OSI/UK User Group Newsletter

Vol.2 No.2 February 1981

Implementing colour

A disassembler □ disc notes on 65D and 65U □ and plenty more besides

Editorial

To moan, or not to moan?
Your editor put his foot in it in a big way when a large chunk of my none-too-friendly comments on the new C1 Series II were reprinted in February's *PCW* — with some unkind comments from Guy Kewney to boot! Fair enough, I should be more careful about how I open my big mouth (or pen), but I must admit I hadn't realised just how widely-read the *Newsletter* has become. So in case anyone read his piece and got the wrong idea, it *is* true that I don't like the C1 Series II — I feel it's moving in a dangerous direction towards the Tandy principle of 'bodge it together with patch-boards here and there'. I'm also annoyed that they still haven't fixed the garbage-collector, but then the cure on the 'new ROM' PETs is usually worse than the original problem — apparently it can take up to twenty *minutes* to sort string arrays into place on a garbage-collection call. I'm still annoyed about the keyboard's non-standard use of the SHIFT keys, but then neither Apple nor Tandy machines have lower-case at all without add-on boards.

But my moans are all relatively minor ones, considering the actual value-for-money of the kit. OSI kit usually works — which is more than can be said of the Trash-80 or the Pet, from what I'm told. The cassette interface is staggeringly reliable — again in comparison with the Tandy or Pet, whose interfaces are notoriously unstable. The disc systems work, which is certainly more than can be said of either Tandy's (personal experience here!) or Commodore's (though not, it would seem, of the excellent CompuThink unit). And the OSI and UK101 boards and board designs are robust enough to withstand the most outrageous hardware modifications that our members have made to them — unlike my colleague's sad experience with a Nascom-2, whose ultra-fine tracks flew off the board at the merest mention of a soldering iron. So when you add it all together, especially with the way the prices at the lower end have still gone *down* when everyone else's have gone up, what I wrote in the first issue of this *Newsletter* still holds good today: that for anything other than absolutely standard off-the-shelf applications with off-the-shelf packages, OSI kit is probably the best value for money around.

Stirring times?
Apart from Guy's 'stirring' above, there's plenty much else moving around in the Ohio Scientific field. For one thing, we've found out a bit more about M/A-Com, the firm who took over Ohio Scientific. They're a large conglomerate — though federation might be a better word — of electronics companies, specialising in microwave and digital communications in particular. They're also apparently rated as 'one of the best growth prospects in the US' — which might or might not be good news. In any case, it seems that their interest in OSI was two-fold: partly to guarantee delivery of OEM boards, and partly because they are busy developing the cable-TV market in the States, and have, it seems, this happy idea of selling a computer package along with cable-TV to give a data-access system rather along the lines of Prestel over here. As a result, it seems they want to push the small-computer market (the C1 and C4 series), rather than let it fade away as we feared they would. Which, again, might just be good news. Just whether this will lead to a change of attitude in the factory is not certain; we hope so, because the old management had a deliberate

policy of ignoring advice, pleas or anything from the outside world — one result of which is the C1 Series II.

Other things are moving here. At Compec, we met up with Bob Cross and Alan Davies from the fledgling Ohio Scientific (UK) Ltd — not part of OSI itself, but a subsidiary of our old friends(?) American Data, the European distributors. At the time we felt it was a big mistake on their part — yet another attempt by American Data to push its absurdly high price structure on the UK market, with Bob and Alan left playing pig-in-the-middle. But we're delighted to find that it hasn't worked out that way at all. The company has now found itself premises in Langley, near Slough, and is setting up what promises to be a proper import and service organisation, dealing direct with the dealers as a distributor rather than with direct sales to end-users. There was plenty of stock actually in the building when I went there, including the hard-disc and network systems — they clearly intend the big systems to be their major line. And although they are fully owned by American Data, they've made it quite clear to AmData's David O'Brien that the British market simply cannot afford those prices — and that fact at last seems to be getting through. Another point from David O'Brien is that he's angling for a percentage of the vast OSI advertising budget (colour back pages on virtually every American computer mag, every month, and plenty more besides) to be allocated to the European market — which may well lead to a major upturn over here. All of which sounds very interesting, and very hopeful.

A new flexibility for OSI systems — and a problem of space
BASIC-in-ROM, Assembler, ExMon. The triumvirate has been with us for a long while. So its interesting to hear of a number of developments in the pipeline. Already here is a 'Toolkit' of BASIC support commands (rather like the PET's 'Programmer's Toolkit'), from Premier Publications — we'll try to get a review for next issue. On word-processors, we already have WP-6502 — much-beloved or much-cursed, depending on your viewpoint! — and on the way is another very interesting-sounding one, again from Premier, to be available in either tape or PROM form. On languages, we know that FORTH has been implemented on the UK101, and a PROM implementation for all systems should be on the way soon; a PROM implementation of PILOT is due in April; and we've even been told of an 8K mini-Pascal in PROM, which should be on the market in another few months.

Notice that most of these are in PROM — the price of 2716/2516 type PROMs has dropped way below that of RAM, to as low as £4 retail, and possibly £2.50 in largish quantities wholesale. Which makes for easy pirating — another of our more serious peeves and problems, but that's another story. But what it does mean is that there are soon going to be heavy and conflicting demands on the memory map — a problem that has already hit the Pet badly. Where do we place all this ROM software? One solution I've been advocating is that we divide the software between support-software for languages (like the Toolkit) and languages themselves (including 'language'-like applications software such as word-processors); and place *all* the languages in *exactly* the same place, since we can't really use more than one at a time.

This would need some kind of 'language card', with hardware or software switches to select which language is to be used at any one time — but that's hardly a difficult design proposition, since all it would need is a repeated set of exactly the same address and data lines, all connected to the 8K socket already provided in the BASIC-4 socket on all the OSI BASIC-in-ROM systems. All the software would be assembled with addresses in the range $A000–$BFFF, switching between them via the monitor, rather as on the Apple. The same space could even be used for RAM rather than PROM on disc systems.

Much the same could be done for the support software — using the $9x page (36K–39K), again switching one or another set on or off via the monitor. A nice advantage of this approach is that power supply requirements would be lower — you wouldn't need to power-up a chip set unless the set was enabled by the software/hardware switches.

Any takers? And anyone willing to help us work out a standard, to avoid the confusion that's now affecting Pet users?

Documentation Corner

One use of WAIT

WAIT is a somewhat under-used instruction; here is a brief routine from *Bob Bonser* to illustrate one use, as a 'password' controller in BASIC-in-ROM.

```
100  POKE 530,1: REM disable CTRL-C
105  POKE 57088,4: WAIT 57088,8
110  POKE 57088,16: WAIT 57088,64
115  POKE 57088,16: WAIT 57088,128
120  POKE 57088,32: WAIT 57088,8
125  POKE 530,0: PRINT "O.K."
```

The routine when run will not reach line 125 until the four specified keys have been pressed in the correct order — with the values here, as for a C2/C4 type (non-inverted) keyboard, the four keys are *N E W RETURN*. For a C1-type (inverted) keyboard, the values shown above must all be subtracted from 255: line 105 would be POKE 57088,251: WAIT 57088,247.

[*Ed.* — has anyone been using WAIT extensively? As usual, it is very poorly described in the standard documentation — particularly the use of the exclusive-OR part of the instruction, which Bob hasn't used here. Any offers?]

Adding your own commands to ROM BASIC

Dr A.H. Kearsey wrote in to remind us of a couple of articles in *Micro*. One, in the June '80 issue, was by *Edward H Carlson* and called *Put your hooks into OSI BASIC*. (OSI followed up the idea in their *Small Systems Journal* in the September '80 issue of *Micro*, with an article on adding commands to 65D BASIC). The idea in Carlson's article was to add a 'compare' test to the $BC character-scan routine [see Vol.1, Issues 1-3 of the Newsletter for details of the $BC routine — *Ed.*], to look for his '%' and '#' user-routine markers. The additions to the routine slowed BASIC a little, but not enough to hurt much. The other article, in the Nov. '80 issue, showed much the same idea applied to the 'OK' message call on BASIC's warm-start and error-recovery, which jumps via $0003-5 — the disadvantage of this latter method being that BASIC prints out the error-message *before* doing the 'OK' call.

A third method is to attach a parser (command interpreter) to the CTRL-C vector on C1/Superboard/UK101 and CEGMON-based systems. The CTRL-C check routine is called between *each* statement, regardless of whether the POKE 530,1 disable is set — see the TRACE routine in this issue's part of the 'Basics of machine-code' series. This would run quite a bit faster than the $BC patch, since the test is called between each statement rather than each character; but it does mean that user-defined commands would have to be within statements on their own (either on a line of their own, or within the usual ':' delimiters).

We'll look at these ideas in more detail next issue; but has anyone tried these routines to date? And any suggestions for worked examples of new BASIC commands that you'd like to see in the Newsletter?

Hardware 'patch' for the 'garbage collector' problem

Michael Lewis writes: I was very impressed by Dick Stibbons' concise description of the garbage collector bug and its solution in the Newsletter Vol.1 No 4. I have used the following hardware modification to apply this solution to my 600 Superboard and I hope that this account will be useful to other users of OSI Basic-in-ROM.

The trick is to swap the chip select lines of the Basic 3 ROM and a spare 2K block of RAM so that the RAM appears at address $B000 and Basic 3 appears at $1800 (The top 2K of an 8K system). Basic 3 may then be copied back into RAM at its normal address and modified by the monitor to cure the bug.

The gates are needed because the Basic 3 ROM is selected as a single 2K block and the RAM is selected as 2 1K blocks. It is also necessary to cut the p.c. track to pin 5 of U15 and provide a link from that pin to ground to enable the CPU to write to the Basic ROM area. The 100Ω resistor put across the break restores the logic conditions without further soldering when the ground link is removed. Removing the link actually leaves the new Basic RAM write-protected.

The connections are made by unplugging the chips and bending their chip select pins outwards so that they do not make contact when the chips are replaced in their sockets. The connections from the additional 7400 chip to the board may be made by pushing wires of a suitable gauge into the appropriate pins of the IC sockets before the chips are replaced. To avoid soldering to the pin on the ROM one could use an individual pin from a broken-up IC socket..

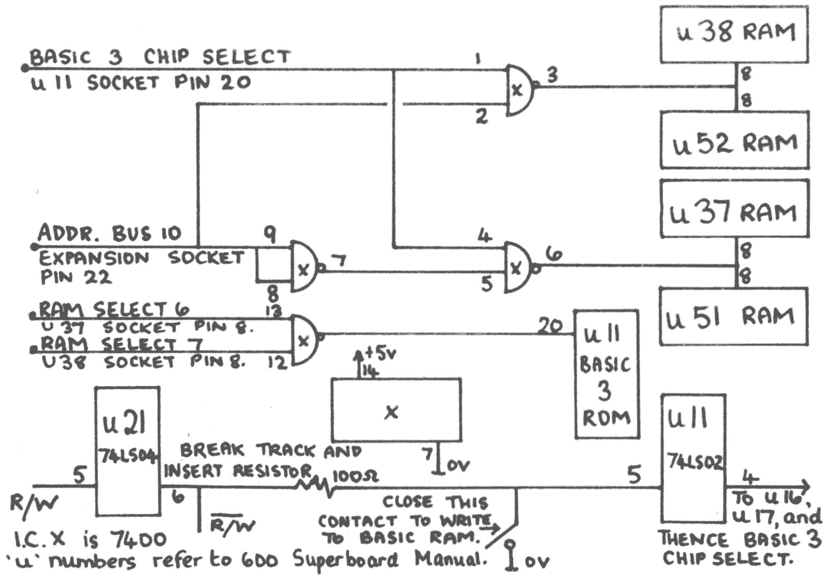
If CEGMON is available the contents of Basic 3 may be moved easily. Otherwise it will be necessary to write a machine code move routine, or load Basic 3, previously stored on a cassette. There is a misprint in the modified program in line 2 on page 20 and also, I think, line 20, page 21 but the meaning is clear. When Basic 3 has been connected it may be saved on cassette as it will have to be loaded (with IC15 pin 5 grounded) every time the computer is switched on.

The demonstration program on p13 [Vol.1 No.4] clearly shows the 'garbage collector' routine working in my Superboard. Without modification the program latches up immediately but with the modified routine it accepts 9 key strokes before giving OM ERROR IN 50 and returning to Basic with OK. Incidentally you *can* put the four POKEs which set the pointers in a program, provided that the program uses no variables. Adding:

```
100  POKE - -: POKE - -: POKE - -: POKE - -
110  END
```

and typing RUN 100 (to set positions)
RUN (to run program)

has worked satisfactorily for me.



The BASICs of machine code

Part 2: the disassembler

Last issue we looked at the basic concepts of working with machine-code: the opcodes and their variations, hexadecimals values, the principles behind machine-code program design, and linking machine-code routines to BASIC. As this series progresses we will go into these in more detail; but for this issue we prefer to work on things in another way, by giving you a means of seeing how other programmers work. This is one of the main tools in the machine-code programmer's tool kit — the disassembler.

One is built into ExMon (the OSI Extended Monitor), namely its 'Q' function; but not everyone has a copy of ExMon. So here is a complete BASIC listing of a machine-code disassembler, adapted from one originally published by *D.N. Sands* in the November 1979 issue of *Practical Computing.* (The main alterations from the original are in adapting the formatting to match a Superboard rather than a Pet; in relocating the subroutines at the beginning, to speed up execution; and in simplifying the main look-up table).

The routine is a fairly simple one, in that it only prints out the address, the opcode and the range of any branch — unlike ExMon's routine, it does not print out the hex values of the actual code. But it will serve to illustrate the point, and as a basis for your own versions.

When RUN, it jumps to the execution loop in lines 1000-1130. This calls for a start address (A$) in *hexadecimal,* which is then converted to decimal (A) by the short loop in 1010-1040. The (Q>9) statement returns a zero if false, –1 if true: thus the (ASC–48) values of the A-F digits are automatically trimmed to decimal 10-15). The next loop prints out a number of lines of code — which could be up to three times as many bytes — the number of lines required (NL) being called for in 1050. Twenty or so lines can be displayed at one time, of course. Answering '0' to the call restarts with a call for a start address (1060).

Line 1080 sets up the byte counter (N) to its default value of 1, then prints the current address in hexadecimal, and a small tab-space. 1090 collects the opcode value (D) at the current address, and then the possible low (L) and high (H) values of an operands in the following two bytes (if they aren't valid as operands they are ignored — but the routine always collects them, for simplicity later). Line 1100 produces two variations on the opcode value: V to disassemble opcodes with addressing-mode variations; and F, to sort out any resulting addressing modes. Respectively, they mask or save bits 2, 3 and 4 of the opcode. Lines 1110-1130 scan the main opcode table, doing the actual disassembly; bump the address counter (A) by the byte counter (N); get the next line; and call for another batch of lines when finished.

The main table loop is in four parts: 400-640, 650-720, 730-750, and 760-960. The first deals with all the simple one-byte instructions: those that set or clear flag bits, those that operate directly on registers, and the system-jump instructions BRK, RTS and RTI. Because these are all one byte long, with the address mode implicit in the instruction, each line RETURNs directly to the calling loop, without looking at either of the following bytes.

The next section deals with the 'branch' instructions — the 'IF:THEN' statements, as described last issue. These do short-range jumps according to the setting of various flag bits — the Carry or C bit (BCC: off and BCS: on), the zero Z (BNE off, BEQ on), overflow V (BVC off, BVS on) and sign N (BPL off, BMI on). On finding one of these, the routine jumps to the branch calculator in 130-150. These are all two-byte instructions, so the value picked up in H for a possible third byte is 'thrown away' and H then used to store the branch distance +127 to –128. The jump location is printed as a hexadecimal address (the location being the *next* current address plus the branch distance: A+H+N), followed by the branch distance, in brackets.

The main table continues with the three three-byte jumps, JSR, JMP Absolute and the rarely-used JMP Indirect. Their jump addresses are collected from the two following bytes, and printed out as a hex address via the routine in 50-120. This converts the decimal combination of the two bytes into four hex values, which are converted as their ASCII letters by the routine in 20-30 (the H$ conversion is the inverse of that in line 1020).

The table continues with the versatility instructions, all of which have addressing-mode variations: for this reason the scan looks at the stripped opcode value V rather than the full value. On finding a valid opcode the routine jumps to the addressing-mode interpreter at 160-260 — entering it at various places in order to handle some of the variations possible between the opcodes and addressing modes. (Not all of these are valid anyway — disassembling ASCII text produces some weird ones like EOR (Ind),Y which don't exist in the 6502 language at all!) If the instruction is two bytes long (as shown by N in the addressing-mode table), H is 'thrown away'; otherwise the two are combined as for the jump instructions above. In either case, the address is printed as a four-digit hex value.

Line 970 at the end of the main look-up table deals with invalid 'instructions', which often turn up when trying to disassemble look-up table tables such as those at the beginning of BASIC-in-ROM ($A000-$A1A0). Lines 270-290 deal with these 'duds' — line 280 is a safety device to prevent screen-clear or backspace under CEGMON, but otherwise is the ASCII value of the 'dud' is printed, making it easier to interpret text look-up tables.

```
 10 GOTO 1000
 19 REM convert hex to ASCII letter
 20 H$=CHR$(48+Q–7*(Q>9))
 30 PRINT H$;: RETURN
 49 REM decimal to hex
 50 IF N=3 THEN L=L+256*H
 60 Q=INT (L/4096): GOSUB 20
 70 L=L–Q*4096
 80 Q=INT(L/256): GOSUB 20
 90 L=L–Q*256
100 Q=INT(L/16): GOSUB 20
110 Q=L–Q*16: GOSUB 20
120 RETURN
129 REM relative addressing
130 H=L: N=2: IF L>127 THEN H=L–256
140 PRINT"  ";: L=A+H+N: GOSUB 50
150 PRINT TAB(16);"(";H;")";: RETURN
159 REM code variation — addressing modes
160 IF F=20 THEN PRINT " 0–pg,Y";: N=2: GOTO 300
170 IF F=0 THEN PRINT TAB(17)"#";: N=2: GOTO 300
180 IF F=8 THEN PRINT " Acc";: RETURN
190 IF F=12 THEN PRINT " Abs";: N=3: GOTO 300
200 IF F=8 THEN PRINT TAB(17)"#";: N=2: GOTO 300
210 IF F=4 THEN PRINT " 0–pg";: N=2: GOTO 300
220 IF F=0 THEN PRINT " (Ind,X)";: N=2: GOTO 300
230 IF F=16 THEN PRINT " (Ind),Y";: N=2: GOTO 300
240 IF F=20 THEN PRINT " 0–pg,X";: N=2: GOTO 300
250 IF F=24 THEN PRINT " Abs,Y";: N=3 GOTO 300
260 IF F=28 THEN PRINT " Abs,X";: N=3: GOTO 300
270 PRINT "Undef ";
280 IF D<32 OR D>127 OR D=95 THEN PRINT".";: GOTO 300
290 PRINT CHR$(D);
```

changes the incoming character to a 'bell' code, ASCII 07 — which would ring the bell on the Teletypes for which BASIC-in-ROM was originally designed, but on OSI systems prints that mysterious half-battleship symbol on the screen.

The $A386 'get a character' subroutine is fairly simple (the exact layout of the routine varies on later UK101s). It first collects a character via the input routine at $FFEB, via the user-defined vector at $0218 (except on standard C2/C4 systems), and finally via the actual input routine in force (usually $FB46 on CEGMON, for example). Then follow a string of NOPs, because the designer presumably intended to add a few more features but never got round to it; and a check for CTRL-O, the 'disable printing' code, which if found inverts the contents of $64 (the 'allow printing' flag, checked each time a character is output to the screen). With this done, the routine returns to whatever called it.

That is one place to start: another pleasant tangle is at $BF2D, the original screen handler, which is the same in both C1s and C2s, and thus has to decide which system it is in before it can start!

Wander around BASIC-in-ROM, and your monitor, making notes as you go on — it's the quickest way there is of understanding how to tackle machine-code programming.

One of the items we need to look at in some detail is the input/output 'vectors' of the C1, UK101 and all CEGMON-based systems. These allow you to define where any character is to come from or go, what BASIC is to do between the execution of each statement (the CTRL-C vector), and how to set up LOAD and SAVE in BASIC. For example, this allows you to PRINT to a parallel printer in BASIC, using one of the add-on PIA boards for the smaller machines, or the CA-12 board on the bus-based systems. These 'vectors' will be the main item for next issue; but for the moment we can use one of them to add a new function to BASIC, namely a TRACE, to print out the line number of the statement just completed (see Vol.1 No.3 for the BASIC execution loop).

Between each statement, BASIC checks the keyboard to see if CTRL-C has been pressed, to allow the user to break into the program. It does this regardless of the CTRL-C disable (POKE 530,1), by the way — that is checked by the final routine in the monitor ROM, not by BASIC. If CTRL-C *is* pressed, the check routine has to print out the current line number (BREAK IN 1234). We can use the subroutine which does this for our own purposes, calling it between each statement as a line-trace function, rather than only once on a break. By writing a short routine to call that subroutine, and then do the CTRL-C check, and then using the 'vectors' to call this routine instead of the usual one, we have our TRACE function.

The following routine is an adaptation of the CEGMON 'trace' routine for standard C1/Superboards, and probably for old-monitor UK101s — New Monitor systems will need to change the start location and CTRL-C jump. The routine is stored in page-2, and *must* start at the same low-order address as the ROM routine, or else BASIC would get lost between the two POKEs required otherwise to set up the routine; it is turned on by pointing the high-order byte of the CTRL-C vector at it. The subroutine called is at $B953, and the 'type mismatch' flag must also be set for string-printing. The routine is placed at $029B, 667_{10} as follows: LDA #$FF, STA $5F, JSR $B953, JMP $FF9B; or, in BASIC:

```
10 FOR X=667 TO 676: READ R: POKE X,R: NEXT
20 DATA 169, 255, 133, 95, 32, 83, 185, 76, 155, 255
30 REM turn 'trace' on with POKE 541,2
40 REM turn 'trace' off with POKE 541,255
50 NEW
```

Disc Notes

At last a large chunk of notes on disc systems, mostly compiled by *Richard Elen.*

Backspace on 65U V1.2

This change was published by OSI in their Technical Newsletter number 28, and has the advantage that it also enables you to use lower-case as well on a polled-keyboard system (the section for $3BE4 onwards).

In addition, if you change $3C61 onwards to 4C,7F,3C you can get rid of the flashing cursor on video systems, if such things annoy you.

If you do the backspace mod, note that it's worth keeping this listing, because if you want to use CEGMON 'D' (for disc systems — now well on its way) you'll have to take this backspace out again!

The CHANGE 'conversation' below will allow OS-65U to erase each character from the screen as it is deleted. Formerly 65U, like BASIC-in-ROM under SYNMON, echoed an underline for each character deleted.

```
RUN"CHANGE","PASS"
DISK CHANGE UTILITY
MODE: HEX(H), DEC(D)? H
UNIT? A
ADDRESS OFFSET? C00
ADDRESS? 055E
0000055E  07 ? 08
0000055F  F0 ? .
ADDRESS? 0560
00000560  14 ? 29
00000561  C9 ? .
ADDRESS? 0574
00000574  D7 ? 15
00000575  E0 ? .
ADDRESS? 058A
0000058A  EA ? A9
0000058B  EA ? 08
0000058C  EA ? 20
0000058D  EA ? EE
0000058E  4A ? 0A
0000058F  90 ? A9
00000590  F6 ? 20
00000591  AD ? 20
00000592  01 ? EE
00000593  FC ? 0A
00000594  EA ? A9
00000595  EA ? 08
00000596  EA ? D0
00000597  EA ? B4
00000598  29 ? .
ADDRESS? 3BFA
00003BFA  EA ? 20
00003BFB  EA ? 39
00003BFC  20 ? 3C
00003BFD  39 ? 24
```

```
00003BFE  3C ? CA
00003BFF  8E ? .
ADDRESS? 3BE4
00003BE4  C9 ? C9
00003BE5  20 ? 08
00003BE6  30 ? F0
00003BE7  17 ? 16
00003BE8  C9 ? C9
00003BE9  80 ? 20
00003BEA  10 ? 30
00003BEB  13 ? 13
00003BEC  9D ? X
OK
CLOSE
OK
```

Labelled GOTO and GOSUB in 65U

For a change, OSI have produced a real winner in OS-65U. It must be one of the most versatile disc operating systems around! One of the functions of its BASIC is that you aren't limited to a numerical constant (line number) in GOTO and GOSUB statements. You can also GOTO or GOSUB to a variable, or even an expression. All you need to do is to give your variable a value, and then GOTO or GOSUB to it; similarly, BASIC can evaluate any sensible expression after the command. This may be very useful: it occurs in the DMS packages, for example, as a rather bizarre ON:GOTO.

Note that the same is not the case with 65D, whose BASIC is little more than a jumped-up 9-digit version of BASIC-in-ROM.

Input and output masking in 65D

Like BASIC-in-ROM, 65D's BASIC masks out control characters and graphics on keyboard input — you can type them, but BASIC ignores them. 65D also has a similar output mask. These are simply altered, either on the disc itself, or probably more sensibly in BEXEC* or the program that needs them.

Input masking: Location $0566 (1382$_{10}$) normally contains 32$_{10}$ — the lowest acceptable ASCII input value (here a 'space'). You can drop it down as low as 1, but remember that if you're loading from cassette, it should be set to at least 11$_{10}$, to avoid problems with LINE FEED giving you a syntax error after every line (unless you want a 'cassette view'). Location $056A (1386$_{10}$) similarly contains the highest allowable ASCII input value.

Output masking: Location $0B0B (2827) contains $7F — you can change this to $FF (255) if required, to input graphics characters and others.

Cassette loading into 65D

About twenty people sent in solutions to Tony Restall's problem with loading from cassettes — thanks, everyone! There are several answers to the problem, of course. One is to set the I/O bits to handle the ACIA. Bit 0 listens to the ACIA (the 'console device' on serial systems), while bit 1 looks at the keyboard. DISK!"IO 03" therefore, will set both bits 0 and 1, and it will accept input from either 'device', allowing you to get control back after the LOAD. You *can't* do this with INPUT#(Device Number) direct from BASIC, because you can't alter a program via INPUT. The I/O Flag bit settings are on p.4 of the User's Guide appendix in the 65D manual.

Another solution, also in the manual (p.8 in the appendix), is POKE 8993,1. This POKEs the I/O distributor input flag directly. The output flag, by the way, is at 8994$_{10}$. These methods will cause control to return to the keyboard automatically if a Syntax Error is encountered, so it's important to wind the tape leader past any 'glitch' characters before starting the load. The difficulty with this method is that there *must* be a syntax error before control returns to the keyboard — a couple of members have commented on the need to hammer their too-good cassette drives in order to provoke the essential 'glitch' at the end of a load!

The real pity, of course, is that you can't set the monitor's old LOAD flag for BASIC-in-ROM. You can't because the usual monitor isn't 'in service' under 65D, and even if it was, the flag is in Page 2 of the memory and would scramble the 65D BASIC keyword address-table. The disc version of CEGMON, however, will allow you to turn on the flag with a POKE, as the flags are moved well out of the way. That way, you can POKE the flag, the cassette will load, SN ERRORs and all, and you get out of the LOAD with the space bar in the usual way for BASIC-in-ROM.

Examining files under 65D

This quick fiddle is a really easy way of looking at the contents of a file in 65D without using tedious BASIC utilities and losing your program in the process. It's best on a system with a 540 board, but even on a C1 the results are useful. Simply CALL the track and sector you require into the screen memory! You'll get a lot of racing-cars (nulls) plus ASCII where there is some. To do this, EXIT from BASIC, and enter the CALL command via the kernel:

```
EXIT
A*CA D000=TT,S
```

— where TT is the track number (two digits!) and S is the sector number. This is also useful for rapid printing to the screen: prepare the page with a program full of PRINT statements, including the command DISK!"SA TT,S=D000/8" after the screen has been prepared. You can play safe and create with the CREATE utility a non-existent file name in the Directory, covering the tracks you're using for this purpose, so you don't inadvertently over-write the display tracks later. Then, a call to the screen as shown above will give you a really instant display. If you SAved a blank screen this way, you'd give yourself a single-command screen-clear.

If you're using colour on a 540 board, this method is very useful for calling up instant colour backgrounds, formatted for a game, or whatever. Simply hit the BREAK key, and use the monitor to create the colour display page (remember to set $DE00 first to display the colour memory) in the region $E000–$E7FF. (CEGMON's 'text entry' and 'move' commands are very useful for this). When the page is completed, type .2A51 G to re-enter the DOS, and SAve the colour memory, in the form SA TT,S=E000/8. The monitor may have crunched BASIC and/or page-0, so reboot after this. A line in your program of the form:

```
100 DISK!"CA E000=TT,S": POKE 56900,V
```

— where V is either 5 (for 64×32) or 4 (for 32×32 display) — will bring up your SAved colour screen.

Indirect files in 65D

The Indirect File capability in 65D is very useful for merging programs with different line numbers, or assembly files. (It also appears in 65U, and would seem to be a way of transferring files between the two systems — has anyone done this yet?). The best description OSI give on how to use them is in the C4 Owners Manual, a rather better effort than the usual. The version on p.61 of the 65D manual is a bit garbled, but essentially correct. One thing to note, however, if you're a UK101 user, is that CTRL-X is used to load the contents of the indirect file into the workspace. On the UK101, the input masking is such that you can access the lower graphics characters directly in ROM BASIC. One of the more useful of these is a '£' sign, which just happens to be CTRL-X. If you even so much as *whisper* CTRL-X at a Compukit keyboard when the system is running under 65D, the whole thing will scramble! The screen will be filled with ahuge load of garbage, mainly out-of-buffer-space characters, and terminated by a Syntax Error. This means that your roulette program, or whatever, shouldn't use the £ character from the keyboard (you won't be able to), although you can happily use PRINT CHR$(24). If you're transferring a program from cassette into Disc BASIC, and you have embedded £ characters in the text (as opposed to CHR$(24)), you'll have *real* problems — whenever the offending character is encountered, the screen will fill up with rubbish followed by a Syntax Error. If you're quick, and stop the cassette, you *may* be able to resume LOADing after you've re-POKEd the input flag, although that line will be missing. Unfortunately, CTRL-X is decoded separately from the rest of the keyboard characters, so POKE 1382 (see above) will not help you. Your best solution is either to LOAD the program under ROM BASIC, edit out the embedded £ characters, re-SAVE and re-load under 65D (tedious); or to write a short machine-code routine attached to the input vector to substitute another character for the '£' ($ perhaps?). This should be of the form:

```
START     CMP #$18   ;  point the input vector at START first!
          BNE OUT    ;  #$18 is '£' — if not, pass char unchanged
          LDA #$xx   ;  xx is the substitute for '£'
OUT       JMP INPUT  ;  INPUT is where the input vector used to point
```

Similar things may happen with '[' (open square brackets) — in this case, your program so far will be stuffed into the indirect file, but little damage should be done (unless it takes too long to do it). Make sure the indirect file addresses are set somewhere outside the workspace you need, to be on the safe side.

65D on C1E and some other 32×48 conversions

A note from *Justin Johnson*: For those people who have done the C1E conversion the keyboard is no longer inverted, this prevents the standard C1 disc operating system from booting up. The correction is the press 'D' and wait for the screen to clear, then BREAK and 'M'. Change locations $24A1 and $24A9 from FF to 00. Go to $2A51 and press G [to warm-start the DOS] and answer 'BASIC' to the *A** prompt. The system will then work, the correction can be made permanent using the zero-page copier utility. The screen correction is given in the Elcomp *First Book of Ohio Scientific, Vol.1* (p.148-150), however where their correction says change $65 to $48 the C1E needs $40.

Diskette allocation: 65D

If you have the 65D manual, and nothing else, you'll notice that p.5 of the appendix gives you a layout for where everything is on the disc. This is for 8in discs only, and 5¼in discs are largely different. Here they are for comparison, showing track, tracks (such as 2–6) or track-and-sector (such as 12/1).

5¼in	*8in*	*Contents*
0	0	OS-65D (Bootstrap format, loads into $2200 up for 8 pages)
1/1	1/1	Rest of 65D: loads into $2A00 for 5 pages (8in)/8 pages (5¼)
13/1	1/2	Track Zero Read/Write utility: loads into $0200
2–6	2–4	9-digit Microsoft BASIC
7–9	5–6	6502 Resident Assembler/Editor
10–11	7	Extended Monitor
12/1	8/1	First page of Directory
12/2	8/2	Second page of Directory
12/3	8/3	Overlay page for 9-digit BASIC
12/4	8/4	PUT/GET overlay for 9-digit BASIC
14-39	9-76	User programs and 65D BASIC Utility programs

Note the interesting differences!

DEALER NOTES

Official dealers again — and that's official!

It's been a long time since any of the OSI dealers in this country have had 'official dealer' status. No-one would buy from Ohio's exclusive European distributors American Data because of its pricing policy, based on the infamous 'dollars equals pounds' equation; and because that distributorship was exclusive, no-one could buy from Ohio either. For realistic prices the dealers either had to buy on the American wholsesale market, bringing the kit in 'by the back door'; or else, like Comp, ignore minor matters like copyright, and manufacture their own versions. This meant lower prices for us, but it also meant no factory back-up for dealers or users; and on a few occasions this has produced unfortunate results — lack of anything but the most rudimentary documentation being an obvious case in point.

But suddenly everything has changed.

On one side, AmData's exclusivity no longer applies. *Watford Electronics* are now official dealers for the 600 series boards (Superboard, C1 and the disc add-ons), buying direct from the factory as a distributor-dealer — a fact reflected in their very low price of £149 (ex. VAT) for the standard Superboard. *Steve Holmes* is in charge of Watford's new specialist computer section, which also handles the Video Genie system. Watford are also brewing a series of add-on boards using the 40-line expansion socket — no further details as yet, but there are certainly hints of some very interesting developments there.

On the other side, American Data have changed not just their face (see the Editorial this issue) but their policy as well. *Alan Davies* and *Bob Crook* (not Cross! — a confusion with Ohio's Jim Cross) have finally convinced AmData's David O'Brien that the UK market simply won't buy a Superboard at £185 — AmData's former recommended price. The 'official' price is now at last what it has been in practice for some time, namely £159. The 'official' prices of the other kit like the 610 board have come into line with current practice too — so congratulations are due to Alan and Bob for injecting a long-overdue note of realism into AmData's British operations. The dealers are understandably still a little cautious, but it certainly seems that we have an 'official dealer' network again.

The price of the big systems — C2-OEM upwards — will probably rise a little, but this time with good reason, for Ohio Scientific (UK) are developing proper back-up facilities at Langley. They are not such much anglicising as re-writing the DMS program suite; they've already arranged full field-service contracts with *Systems Reliability* for C3-OEM upwards (and also, we believe, schools systems); they've at last gathered together some real documentation for all OSI kit including that for the MPI mini-floppy drives used in the C1P-MF [we have a copy if anyone needs it now — *Ed.*]; and they're organising 'image' advertising for the entire Ohio range.

All the signs of a proper dealer network and dealer support, based around Watford Electronics and Ohio Scientific (UK) — and that's something we're very glad to see.

BASIC 3 and the 'garbage collector' bug

Last issue we commented that no-one had actually got round to producing a PROM with the User Group's patch in it: we're glad to note that this has changed. As can be seen elsewhere in this issue, at least *two* firms are doing it: Martin Spalton's new PROM programming service, and Mutek. Martin is doing a special offer to Group members of £10 (no VAT) for a 2716 PROM and installation instructions; Mutek offer theirs along with another 2716 with a replacement for BASIC 1 (with the input masking removed to allow graphics entry), at £13.80 (inc. VAT) for the *pair*, to Group members (£17.25 inc. VAT to non-members). Aardvark also do a version in the States, for $19.95, but we don't know which fix they've used — we've been sent one by a member, but haven't had a chance to look at it yet!

Great products from Mutek

BASIC 1 & 3

Two replacement PROMs for all OSI and UK101 BASIC-in-ROM systems:

BASIC 1 removes the input 'mask' to allow direct entry of graphics from either keyboard or cassette; corrects error-messages to two-letter codes; and replaces the unused 'NULL' command with a machine-code 'CALL' instruction from BASIC.

BASIC 3 contains the patch for the string-handling 'garbage-collector' bug, as described by the OSI/UK User Group.

Special price to User Group members: **£12.00**+VAT
for the *pair*, including notes and installation instructions.
To help us maintain this low price, please send a stamped addressed 'Jiffy-Bag' envelope with your order.

8K memory/PIA board

A professional-quality expansion board for all Superboard and UK101 systems.
★ 8K of reliable static memory (2114L3) ★
★ Two-way, 16-line parallel port (6821 PIA) ★
★ Fully buffered — boards can be linked ★

Assembled, tested and guaranteed: **£65.00**+VAT
Ribbon-cable and plugs: add £8.00+VAT

Serial to parallel converter

A converter for all systems, with links to allow for differing word lengths and formats.
300–9600 baud RS232 serial input — Centronics-compatible parallel output.

Price including case, cables and documentation: **£40.00**+VAT

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

We still don't know whether CompShop have done anything about the bug. Our member *Roger Cuthbert* sent them a complete listing of a fix well over a year ago, and *Dick Stibbons* did so at the same time as sending his version to us. But so far, no sign of it. The UK101 was temporarily out of production a month back when they ran out of ROMs; their new batch was due in at the beginning of February, and we don't yet know what's in them.

There still remains the tricky point of copyright — technically, anyone selling copies of the BASIC would be in breach of OSI's and Microsoft's licences. It seems unlikely that either would do anything about it though, partly because it's patching their mistakes, and partly because it is sold as a replacement rather than as a new copy. One solution would be actually to do it as an exchange, to prove that it is a replacement — so don't be surprised if it has to be done this way!

Hardware offers

A string of offers on printers, printer mechanisms and disc drives from our member *David Hardman*:

Further to our recent phone conversation, I should like to confirm the various points raised:

i) We could supply GP80 graphic printers [ie Seikosha unihammer unit] at a special discount to members of £220 (RRP £250).

ii) We could supply FP1500 25 chs/sec daisy wheel printers at £920 each, provided we could batch together an order for say 10 units, again at a special discount price to members (RRP £1085).

iii) We could supply double-sided, double density 8" disc drives at approx. £300-£350, these prices would depend on the number ordered and the method of shipment.

All the prices quoted would be exclusive of VAT and carriage. All three products are manufactured in Japan by well known and reliable manufacturers.

Incidentally we now have details on the new *Seiko* 16 column matrix print mechanism, which is intended to be used in hand held calculators. It runs from a 5v supply and is a ribbon impact type so it uses ordinary paper. We should be able to supply small quantities of these units (no drive electronics) for £25 (exc. VAT) or 500+ for £12.00.

Fianlly, we have a quantity of 80/132 column matrix print mechanisms (no electronics) available for £130 (exc. VAT). These are *brand new*, surplus to requirements but would require a driver card. Should anyone be interested in designing a suitable card and driver software, we can provide full technical support and would be interested in a royalty agreement.

As we are in direct contact with the Japanese trading company, who represent the manufacturers on a world-wide basis, we can expect the highest possible level of technical assistance.

David Hardman is at *Jayman Electro Devices*, 85 Lees Road, Oldham, Lancs; tel: 061-652 1604.

SMALL ADS

EPROM programming service, 'garbage collector' patch
Martin Spalton, 90 Willowfields, Hilton, Derby DE6 5GU. Tel: Etwall (028 373) 3802.
I am offering an EPROM programming service for both small and large quantities. Currently I deal only in 5v 2716's but many other types will be available soon. Prices including erasure are available on request.

I also offer 'BASIC 3' for Superboard or Compukit with the 'garbage collector' routine corrected as published in this Newsletter. This you can have on 5v 2716 with fitting instructions for £12 (incl. p&p, no VAT). *Special offer to OSI/UK User Group members: £10.* Currently ex-stock.

Expansions for Superboard/C1, Compukit.
Bharat Mistry, 75 St Margaret's Road, Bradford BD7 2BY, W. Yorks.
Light pen £12. Parallel I/O port, two versions: £18, advanced £35. Both provide two eight-bit I/O ports, and have binary I/O for development.

EPROM programmers, two versions: i) requires I/O port, £28; ii) plugs into expansion socket, £38. Both programmers capable of programming single-supply EPROMs.

Digital to analogue convertor board with simple sound int, two precision motor speed control: £25.

Soon to be released (by the time this is printed — we hope!) — sound board using AY 3-8910: ≈£30. EPROM eraser: ≈£25. 8K ROM board, 16K RAM board. 8K RAM + 8K ROM board. Triac controller board and more...

Note: All items are ready built and tested with documentation. For kits please *add* £2 extra per item. Members of User Group please *deduct* £2 per item.

We can also combine board at your request (except EPROM programmer!), this in order to reduce *your cost* and so that you have a compact unit. Please ask for quote.

Please send SAE for information only!

Cheques/POs to *Bharat Mistry* please. All prices inclusive.

GLITCHES

CEGMON/65D conversion
Not so much a glitch as an addendum. The linker routine by *Justin Johnson* on p.16 of this issue has the DATA lines set up for his own 64×32 display. After the DATA 8 in line 1030 the values are machine-specific, and should be copied from locations 64434-64462 inclusive ($FBB2-$FBCE) on your own version of CEGMON. Line 1070 seems to be redundant, in that the values of that line (stored in $0235-$0240) are not actually used by CEGMON.

Jack Pike's 'fast print' routine
Jack has found a slight bug in his routine, published last issue. As printed it will work all right, except on rare occasions when the 'message' as stored spans a page boundary. He suggests that the routine should be amended to end (in hex code) '90 02 E6 C4 60'.

The wrong pet . . .
Finally, a sad tale related to us by one of the dealers. He'd ordered a 32K Pet from the States, and it duly arrived by air-freight along with the rest of his assorted consignment of kit. The box contained the polystyrene foam packaging all right, but the Pet itself was not there — it had been stolen in transit. But the box was not exactly empty: in the place of the Pet were three ten-pound bags of Kitty-Kat litter, in case the non-existent Pet wanted to wee itself on the flight . . .

Single drive copying in 65D
If you have only one drive, like most of us, you will be aware of how tedious it is to copy the operating system onto a new disc. First you have to transfer track zero with the track zero read/write utility, then you have to do a string of CAlls and SAves to the new disc. This isn't too much trouble in some ways, because you can quite happily CAll several sectors into memory at once if you've got the room, and then SAve them back to the new diskette, as long as you remember where they are, and how many pages each sector needs.

However you do your copying, you first need to get a list of the tracks and sectors occupied by the operating system, BASIC and the Assembler/ExMon, and how many pages each requires. You do this with the BASIC utility SECDIR, or from the kernel with DIR TT (where TT is the track number) — the latter only gives you the layout for the one track. A useful tip is to *write this down and keep it* because it won't all stay on the screen at once. Here is the layout of tracks 1 through 8 on an 8in diskette under 65D — would any member care to send us an equivalent for 5¼in discs? Note that the track and sector numbers are in decimal, but the number of pages (256-byte blocks) is in *hex* — take care!

Track	*Sector*	*Pages*
1	01	05
	02	05
2	01	0B
3	01	0B
4	01	0B
5	01	0B
6	01	0A
7	01	09
8	01	01
	02	01
	03	01
	04	01

Assembler and ExMon in late-issue 65D
In recent issues of 65D the 'resident' Assembler/Editor and ExMon have not been resident at all — they aren't even on the disc! For the last year or so — since the introduction of 65D colour systems — OSI have been supplying 65D discs for video systems which have no Assembler or ExMon. If you try to call them, *even if you copy them onto the disc in the right places,* you will get an error. This is because BEXEC* contains a couple of POKEs at the beginning to remove the commands from the kernel! They are marked with REMs, and are easily removed by unlocking the system, taking them out, and PUtting the altered BEXEC* back on the disc. Then you'll have to reboot before you can call the Assembler or ExMon, assuming, that is, that you've got them there from another disc.

It seems a little odd to us that 65D, which is specifically marketed as a 'development' DOS rather than an applications system, should be denied its former development tools — but then OSI definitely has some strange ideas at times! In principle the Assembler and ExMon are available separately, but the dealers don't seem to have any copies over here. If you want the Assembler and ExMon for your 65D, your best bet is to locate and older copy of 65D — pre- about August '80.

Moving between BASIC, Assembler and ExMon in 65D
An interesting problem is that while, in principle, you can call ExMon and the Assembler direct from BASIC (such as ending BEXEC* with DISK!"ASM"), it doesn't quite work. If you do it this way, you'll often get a lockup halfway through your use of the Assembler, and the DOS has difficulty in LOADing disc files. To play safe, exit BASIC first:

EXIT
*A**ASM (or *A**EM)

This may be because calling ASM or EM direct from BEXEC* does not reset zero-page as far as the DOS is concerned, whereas an exit to the DOS first calls a 'swapper' routine that restores the operating-system's use of both page-zero and the stack, and then swaps it out again when the Assembler is called. We should be able to check on this when our copy of the 65D source code — recently published by OSI — finally arrives.

Note that as the Extended Monitor and Assembler both share the 'transient utility' area of memory — between the stack and the bottom of 65D, from $0200–$2200 — you can move freely from one to the other with EX... *A**RE EM or RE AS, rather than re-loading them from disc every time with EM or ASM. You *can't* do this between BASIC and EM/ASM because they don't exist in RAM at the same time — they are loaded into the same space by the DOS.

CEGMON Notes

Using the 'move' subroutine with BASIC

As described elsewhere in this issue, the machine-code 'block move' routine makes certain types of graphics handling — such as setting up colour backgrounds — very much easier. Several people have used it with BASIC, for fast-moving graphics — but as Premier Software's *John Hooker* reminded us, there is one catch in trying to use it directly from BASIC as a USR(X) call. He'd been assembling a video picture off-screen, and then swapping it into screen memory, only to find that the picture was displaced by about 90 bytes from where it was meant to be. The reason is that, as stated in the CEGMON manual, *the 'move' routine expects the machine-code Y-register to be zero* — which it rarely is in BASIC. You cannot set that register directly in BASIC — it has to be done in machine-code — *so the 'move' routine cannot be called direct from BASIC without going through a short machine-code routine first.* Set up the POKEs for a block move as described in the manual; set up a two-instruction machine-code routine of LDY #$00, JMP $FDE4 somewhere suitable, such as at $0240; and do your USR(X) call to that location to reset the register and then do the actual move.

In BASIC (ignoring the set up of the move's start, end and new start) this would come out as:

```
10 FOR X=0 TO 4: READ N: POKE 576+X,N: NEXT
20 DATA 160, 0, 76, 228, 253
30 POKE 11,64: POKE 12,2
```

Linking CEGMON to OS-65D

One impressive solution to this tricky problem from *Justin Johnson:* I have now written a complete link routine that allows 65D V3.1 to use all the CEGMON facilities. I enclose all the information, and please print it in the Newsletter if you so wish. However, I would like to retain the copyright, I would object very strongly to any dealer selling the program but I hope it may be of use to members.

The CEGMON problem is that it needs to use page-2 as well as the 65D BASIC. I have taken the view that CEGMON would cost money to change but the disc BASIC is more suitable for changing. The solution I have employed is to shift the page-2 BASIC code to the top page of memory and change all the calls to the new addresses, I was surprised to find that only 4 appeared to be needed. Page-2 is then filled up with the data needed by CEGMON. A series of POKEs then links the backspace, the screen and the editor. An extra POKE removes the keyboard mask for characters under 32 in the character set. An added bonus is that BASIC will now warm-start after a BREAK, which my system would not do under CEGMON. This allows the use of the CEGMON machine code monitor and a return to BASIC without loss of BASIC program. This is very useful as the Extended Monitor and Assembler are no longer supplied as standard with OS65D V3.1 (a point that members may not realise), it allows patches to be made without the ExMon.

I have given this program a lot of use and it appears to be crash-proof in BASIC, and I think it is OK with ExMon; it will not work with the Assembler but a similar program could be written to cope with that. I would be interested to know if people run into problems or if the routine could be improved.

```
1 REM J.G.J. CEGMON–OS65D linker
2 REM restores warm-start and all CEGMON functions
3 REM partition off top page of memory
5 L=PEEK(133): POKE 133,(L–1)
7 POKE 8960,(L–1)
8 REM transfer page-2 of 65D to top of memory
10 FOR J=512 TO 767
20 A=PEEK(J): POKE J+(L–2)*256,A
25 NEXT J
30 REM change vectors from page-2 to top page
40 READ N: IF N=999 THEN 900
60 POKE N+2,L
70 GOTO 40
100 DATA 2037
110 DATA 2041
120 DATA 4353
130 DATA 8908
500 DATA 999
900 REM put data into page-2 for CEGMON
1000 PRINT: PRINT: PRINT
1010 DATA 0, 32, 10, 0, 0, 0, 0, 8, 43, 11, 35, 15
1020 DATA 87, 11, 165, 9, 165, 8, 0, 87, 100, 87, 87
1030 DATA 8, 70, 251, 155, 255, 148, 251, 112, 254, 123
1040 DATA 254, 47, 128, 208, 64, 215
1050 DATA 189, 128, 208, 157, 64, 215, 202, 96
1060 DATA 0, 32, 192, 210, 136, 249
1070 DATA 6, 97, 6, 52, 27, 199, 27, 83, 27, 247, 34, 4
1080 DATA –1
1110 J=511
1120 READ N: IF N=–1 THEN 1190
1130 J=J+1: POKE J,N
1140 GOTO 1120
1190 REM POKE to link backspace
1200 POKE 1419,255: POKE 2820,255: POKE 11430,255
1205 REM POKE to change screen vector
1210 POKE 8979,53: POKE 8980,248
1212 REM POKE to link editor
1213 POKE 9522,189: POKE 9523,250
1217 POKE 1382,8: REM remove keyboard masking
1220 PRINT CHR$(26)
1225 PRINT
1230 PRINT"          CEGMON LINKED"
1240 PRINT"          J.G.J.   1981"
```

PAL colour for OSI systems

Richard Elen

Many people who have seen or used the colour graphics systems on machines such as the *Apple][* will be convinced of the fact that colour graphics can be a valuable aid, enabling information to be presented more clearly on a screen, as well as making possible the display of graphic patterns and computer art. With the introduction of the *540* board, OSI committed themselves to entering this area, a committment that has been developed subsequently via the Revision B-1 540 video board and the *Superboard Series 2.* This article provides an overview of the possibilities of utilising existing colour facilities on OSI equipment in the UK, and also points to a number of approaches to PAL colour generation which may be useful to owners of OSI equipment which does not have an on-board colour capability.

Colour and the memory map

When OSI decided to redesign the original 540 board, as used on the *C2,* they replaced the original *2102* memory chips with *2114*s, adding in the dealer notes of the time that their intention was to design a new version of the board which had colour video and sound-generation capability. This was finally made available in the shape of the *540B-1* board, and is now fitted to all OSI machines utilising the standard backplane and memory-mapped video — C2 (now C4), C8 and boards for the C2-OEM and C3 series.

In all OSI systems to date, the video memory — used for character data storage — is situated at \$D000 (53248_{10}) upwards. In the case of the standard Superboard, 1K×8 is implemented, while the 540 board (e.g. in the C4) uses 2K, to provide 32 rows of 64 characters displayed. Colour on this board is implemented by 2K×4 bits of separate colour memory, located at \$E000 to \$E7FF (57344_{10} to 59391_{10}). The four bits per location are suitable for the display of 16 colours, as described later. The standard Superboard has no memory in this area, and it is conceivable that extra decoding and memory could be added to offer similar facilities to the 540 board, by examining the 540 colour circuitry in detail. In the meantime, it is possible to implement colour graphics on the Superboard and UK101 by using such systems as that developed by *William Stuart Systems.* This uses a Colour Modulator Board (see later) and a graphics board to enable the upper-bit-set graphics from the character generator to produce pixel-addressable coloured cells under software control. This has the advantage that, as no further memory-map area is used and the normal character memory effectively stores the colour data, coloured displays scroll with the text and may be cleared with the normal screen-clear functions, while the memory set-up of the 540 board requires quite separate control of colour and character displays.

Unfortunately, OSI have made a serious error in their design of the new *Superboard Series 2,* in the area of colour graphics. As far as we can judge, they have implemented colour as separate memory on the new version of the *610* board, called the *630.* While this is in principle the same as on the 540 board, they have slipped up in the fact that, rather than place the colour memory at \$E000, they have placed it at \$D400 — in other words, not only is the memory map incompatible with the C4 series, it means that you cannot expand the character memory to 2K without abandoning their colour system! If anyone from OSI is reading this, *a)* why didn't you put a 64×32 (or at least 48×32) display on the Series 2, and *b)* why didn't you put the colour memory at \$E000?

The 540 board colour circuitry

The \$Exxx page of memory is decoded by U6F (7430) and feeds U6E (74LS139) and U6B (74LS157) to provide the required control signals for the colour memory. Data lines D0 through D3 (inverted) appear at an 8T26 transceiver chip (U1H) and drive, with the video address lines VA0-VA9, a pair of 2114-L3 memory chips (U1F, U1G), configured as

2K×4 bits. The memory contents are scanned by the video clocking system, and a pair of flip-flops, U1E and U1D (74175) supply four signals to the colour encoding circuitry, corresponding to the contents of bits 0, 1, 2 and 3 of the current colour memory location, being Invert, Red, Green and Blue respectively. The Invert signal is applied to an Exclusive-OR gate, U2B (part of a 74LS86) in the black-and-white video path; thus the B&W signal produces either white characters on a black ground, or vice versa, depending on this bit setting.

The Red, Green and Blue signals appear at J1, pins 2, 4 and 3 respectively, the Red signal normally being replaced by a colour frequency adjustment line from the colour pot on the A15 board. Solder pads and a bridge are supplied on the board near the junction of C6 and R12 to enable the Red signal to appear at J1 pin 2 as an option. It should be noted, however, that these signals are unblanked, and may not be suitable for use in colour generation. In addition, they do not contain any character information, and, as only three signals are available, only eight colours (including black and white) may be generated from these feeds.

The RGB lines are gated (by U1C, a 7400) with a blanking signal derived from a 74123 monostable (part of U4A) driven from the Horizontal Sync signal, and are converted to active Low and applied to the control inputs of a 74151 multiplexer (U1A). According to the bit settings applied here, the multiplexer selects one of seven colour subcarrier signals, CD0-CD6, and it appears at the output of U1A. The seven subcarrier signals are derived from a twin-gate crystal oscillator running at 3·579545MHz, the US NTSC standard colour subcarrier frequency. A third gate buffers this signal and provides the reference colour burst, CD0, which is selected by the multiplexer during the blanking period; the other colour delay (CD) lines are derived from six inverters in a selected DN74L04N (U1B) to generate appropriately-delayed colour signals with the requisite phase angles by means of propagation delay in the chip.

The eight colours selected by U1A are fed via the 'Color' pot R10 and 0·001 capacitor C7, to the base of the mixer transistor Q1, where the colour information is added to the sync signal and the black-and-white character information. This is where the other eight colours are generated: the B&W character data is added to the colour information and creates a lighter version of the same hue, but different enough to be regarded as a separate colour. Q1 is configured as an emitter-follower and feeds pin 5 of J1 via a 200Ω video level pot, R9. The base of Q1 is also connected to pin 7 of J1, thus providing a TTL-compatible composite video output. This signal is usually driven into a modulator, often with the use of a low-value resistor to earth to attenuate the signal to about 1v peak-to-peak. Better results can be obtained by connecting the modulator direct to pin 5, the emitter-follower output, as both level and impedance are better suited to the application.

The difficulty in 'subverting' the original circuitry for PAL use lies in the fact that under the PAL European colour TV system, the blue signal must be 180° out of phase with the reference burst; and red and green must be ±45°, the phase relationship of the red and green signals being reversed each line. This is not easy to arrange with the existing circuitry. In addition, the NTSC circuit depends on the propagation delay at 3·58MHz in the 7404, U1B. At the PAL colour subcarrier frequency of 4·433618MHz, different delays would be needed.

The solution to this problem is to add a PAL colour encoder to replace the existing NTSC circuitry. It must be capable of taking the RGB signals at TTL level, and producing a composite PAL-encoded signal; in addition, it must be able to have the B&W character signal added to the colour composite, either in the decoder or via the existing output circuitry of Q1. A number of encoders are available on the market which should be suitable for this purpose, and many types of circuitry are employed in this area, including propagation delays, digital encoding, and analogue matrixing. One method we have used successfully for this purpose is the William Stuart Systems *Colour Modulator Board*, which is also utilised in their implementation of colour for the Superboard and UK101. This board has the added advantages that the B&W signal may be applied directly to the board, and that the board includes a UHF modulator, so it is possible to preserve a separate B&W output to a TV or video monitor, in addition to the colour display, if required.

A practical PAL system for the 540 board

To implement this modification, you will need the following components:

1 A 540B-1 colour video board (if you have a B&W version of this board, implement the necessary components with reference to the manual).

2 A small piece of Veroboard for interfacing the 540 to the William Stuart Systems board, on which will be mounted:
   1×74LS26
   4×10K linear trimmers
   4×1K pullup resistors
   2×10K, 1×22K and 1×47K mixing resistors
   4×small-signal silicon diodes

3 One 7400 IC, to be mounted in the prototyping area of the 540 board.

4 One William Stuart Systems Colour Modulator Board (available from WSS, Dower House, Herongate, Brentwood, Essex CM13 3SD), costing £12 as a kit, or £18 built.

5 Sundry items, such as PC Board mounting pillars, insulated wire, UHF coax cable, and a suitable UHF output socket to mount on your machine.

In addition, you will require a stable +12v supply to drive the modulator board. WSS recommend 7·5v, but we have obtained best results with 12v. This voltage may be obtained from an overwind on your PSU transformer, or by utilising the fact that many PSU transformers use 9-0-9 secondaries. This voltage may be rectified and regulated.

Signals required

The following table indicates the various signals required to be applied to the Vero interface board to the WSS modulator unit. Wires should be run from these points to connect with the interface board:

Signal	*Source on 540 board*
Sync	U4D pin 6
$\overline{\text{Red}}$	U1A pin 11
$\overline{\text{Green}}$	U1A pin 10
$\overline{\text{Blue}}$	U1A pin 9
B&W video	U4D pin 11 connected to pins 1 and 2 of the 7400 in the prototyping area, connect pin 3 to 7400 to interface board.

Checking the 540 board

Check the 540 to make sure that all components are present. Particularly, ensure that the colour memory chips U1F and U1G (2114-L3) are fitted, and the 8T26 at U1H. Examine the board around U2B, and remove any wire link joining pins 5 and 6. Ensure that the tracks to pin 4 (from U1D pin 2), pin 5 (from U3D pin 9) and pin 6 (to U4E pin 1) are undamaged; reconnect them if they have been attacked. This restores the B&W normal/invert mode of operation, and may be very useful, even without the fitting of a colour modulator board. On a B&W TV, it will now be possible to display different shades of grey, and normal or inverse video, according to the number ($0\text{-}15_{10}$) POKEd into the colour memory. Odd numbers give normal video, even numbers give inverse. The exact number will define the shade of grey. If you do not wish to have a 'grey scale' display on B&W, and only require normal/inverse, turn the Color pot R10 down to zero, or remove the 0·001 capacitor C7.

Richard's interface board in front of the William Stuart colour modulator board. From left to right on interface: 12V regulator circuit; video, red, green and blue trimpots; 7426 and allied components.

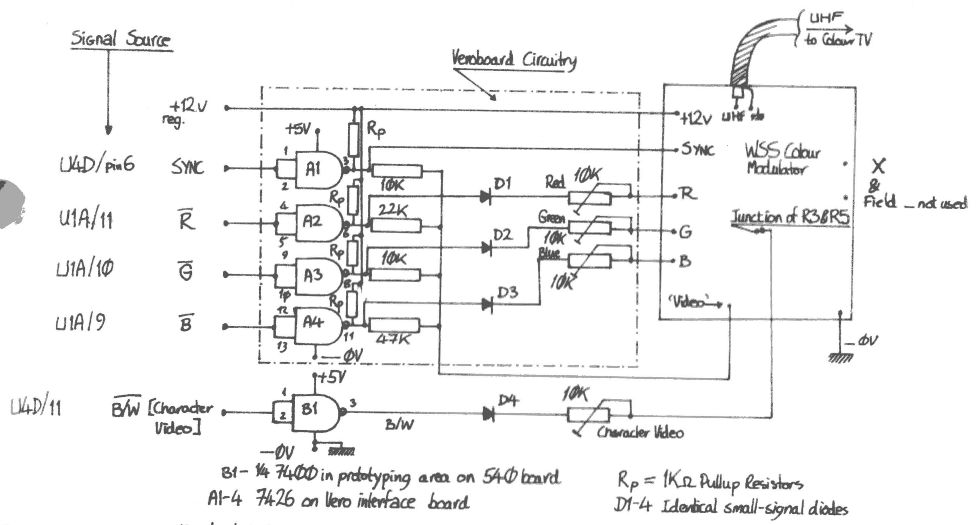


Fig. 1: Interface between OSI 540 Rev B Colour board and William Stuart Colour Modulator

Construction

Fit 7400 in prototyping area. Connect pin 14 to +5v, pin 7 to ground. Connect B&W video line from U4D pin 11 to 7400 pins 1 and 2.

Now construct the interface board on a small piece of Vero, according to the circuit in *Fig. 1*. Run wires from the outputs of the four 10K trimmers to connect to the WSS board. *NOTE* that the four 1K pullup resistors are connected to the modulator supply (+12v) while the 7426 is connected to +5v. The Vero and the WSS board should then be connected together as shown in *Fig. 1*. Connect up the power lines to both boards, fit a suitable output socket to your machine (e.g. Belling-Lee coax socket); connect high-quality UHF coax between WSS UHF output and socket. Mount both boards firmly in a suitable position with PCB mounting clips or spacers. Do not use sticky tape or pads as this will affect UHF stability. If desired, mount WSS board firmly in a metal can (e.g. tobacco tin) with a hole to access the UHF frequency trimmer. Connect the lines derived above the Vero interface board.

Check all wiring between the boards, and check the derivation of signals and wiring to the Vero interface card.

Setting up

Connect the UHF output to a good colour TV via a good-quality UHF coax lead.
Ensure that you can get at the four trimmers on the Veroboard.
Turn R, G, and B pots down to zero (max resistance). Turn Video pot to half-way.
Tune TV to around channel 36; set vanes of UHF trimmer on WSS board to fully closed.
Power up.

Open vanes of UHF trimmer slowly and carefully with non-metallic trimming tool until screen goes dark (vanes should be open about 30°). If no signal is apparent, check voltage at modulator. Adjust TV tuning slightly with AFC *off* until a stable picture of power-on garbage is obtained, adjusting vertical and horizontal holds if necessary. Adjust Video pot temporarily for best readability. If no characters are visible, check wiring of Video signal path.

Press BREAK key and 'M' to enter monitor. Load Extended Monitor if CEGMON is not available.

Open location $DE00, and enter 05. The screen should now show a pattern of random white squares.

Open location $E000 and enter 00. Increment to $E001 and enter 00 again. Do this twice more until you have entered four 00s. Open the next locations and enter 01 four times. Continue incrementing locations and values until you have entered 0F four times. A strip of alternate white and black bars should appear on the top line of the screen. Using the 'Move' command, copy this line into the rest of the colour memory. Under CEGMON, the complete operation is:

```
E000/xx 00,00,00,00,01,01,01,01,02,02,02,02, . . . to . . . 0F,0F,0F,0F
ME000,E800>E040
```

This should result in a number of black and white vertical bars appearing over the screen. Turn Video pot to zero. The screen should go dark.

Advance the Red pot to about three-quarters of full. Adjust the TV tuning carefully to display a wide red bar a third of the way from the left of the screen. If the red is muddy or brownish, advance the pot a little more. If there is ghosting or poor resolution of the bar, retune the TV and locate a different harmonic. There are several of these across the UHF TV band, and some are better than others. Choose one as low in frequency (low channel number) as possible without it being on the same channel as a local broadcast signal (this may cause interference patterns on your display). If no red bar is visible, repeat with the green pot. If no colours appear, check wiring to the Vero interface board, particularly the orientation of the diodes and the power connections. Check that the 7426 goes to +5v and the pullups to +12v.

Having established the presence of the Red signal, turn the pot down and bring up the Green level to about the same (three-quarters). Check that a green bar appears to the right of the red bar's position. If so, repeat the procedure with the Blue pot, which should produce a blue bar at the centre of the screen. If any one of the primary colours does not appear, check the wiring between the 540 board and the WSS unit, including the interface board, for that particular colour channel.

Leave the Blue pot at a level which gives a good, saturated blue bar. If the bar tears slightly, or is slightly right of the underlying dark grey bar on which it should be, reduce the level slightly and/or try another harmonic on the UHF signal. Bring up the Red pot until the bar to the right of the blue area shows a good magenta ('purple'). Check this against the IBA morning test-pattern if in doubt, or adjust to taste. Now bring up the Green pot until the bar to the right of the magenta one shows a good cyan ('Sky Blue'). In addition, a good yellow bar should appear between the green and blue bars. Make fine adjustments to suit your personal taste. Note that the bar on the far right of the screen is the most variable (it is the sum of R, G and B). OSI set this up to give a yellow; this can be simulated by adjusting the relative levels. However this will make the magenta more of a burgundy colour, and the original yellow, olive-green. With a good magenta, cyan and yellow, this right-hand bar will probably be fawn.

The next step requires special care. Advance the Video pot slowly and carefully. As this is done, the right-hand half of each colour bar will increase in brightness, and characters will become visible. The correct setting is a compromise between good character readability and good 'inverted' colours. When set correctly, the the right-hand half of the red bar should be a good orange, and the right-hand part of the magenta, a light mauve, for example. Too much video level will make the inverted colours pale and tending towards white; too little and the characters will be unreadable. Note that characters on a non-inverted colour will appear in the adjacent inverted colour, and vice versa.

Using OSI colour graphics

As has been noted, there are two distinct video memory areas, both of 2K. The first is used for storing character data and is located at $D000 to $D7FF; this is the usual 8-bits wide. The second area stores colour data, and is only four bits wide: this is located from $E000 to $E7FF ($57344_{10}$–$59391_{10}$), i.e. it is 4096_{10} bytes beyond the character memory. Thus, the colour of any cell on the screen may be set by POKEing a location P+4096, where P is the location of the character on-screen. The colour display is enabled by a software switch at $DE00 ($56832_{10}$). This is initialised on power-up, *not* Reset, to 01, which disables colour, sets the display to 64×32 format, and turns the sound (if fitted) off. The complete table is given on page 77 of the October 1979 C4 Manual ('preliminary' as always!) and is as follows:

POKE 56832,N to enable:

N	*Chars/line*	*Sound*	*Display*
0	32	off	B&W
1	64	off	B&W
2	32	on	B&W
3	64	on	B&W
4	32	off	Colour
5	64	off	Colour
6	32	on	Colour
7	64	on	Colour

It should be noted that Reset *does not* clear the colour memory; thus, when colour is implemented, POKEing 56832,5 will produce a weird random colour pattern: be warned — the first thing you need to write is a background-colour setting routine! The four bits of colour memory per location are used to provide four signals, corresponding the Red, Green, Blue (the primary colour TV 'colours') and 'invert'. This last signal is used to invert the B&W video signal to give either white characters on a black background ('normal') or vice versa ('inverted'). This inversion signal is added to the colour signal to produce a change of brightness in the colour, which changes it quite significantly. Thus this four-bit system offers 16 colours, not the simple eight you would expect from mere RGB combinations. These 16 colours may be used independently, without imposing characters on the coloured cell, but when characters are used, they, obviously, take the 'inverted' version of the colour determined by the RGB combination. Thus colour number 2, Red, when used as a background colour, will have characters displayed on it in 'Inverted Red', a rather tasty orange. If the colour is set to 'Inverted Red', number 3, then characters will appear on it in Red. The complete table of colours is shown below: remember that when you PEEK colour locations you will only get a value corresponding to a 4-bit code, and not the normal eight. The upper four (unused) bits at each colour location appear as an $E, because the colour memory is in $Exxx pages, and this value — as 240_{10} when PEEKed — should be subtracted to get the true colour value. Thus, if you are working in machine-code and you set location $E000 to Red ($02), you'll get $E2 when you examine that location subsequently. The four upper bits could conceivably be used to set totally different foreground and background colours by adding some extra memory and a bit of circuitry (we're looking into it).

Here is a corrected PAL version of the chart which appears on page 80 of the C4 manual. Where possible, 'sensible' names have been given to the colours, in addition to OSI's description. This chart takes into account a mysterious colour inversion effect which transposes colours 14 and 15 with 0 and 1, compared with American NTSC system.

		Bits	3	2	1	0	
Decimal Value	*Colour Name*	*OSI Colour Name*	B	G	R	I	(1=on)
0	Black	Black	0	0	0	0	
1	Grey	Inverted Black	0	0	0	1	
2	Red	Red	0	0	1	0	
3	Orange	Inverted Red	0	0	1	1	
4	Green	Green	0	1	0	0	
5	Light Green	Inverted Green	0	1	0	1	
6	Yellow	Olive Green	0	1	1	0	
7	Light Yellow	Inverted Olive Green	0	1	1	1	
8	Blue	Blue	1	0	0	0	
9	Light Blue	Inverted Blue	1	0	0	1	
10	Magenta	Purple	1	0	1	0	
11	Mauve	Inverted Purple	1	0	1	1	
12	Cyan	Sky Blue	1	1	0	0	
13	Sky Blue	Inverted Sky Blue	1	1	0	1	
14	Fawn	Yellow	1	1	1	0	
15	White	Inverted Yellow	1	1	1	1	

OSI have done an interesting little trick with their colour-cell allocation: the rightmost cell on a line that can be addressed 'wraps round' to the left-hand edge of the screen, and the leftmost addressable cell also lends its colour to the (non-addressable) 'cell' on its left. This means that when you fill the screen with a given background colour, the *whole screen* is filled with that colour, and not just a square box with black edges. This can be quite useful. The first two (leftmost) cells on the top line of the screen also contribute their colour to the top and bottom of the screen outside the addressable area to complete the effect, with the proviso that colouring only the leftmost top-line cell will result in colour

bars half a cell wide extending across the screen above and below the addressable area. If you *want* a 64×32 colour box with a black border, avoid using the rightmost colour cells on each line and the first two cells on the top line. It should also be noted that when an inverted colour occupies the rightmost cell on a line, the 'wrap-round' colour will be the *non-inverted* variety, as the inverted colours are derived by turning on the appropriate*character memory* cell as well as the colour memory cell. Thus, if you fill the screen with an *inverted* colour, you will obtain a 64×32 screen of the desired colour, bounded by the corresponding non-inverted colour. Complicated, but versatile.

Useful routines for colour work

Setting a colour background

```
10     REM COL = COLOUR NUMBER
20     FOR P=57344 TO 59391
30     POKE P,COL
40     NEXT P
```

Note that colours can normally only be accessed via POKEs, although you *can* — if you're clever — move CEGMON's scrolling window to point at the colour memory.

Colour co-ordinate plotting

```
  10     REM X=HORIZONTAL COORD : Y=VERTICAL COORD : COL=COLOUR
  20     DIM A(32)
.
.
1000     REM : ARRAY SUBROUTINE
1005     I=0
1010     FOR P=57344 TO 59391 STEP 64
1020     A(I)=P
1030     I=I+1
1040     NEXT P
1050     RETURN
.
.
2000     REM : PLOT SUBROUTINE
2010     P=A(Y)+X
2020     POKE P,COL
2030     RETURN
```

This is fine for either 64×32 or 32×32 displays.

Colour pattern generators

(a) The Marvellous Mantra Machine

This is based on a program in the Apple Manual, with mods for OSI and a couple of variations, including the use of coloured characters. It's very tasty for showing off the colour system to other people!

(b) Colour demo

This short program displays attractive block patterns.

The Marvellous Mantra Machine

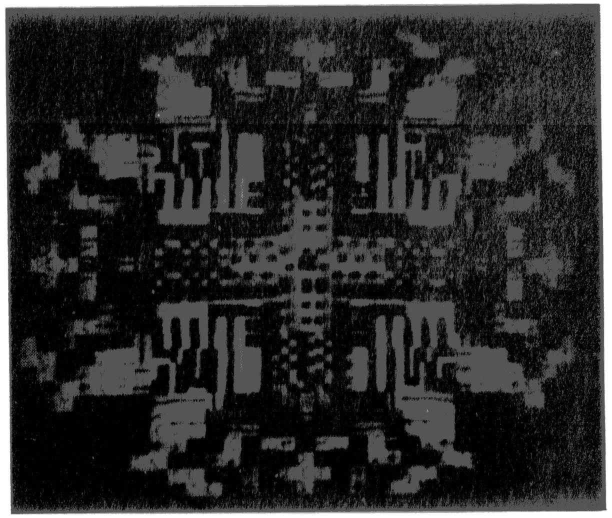
(Note CEGMON screen-clear |CHR$(26)| in lines 15, 50, 67, 108.)

```
 10  DIM A(32)
 12  POKE 56900,0
 15  PRINT CHR$(26): GOSUB 2000
 20 PRINT: PRINT: PRINT"The Marvellous Mantra Machine"
 30 PRINT"OSI version by R. Elen  © 1980"
 40 PRINT: PRINT: PRINT: INPUT"Would you like instructions";D$
 45 IF LEFT$(D$,1)="N" THEN 60
 46 IF LEFT$(D$,1)<>"Y" THEN 42
 50 PRINT CHR$(26): PRINT: PRINT"The Marvellous Mantra Machine"
 51 PRINT"is a colour pattern generator"
 52 PRINT"offering:"
 53 PRINT: PRINT"1: Yantra-like changing patterns"
 54 PRINT"2: Mosaic-like random patterns"
 55 PRINT"3: 'Carpet-like' patterns"
 56 PRINT: PRINT"Options follow in a moment . . ."
 57 FOR I=1 TO 5000: NEXT I
 60 REM
 65 COL=6: GOSUB 1000
 67 PRINT CHR$(26)
 70 PRINT: PRINT"The Marvellous Mantra Machine: Options"
 75 PRINT"————————————————————————————————————"
 76 PRINT: PRINT
 80 FOR I=1 TO 2500: NEXT I
 90 INPUT"Enter 0 for colours only; 1 for colours and characters";Q
100 INPUT"Enter a number between 0 and 16";Z: IF Z<0 OR Z>16 THEN 100
101 PRINT: PRINT"Enter 0 to keep pattern centre open;"
102 INPUT"        1 to fill pattern centre";A
103 PRINT: PRINT"Enter 0 for random colours; 1 for regular colours"
104 INPUT C: IF C=1 OR A=1 THEN 101
105 IF C=1 THEN INPUT"Yantras (0) or 'Carpets' (1)";B
106 IF B>1 OR B<0 THEN 105
107 IF C=0 THEN B=0
108 PRINT CHR$(26): POKE 56900,4
109 FOR W=3 TO 50
110 FOR I=1 TO 16
111 IF C=0 THEN 115
112 IF B=1 THEN COL=INT(RND(1)*16)
115 FOR J=1 TO 15
116 IF A=0 AND I=16 THEN 145
119 IF Z<>0 THEN K=J+2: GOTO 127
120 K=J+I
127 IF B=1 THEN 131
128 IF C=0 THEN COL=INT(RND(1)*16): GOTO 131
130 COL=J*3/(I+3)+I*W/12
131 X=1: Y=K: GOSUB 3000
132 Y=32–K: GOSUB 3000
133 X=32–I: GOSUB 3000
134 Y=K: GOSUB 3000
135 X=K: Y=I: GOSUB 3000
136 Y=32–I: GOSUB 3000
137 X=32–K: GOSUB 3000
138 Y=I: GOSUB 3000
140 NEXT J,I
145 NEXT W: GOTO 109
999 END
```

```
1000 FOR P=57344 TO 59391
1010 POKE P, COL
1020 NEXT P: POKE 56900,5
1030 RETURN
2000 I=0
2005 FOR P=57344 TO 59391 STEP 64
2010 A(I)=P
2020 I=I+1
2030 NEXT P
2040 RETURN
3000 P=A(Y)+X
3010 POKE P,COL
3012 CH=COL*2+21: IF CH>47 THEN CH=CH+76
3015 IF Q=1 THEN POKE P–4096, CH
3020 RETURN
```

Colour demo

```
 1 PRINT CHR$(26)
 2 J=INT(RND(1)*16)
 3 I=0
 5 POKE 56900,5
 6 FOR P=57344 TO 59391: POKE P,J: NEXT P
 7 N=0: M=0: I=I+1
10 FOR P=57344 TO 59391
30 M=M+1: IF M>I THEN M=0: N=N+1
35 IF N>15 THEN N=0
37 POKE P,N
40 NEXT P
50 GOTO 7
```


Sample run of 'Mantra' program
(Polaroid photo)

Aardvark 8K Memory Board

Alan Batch

This cost me $29.95 from Aardvark Technical Services, 1690 Bolton, Walled Lake, MI 48088, plus postage. I used my Barclaycard Visa account so there was no problem about sending the right money.

I got back a double-sided PCB, with plated through holes and roller tinned plus a set of photostat instructions, component layout and circuit diagram. The board is not solder-masked or screen printed with component positions. Finding the correct positions for four resistors and two capacitors requires you to read the circuit. The other component positions are easy to find but the space between 2114s is cramped and the bypass capacitors have to be angled.

The component list calls for 19 bypass capacitors, but the board fits 22 using an additional later Aardvark modification. The positions for the NAND and NOR gates have to be worked out using the circuit diagram.

In spite of its capacitors the board soldered up well and the instructions suggest that you test well for solder bridges, particularly along the bus-rails. (My fault was more difficult; too much solder causing a bridge under a socket between 2 pins and a feed through track.)

My UK101 now draws 2.8 amps so that the standard power supply is OK. You can configure the board for 9th to 16thK or for 17th to 24thK but the 3rd 8K would need an improved power supply.

Additionally a 6821 PIA can be fitted to the board if you want and a program for simple sound generation is given using a 2N2222 and an 8ohm speaker. These I have not tried.

There is a program for memory testing provided which will quickly show up any construction faults but takes a very long time to complete by poking every memory location with a FOR-NEXT from 0 to 255 and checking what is PEEKed, but it is very rigorous.

Provided you have kit construction experience a further 8K of memory for £66 is good value (or £78 if you have no 40-pin lead or 8T28s). If you have some experience and can get your components from one of the advertisers in the hobby press, such as Watford Electronics, then you will be as pleased as me. If you are a beginner and have no experience then this is not the way to expand your memory.