetised.

2. Problems relating to the way BASIC inputs a line remain to be solved. When you write a line of BASIC and press return, the computer does a lot of work to your BASIC line. It scans it, tokenizing or pre-digesting any key-words, finds out where to insert the 'tokenised' line, re-shuffles the stored program to make room for the line, copies it into the space created, and finally re-links the whole program at its re-arranged location. This obviously takes time, the actual amount depending to some degree on how far down the token list your commands actually are (MID$ is last).

Now the cassette interface saves a program as an ASCII file – exactly as it is LISTed. In this way the cassette can pretend to be the keyboard on LOAD. This has the advantage that special 'merge' programs are not needed to concatenate files, and that, should something go wrong, you can nearly always recover some part of your hard work. However, it has the disadvantage that BASIC has to go through the whole of the process described above for every line that comes off the cassette. To give BASIC time to do this, the monitor is preset to issue 10 NULLs after each carriage return when in the SAVE mode. As tape speeds are increased, this number becomes insufficient and so when SAVEing at higher speeds, a suitable number of nulls must be issued. This is accomplished by POKE13,*xx* where *xx* is the number required, which you must determine by experiment and may be as great as 75. Obviously this slows down the whole process, so that as far as BASIC is concerned, a point of diminishing returns is reached as the Baud rate increases.

Even worse, if a line is longer than the screen width, the time taken for the screen to scroll may be enough to cause the loss of the next character, and the previous one is repeated (a good explanation as to why the character is repeated eludes me at present, but it is). Were it to lose the carriage return, the line will not terminate, and all manner of confusion result. One solution to this problem (if you have CEGMON) is to limit the scrolling window or substitute a 'home cursor' command for the carriage return by a small machine code patch to OUTVEC. Another simple solution is to type:

```
SAVE
POKE 13,<number of nulls>
?: ?CHR$(15); CHR$(95): ?: LIST (...start the tape...)
```

when about to SAVE a program which puts a control O on tape and suppresses screen output, at least until an error actually occurs. By far the neatest way of handling the situation is to write an 'indirect file' handler as is used to merge files on the disk systems, we may publish a discussion on this topic at a later date.

As you can see, fast tape handling may bring problems with BASIC. Fortunately this is not the case with the assembler, as long as no line numbers get corrupted (the assembler is not fast at inserting lines!) so if you do a lot of assembler work, 4800 Baud is an unqualified advantage. In machine code LOADs there should be no problems, although a checksum loader might have problems at very high speeds.

I found a 4800 Baud interface on a C2 was well worth while. BASIC loads were much faster, and machine code loads were stunning. ExMon loaded in a few seconds, the assembler in about 35 secs (it took longer to find them than to load them), both in straight OSI tape dump format. If a byte-loader had been used these last could have been speeded up by virtue of the fact that the standard OSI save format for a byte outputs two ASCII digits and a <RETURN>. I found the system reliable enough for everyday use, although I would keep at least two high speed copies of everything, and maybe even a safety copy at 300 Baud. However, I do have a good quality cassette deck....

The BASICs of machine-code
Part 5: The processor status flags

Tom Graves

For the past few parts of this series we have digressed somewhat, through some of the basic concepts of machine-code; now it is time we returned to the main theme of the series, that of presenting and testing machine-code programs in terms of a 'dummy-BASIC'.

In the first part we looked briefly at most of the 6502's instructions, and ways of simulating them in BASIC. One group of instructions which causes a lot of confusion is the *branches* — BNE, BCC and so on. The difficulty seems to be not only the 'relative jump' nature of branches, but also that they are always *conditional* (unlike JMP or JSR, the direct equivalents of GOTO and GOSUB), depending on the state of specific bits in one of the processor's registers, the **P** or *processor status* register. Keeping track of the status of this status register is one of the trickier parts of machine-code programming — and is the usual cause of unexpected jumps in the program!

The register is, like all of the accessible registers in the 6502, eight bits wide; only seven of these are used (bit 5 is the unused one), and only the four controlling the branches — **S**ign, o**V**erflow, **Z**ero and **C**arry — need concern us for now. Of the others, two — **B**reak and **I**nterrupt — are used for interrupt-handling, which comes *much* later, and the last, the **D**ecimal mode flag, is something we had also best leave until later, to avoid deep tangles in the 6502's notions of arithmetic.

The branches operate on the state of the remaining status flags. There are eight branch op-codes, acting as pairs on one of the status bits:

BPL	IF N=0 THEN GOTO...
BMI	IF N=1 THEN GOTO...
BVC	IF V=0 THEN GOTO...
BVS	IF V=1 THEN GOTO...
BNE	IF Z=0 THEN GOTO...
BEQ	IF Z=1 THEN GOTO...
BCC	IF C=0 THEN GOTO...
BCS	IF C=1 THEN GOTO...

Note the catch with BNE and BEQ — they are the other way round to what you might expect! A zero result *sets* the Z flag; a *non-zero* result *clears* the flag.

The simplest way of handling the flags is to treat them as BASIC variables in their own right — we just have to remember that they can only hold the numbers 0 or 1, unlike the A, X and Y variables we are using to represent the system registers. We now also need to add two more variables, namely MEM and RES. MEM we will use as a kind of general-purpose parking area, representing either a memory address or, for some instructions like ROL A which treat registers like memory, for the registers themselves. RES we will use as a kind of 'register for results'; there is actually a temporary-register of this kind within the 6502, but it is not accessible to the programmer.

Once we have these two variables, we can use them to set and clear the status flags — the N, V, Z and C variables — as required by each machine-code instruction. For example, about half of the 6502 instructions act on the N and Z flags, so we can build a BASIC subroutine to represent this:

```
10  Z=1: IF RES THEN Z=0
20  N=0: IF (RES AND 128) THEN N=1
30  RETURN
```

The Zero flag is cleared if there is a non-zero result in RES; the top bit of the byte represents the sign, hence (RES AND 128) returns the value of the sign. (Note that we don't need to say IF RES<>0 THEN... — the simpler statement IF RES THEN... produces the same effect). The op-codes for which this subroutine alone suffices are AND, DEC, DEX, DEY, EOR, INC, INX, INY, LDA, LDX, LDY, ORA, PLA, TAX, TAY, TXA and TYA.

Ignoring the confusing BIT opcode for a moment, the only two opcodes affecting the V flag are ADC and SBC — not surprising, since the two's complement overflow status which it represents is only important in arithmetic operations. These two also act on N and Z, so their flag-update subroutine includes the following line:

```
50  V=0: IF (RES AND 64) THEN V=1
```

...'overflow' relating to bit 6 of a byte — something else we'll leave until later. The remaining flag, the Carry, is acted on in three different ways, depending on whether an add, a subtract or a rotation is going on. For ADC and the multiplying rotates ASL and ROL, the carry is set if the result is larger than a byte can hold:

```
70  C=0: IF RES>255 THEN RES=RES–256: C=1
```

For SBC and the compares (CMP, CPX, CPY), the carry is set if the result is less than zero — the carry here represents a 'borrow':

```
90  C=0: IF RES<0 THEN RES=RES+256: C=1
```

The last group is the dividing rotates, LSR and ROR; for these the carry represents the remainder after the 'divide by two' operation:

```
110  C=0: IF (RES–INT(RES)) THEN C=1
```

...this line indicating the importance of integer arithmetic in our 'dummy-BASIC', since there should only be 0 or .5 resulting from the (RES–INT(RES)) operation.

There are, of course, other instructions which act on the flags — those which act directly on them, such as SEC and CLD. Those, along with the BIT instruction, we'll look at next time as we develop the 'dummy-BASIC'. For now, we can finish off this main group by constructing a complete set of subroutines for the flags, along with an 'index' for the relevant instructions:

```
10   V=0: IF (RES AND 64) THEN V=1
20   RETURN
30   GOSUB 10
40   C=0: IF RES>255 THEN C=1 50   GOTO 100
60   GOSUB 10
70   C=0: IF RES<0 THEN C=1
80   GOTO 100
90   C=0: IF (RES–INT(RES)) THEN C=1
100  Z=1: IF RES THEN Z=0
110  N=0: IF (RES AND 128) THEN N=1
120  RETURN
```

ADC	GOSUB 30	LSR	GOSUB 90
AND	GOSUB 100	ORA	GOSUB 100
ASL	GOSUB 40	PLA	GOSUB 100
CMP, CPX, CPY	GOSUB 70	ROL	GOSUB 40
DEC, DEX, DEY	GOSUB 100	ROR	GOSUB 90
EOR	GOSUB 100	SBC	GOSUB 60
INC, INX, INY	GOSUB 100	TAX, TAY	GOSUB 100
LDA, LDX, LDY	GOSUB 100	TXA, TYA	GOSUB 100

'LIST BASIC VARIABLES AND FUNCTIONS' by Dr. Mike Whittle

```
0235 A57B    LDA $7B       ;  START ADDRESS LOW
0237 8550    STA $50       ;  TO 50
0239 A57C    LDA $7C       ;  START ADDRESS HIGH
023B 8551    STA $51       ;  TO 51
023D A9FD    LDA #$FD      ;  SPACE BAR ROW
023F 8D00DF  STA $DF00     ;  TO KEYBOARD
0242 A90E    LDA #$0E      ;A LINE COUNT = 15
0244 8553    STA $53       ;  TO 53
0246 A90A    LDA #$0A      ;B LOAD 'LF'
0248 20EEFF  JSR $FFEE     ;  PRINT
024B A90D    LDA #$0D      ;  LOAD 'CR'
024D 20EEFF  JSR $FFEE     ;  PRINT
0250 C653    DEC $53       ;  DECREMENT LINE COUNT
0252 D009    BNE $025D     ;  TO E IF +VE
0254 A9EF    LDA #$EF      ;  SPACE BAR COLUMN
0256 CD00DF  CMP $DF00     ;D CHECK KEYBOARD
0259 D0FB    BNE $0256     ;  TO D IF NOT PRESSED
025B F0E5    BEQ $0242     ;  TO A IF PRESSED
025D A550    LDA $50       ;E CURRENT ADDRESS LOW
025F C57D    CMP $7D       ;  CHECK END ADDRESS
0261 D009    BNE $026C     ;  TO F IF NO MATCH
0263 A551    LDA $51       ;  CURRENT ADDRESS HIGH
0265 C57E    CMP $7E       ;  CHECK END ADDRESS
0267 D003    BNE $026C     ;  TO F IF NO MATCH
0269 4C74A2  JMP $A274     ;  RETURN TO BASIC
026C A005    LDY #$05      ;F COUNT 6 BYTES
026E B150    LDA ($50),Y   ;G PICK UP VARIABLE
0270 99AA00  STA $00AA,Y   ;  STORE IN AA/AF
0273 88      DEY           ;  DECREMENT COUNTER
0274 10F8    BPL $026E     ;  TO G IF NOT FINISHED
0276 8452    STY $52       ;  FF TO 52 AS 'NUMERAL' FLAG
0278 A5AA    LDA $AA       ;  VARIABLE NAME FIRST LETTER
027A 100E    BPL $028A     ;  TO H IF TOP BIT CLEAR
027C 48      PHA           ;  PUSH ONTO STACK
027D A946    LDA #$46      ;  LOAD 'F'
027F 20EEFF  JSR $FFEE     ;  PRINT
0282 A94E    LDA #$4E      ;  LOAD 'N'
0284 20EEFF  JSR $FFEE     ;  PRINT
0287 68      PLA           ;  PULL OFF STACK
0288 E652    INC $52       ;  CLEAR 'NUMERAL' FLAG
028A 297F    AND #$7F      ;H CLEAR TOP BIT
028C 20EEFF  JSR $FFEE     ;  PRINT
028F A5AB    LDA $AB       ;  VARIABLE NAME SECOND LETTER
0291 D002    BNE $0295     ;  TO I IF NOT NULL
0293 A920    LDA #$20      ;  LOAD 'SP'
```

```
0295 48     PHA           ;I  PUSH ONTO STACK
0296 297F   AND #$7F      ;   CLEAR TOP BIT
0298 20EEFF JSR $FFEE     ;   PRINT
029B 68     PLA           ;   PULL OFF STACK
029C 1018   BPL $02B6     ;   TO K IF TOP BIT CLEAR
029E A924   LDA #$24      ;   LOAD '$'
02A0 20EEFF JSR $FFEE     ;   PRINT
02A3 E652   INC $52       ;   CLEAR 'NUMERAL' FLAG
02A5 A920   LDA #$20      ;   LOAD 'SP'
02A7 20EEFF JSR $FFEE     ;   PRINT
02AA A000   LDY #$00      ;   COUNTER FOR ASCII
02AC B1AD   LDA ($AD),Y   ;J  PICK UP CHARACTER
02AE 20EEFF JSR $FFEE     ;   PRINT
02B1 C8     INY           ;   INCREMENT COUNTER
02B2 C4AC   CPY $AC       ;   TEST FOR ALL DONE
02B4 30F6   BMI $02AC     ;   TO J IF NOT
02B6 2452   BIT $52       ;K  CHECK 'NUMERAL' FLAG
02B8 101C   BPL $02D6     ;   TO M IF CLEARED
02BA A920   LDA #$20      ;   LOAD 'SP'
02BC 20EEFF JSR $FFEE     ;   PRINT
02BF A5AD   LDA $AD       ;   LOAD HIGH BYTE OF NUMBER
02C1 48     PHA           ;   PUSH ONTO STACK
02C2 0980   ORA #$80      ;   SET TOP BIT
02C4 85AD   STA $AD       ;   REPLACE IN AD
02C6 206EB9 JSR $B96E     ;   CONVERT TO ASCII AT 0100 ON
02C9 68     PLA           ;   PULL OFF STACK
02CA 1005   BPL $02D1     ;   TO L IF SIGN BIT CLEAR
02CC A92D   LDA #$2D      ;   LOAD '-'
02CE 8D0001 STA $0100     ;   STORE IN 0100
02D1 A900   LDA #$00      ;L  LOAD NULL FOR PRINT ROUTINE
02D3 20C3A8 JSR $A8C3     ;   PRINT ASCII FROM 0100 ON
02D6 18     CLC           ;M  PREPARE TO ADD
02D7 A550   LDA $50       ;   CURRENT ADDRESS LOW
02D9 6906   ADC #$06      ;   ADD 6
02DB 8550   STA $50       ;   REPLACE
02DD 9002   BCC $02E1     ;   TO N IF NO CARRY
02DF E651   INC $51       ;   INCREMENT CURENT ADDRESS HIGH
02E1 4C4602 JMP $0246     ;N  TO B FOR NEXT VARIABLE
```

Glitches

Red faces department!

We gaily printed a statement in last issue about Tony Parsons' renumber routine being error free – it was when Tony sent it here, but as most people will have discovered, it most certainly wasn't by the time our keyboard op had finished with it! (To be fair, typing ASCII on a non-ASCII typesetter keyboard isn't easy – on our now-departed CRTronic the simple statement A=B+C had to be typed as ◇f1◊A◇f8◊c◇f1◊B◇f8◊a◇f1◊C, so you can imagine what typical BASIC was like! I'm typing this on a standard ASCII terminal, sending it out to the setter via the Anvil code-converter – so it should be right this time...). Anyway, herewith Tony's corrections – including a number of lines that our op completely missed:

```
63090 H=PEEK(A+4): I=PEEK(A+5): K=PEEK(A+6)
63350 A1=D1: N1=N1+1: GOTO 63320
63400 FOR X=1 TO LEN(N$)-1
63412 X1=LEN(O$)-LEN(N$)
63413 FOR X=1 TO X1
63414 POKE (A+5-LEN(N$)-X),32: NEXT
63420 A=A7: GOTO 63081
63765 FOR X=1 TO 20000
```

Tony has found that this last line confuses some people. It is of course a 'dummy operation', saving values on the stack in the only way that our BASIC knows; the loop is jumped out of before completion, since no-one has enough memory to hold 20,000 lines of BASIC!

Red faces department II

Not so much a glitch as a reply, this one: several people took exception to my casual reference to *PC, PCW* and allied mags as 'the comics'. Why this should be taken as an insult I'm not sure, but the term is a reasonable one, since most of us devour their contents and drop them to one side in exactly the same way we treated (treat?) the *Beano* or *Private Eye*. Don't we?

The only thing which worries me is that some members may be taking altogether too serious a view of our interest in micros. The whole business of computing is insane anyway – poking around in the depths of multi-valued inverse logic – so we need those 'comics' not just to keep us up to date, but to reintroduce a sense of hilarity as well at the time you need it most: at two o'clock in the morning, when you're chasing yet *another* bug in your program!

Red faces department III

And another different kind of glitch – the late arrival of this Newsletter. Some members will know just how much work a newsletter of this kind involves; most, I presume, will not. Your editor has been somewhat overloaded recently with a little problem of commercial survival in a new venture – hence the delays in delivering this issue, which had to take a rather lower priority. It's not so much the production work – made much simpler now that we can take tapes and disks, and output them straight into the setter – but the editing work, the sorting of all the tiny notes and comments and queries you send in. It's now obvious that I cannot handle both aspects and still keep the Newsletter on time; so from next issue (August according to the schedule) I will be handling the production only, with Richard and George between them handling editing. And apologies to everyone for all the delays to date – we'll be back on schedule soon – promise!

```
 10 REM * MOTH ROUND CANDLE NOVELTY ROUTINE *
 20 REM * FOR UK101 32 X 48 WITH CEGMON *
 30 CL$=CHR$(26):B=57088:Z=54050:PRINTCL$;
 40 REM * SET UP CANDLE AND HOLDER *
 50 FORG=(Z+320)TO(Z+1152)STEP64
 60 FORH=1TO2:POKEG+H,187:NEXT:NEXT
 70 FORW=55199TO55208:POKEW,145:NEXT
 80 POKE55134,190:POKE55145,189
 90 POKE55138,207:POKE55141,210:POKEZ+258,222
 100 REM * RANDOM GENERATOR *
 110 X=16:J=1
 120 POKEZ,X
 130 K=INT(RND(1)*8)+1
 140 REM * CANDLE FLAME SIMULATED FLICKER *
 150 IFK=<4GOTO170
 160 POKE54180,32:POKE54244,32:POKE54179,178:POKE54243,177:GOTO180
 170 POKE54179,32:POKE54243,32:POKE54180,176:POKE54244,175
 180 GOSUB330
 190 REM * 'MOTH' RANDOM FLUTTER *
 200 ONKGOTO210,220,230,240,250,260,270,280
 210 POKEZ,32:Z=Z+1:X=237:GOSUB350:GOTO300
 220 POKEZ,32:Z=Z-1:X=239:GOSUB350:GOTO300
 230 POKEZ,32:Z=Z+64:X=238:GOSUB350:GOTO300
 240 POKEZ,32:Z=Z-64:X=236:GOSUB350:GOTO300
 250 POKEZ,32:Z=Z+63:X=238:GOSUB350:GOTO300
 260 POKEZ,32:Z=Z+65:X=237:GOSUB350:GOTO300
 270 POKEZ,32:Z=Z-63:X=236:GOSUB350:GOTO300
 280 POKEZ,32:Z=Z-65:X=239:GOSUB350:GOTO300
 290 REM * PROTECTION OF CANDLE BASE FROM ERASURE *
 300 IFZ<53250ORZ>55290THENZ=54242:GOSUB330
 310 GOTO120
 320 REM * FLUTTER SPEED *
 330 FORP=1TO10:NEXTP:RETURN
 340 REM * MOTH JUMPS UP IF NEAR CANDLE *
 350 IFPEEK(Z+128)=187ORPEEK(Z+129)=187ORPEEK(Z+127)=187THENZ=Z-512
 360 REM * MOTH DROPS DIAGONALLY IF CLOSE TO FLAME *
 370 IFPEEK(Z+64)=178ORPEEK(Z+65)=178ORPEEK(Z+1)=178THENGOSUB410
 380 IFPEEK(Z+64)=176ORPEEK(Z+63)=176ORPEEK(Z-1)=176THENGOSUB440
 390 IFZ>55131THENZ=55067
 400 RETURN
 410 W=1
 420 Z=Z+63:W=W+1:IFW>11THENRETURN
 430 GOTO420
 440 W=1
 450 Z=Z+65:W=W+1:IFW>11THENRETURN
 460 GOTO450

OK
```