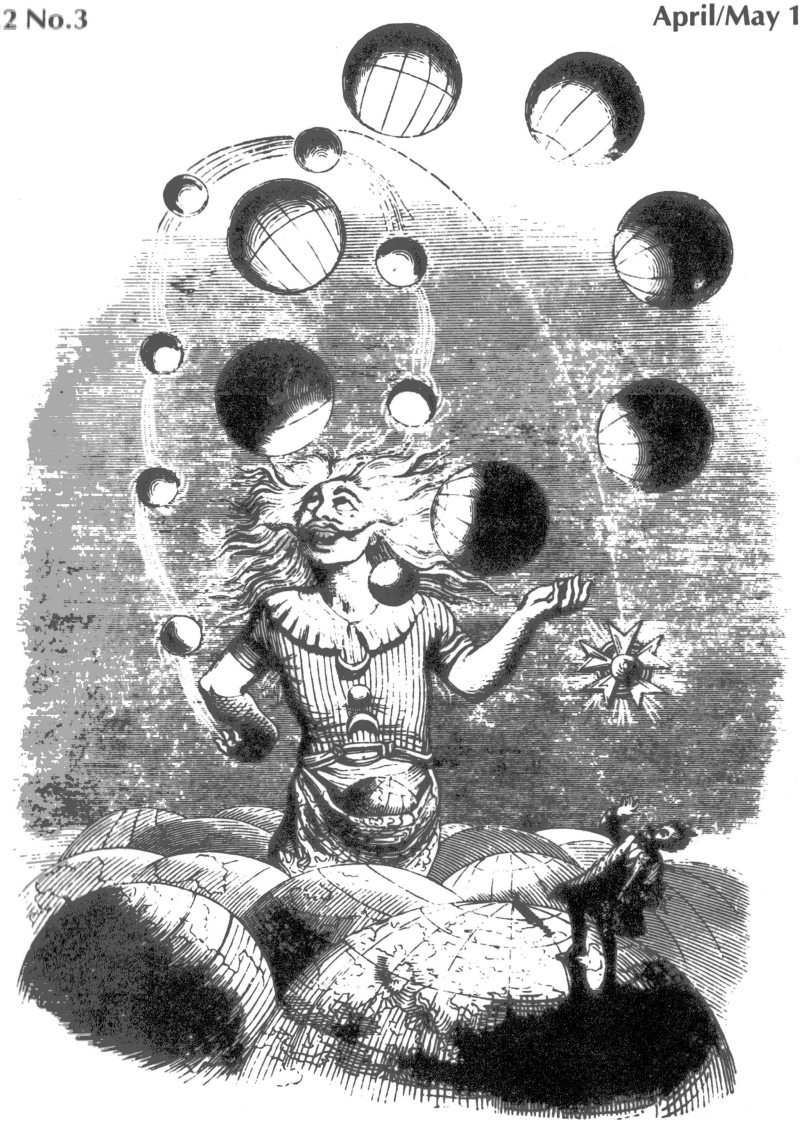
OSI/UK User Group Newsletter

Vol.2 No.3 April/May 1981



Binary juggling

8″ discs on Superboard □ BASIC renumberer □ Series 2 review
and plenty more besides!

Editorial

The influence of Ma-Com?

Despite the utterances of a trade colleague who insists that OSI are 'not just dying, but dead' in the States, the re-formed (or reformed?) company seems remarkably active. Particularly on larger systems, perhaps, but on smaller systems too. For example, already known to exist are: a 5¼" mini-Winchester, about 5Mbyte capacity (release date August); the 541 high-res (512×256) graphics board, complete with light-pen interface, and stackable for colour; a double-processor 'evaluation' board with both Z8000 and 68000 processors — complete with a Microsoft Unix package (Unix multi-user operating system, BASIC, Pascal, Fortran, and C languages) — for release Autumn '81; and communications software for the Ethernet network protocol, available at the end of '81.

Known to be under development is a complete new 'personal' range of 16-bit systems, the C5; a new high-density (1 Mbyte) 5¼" mini-floppy add-on; and a double density IBM-compatible 8" disc controller board. More on the rumour side, but still interesting, is the murmur that OSI's research/development wing is due to move to the more fertile lands of California, at the Okidata plant that OSI bought before being taken over by Ma-Com; and a rather more definite murmur that that remarkable hardware design team are working on tweaking the already impressive performance of the Okidata 74 Mbyte drive to 300 Mbyte by 1982, and aiming for 1 Gbyte (1000 Mbyte) by 1983!

Clearly quite a few of OSI's stated plans are communications-oriented, following the influence of its new owners Ma-Com, who 'intend to be the leaders in the communications industry', to quote one of their recent blurbs. With the increasing interest in networks and 'electronic mail' things certainly do look interesting on the OSI front.

A problem of time

As you will no doubt have noticed, this Newsletter is distinctly late! Partly because of classic difficulties like school holidays; partly because of major equipment changes going on at Wordsmiths; but mainly because the User Group has become too big to be handled as a part-time occupation. Elsewhere in this issue you'll find a section marked HELP! — and that is exactly what we need.

We urgently need people to help us with the *practical* running of the Group. Handling telephone calls, answering queries; editing rough texts into readable form, redrawing scribbled schematics; checking out routines sent in, researching all the various loopholes and ideas that arise in this fascinating field. We've asked people to do this before, and many people have volunteered — the problem is that we still have to sort all the material out first before sending it on, and even that is now taking up more time than we can afford (between us we probably average 10-12 hours a day on User Group matters . . .). So would any members be interested in helping us by literally taking away some of the workload! We have suitable facilities for testing in the ComputrTown set-up in Street, if anyone is down Somerset way; and both George and Richard are accessible — if not always available — in London.

Get in touch with us as soon as you can if you really can help!

Editor/documentation
Tom Graves: 19a West End, Street, Somerset BA16 0LQ; Street (0458) 45359 (pref. early evenings)
Hardware/disc systems
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

Input without delimiters

Roger Derry has finally cracked the input-without-delimiters problem set us by Jack Pike some time ago. In ROM BASIC, the INPUT routine treats commas and colons as delimiters — causing real problems with text-handling applications, which tend to throw away chunks of name-and-address files with the bald statement EXTRA IGNORED. We have had a number of 'fixes', most of them using machine-code patches intercepting the text passing through the input vector. Roger's routine, however, is entirely in BASIC:

You have written several times about ways of being able to input strings without being clobbered because BASIC uses colons and commas as delimiters. So far most of the ways round have used machine code or relied on string addition. I enclose a short demonstration routine that uses no machine code and will accept any keyable character, with 'RETURN' as its delimiter. The only character not accepted is 'LINE-FEED', which can be dealt with in software.

The routine is a development of the Aardvark screen-clear routine published by you about a year ago. I have already written to Richard and George about some other uses for the technique, but here is a specific use. It relies on the fact that if you POKE the string pointers onto the screen and place, in this case, A$ onto the screen, the computer will always look there to find A$. Any string made equal to A$ will be filled with the *current* contents of that part of the screen.

In the enclosed routine (written for a UK101 under CEGMON, but easily adaptable for the other monitors) the cursor is first put at the bottom of the screen so that linefeeds do not affect the line being written on. The routine then calculates the address of the *end* of the line using CEGMON's SWIDTH screen line-length and the text pointers. A$ is then manufactured to be SWIDTH long and then stored onto the screen by a dummy addition. I have made A$ up of CHR$(187)s so as to be visible, but a practical routine would use spaces. After this initialisation the normal keyboard (or editor) call is used to print onto the screen. At this stage no strings are involved — certainly no string addition. Lines 330, 335, 340 and 350 are used to check for RETURN, END-OF-LINE or SPACE with eight characters of end-of-line for auto-CR/LF. Once a new-line requirement is detected, the characters on the screen are put into a string. As the line may be shorter than SWIDTH, LEFT$(A$, PEEK(512)) is used to edit A$ to the same length as the position of the cursor along the line (512 holding the current screen-cursor displacement from the beginning of the line), so that no redundant spaces are included in the string to screw up subsequent editing or manipulation. 'LINE-FEED' could be used as an 'ignore-line' control rather like @ in BASIC.

The following routine is just a demonstration, but it should give some idea of what this type of routine can do.

```
120 FOR D=1 TO 32: REM simple screen-clear
125 REM record end-of-text line (CEGMON pointers)
126 REM 546 is SWIDTH, 555/6 are TEXT pointers
130 A=PEEK(555): B=PEEK(556)
140 Q=B*256+A+PEEK(546)
145 REM force A$ to be on screen
150 S=INT(Q/256)
160 Q=Q-(S*256)
170 REM generate A$ to be as long as screen width
171 REM use CHR$(32) for text line not to be shown
180 FOR X=1 TO PEEK(546): A$=A$+ CHR$(187): NEXT
185 REM put A$ onto screen
190 C=PEEK(129): D=PEEK(130): POKE 129,Q: POKE 130,S
200 A$=A$+"": POKE 129,C: POKE 130,D
```

```
250 REM text entry
300 POKE 11,0: POKE 12,253: REM calls keyboard routine direct
310 DIM A$(10): REM array for 11 lines of text
320 FOR X=1 TO 10
330 IF PEEK(512)>46 THEN KS=13: GOTO 335
332 GOSUB 2000
335 IF KS=13 AND PEEK(512)=0 THEN 330
340 IF KS=13 THEN A$(X)=LEFT$(A$,PEEK(512)): PRINT: GOTO 400
350 IF KS=32 AND PEEK(512)>PEEK(546)-8 THEN PRINT " ";: KS=13: GOTO 340
360 PRINT CHR$(KS);: GOTO 330
400 NEXT X
500 FOR X=0 TO 10: PRINT A$(X): NEXT
600 END
999 :
2000 X=USR(X): KS=PEEK(533): RETURN
2010 REM KS=PEEK(531) on non-CEGMON monitors
```

Stack limits in FOR/NEXT and GOSUB: 'the other OM error' again

If you've found that your otherwise reliable program suddenly develops OM ERRORs, it's probably because you've jumped out of a FOR/NEXT loop or GOSUB without finishing it properly. FOR/NEXT loops must complete their sequence; if you want to jump out, set the X or whatever to the last value, and then call a NEXT. And GOSUBs must meet their RETURNs in the right sequence.

OSI's use of the stack in BASIC is most peculiar. A whole mass of problems arises because, for some reason known only to themselves, OSI have placed their NMI and IRQ interrupt vectors in the middle of the 6502 system stack. The monitor resets the stack pointer to $0128. On the other hand, BASIC's warmstart expects it to be at $01FF, and complains by issuing an OM ERROR on the first active command after a warmstart. (If this worries you, type a dummy line after a warmstart: the error printer — for an SN ERROR, for example — resets the stack, as does the RUN command.)

Other problems arise because OSI's BASIC jealously protects the stack area below $0130 — the NMI jump location. It also happens to use the space below this in order to construct numeric output to the screen. Look at the code from $B95E onwards to see how it does this. BASIC commands use the stack extensively for subroutine calls — printing a single character to the screen uses 19 stack levels, for example. But FOR/NEXT and GOSUB use the stack very heavily to store their return line-numbers and values, so that nested FOR/NEXTs and GOSUBs can quickly run out of room. Each GOSUB uses seven stack levels; each FOR/NEXT loop uses sixteen. Which means that FOR/NEXT calls can only be nested twelve depe — not including stack levels required for printing or anything else — while GOSUBs can in theory be stacked twenty-seven deep.

The other problem will occur if you try to use the NMI and, particularly, the IRQ vectors of the 6502 to drive items like a real-time clock — because BASIC's stack use will almost certainly overwrite their code. The IRQ jump is stored at $01C0 in the stack; it would only take a single nested FOR/NEXT (a timing loop inside an output-array routine, for example) with a PRINT call to overwrite the IRQ jump code. The NMI is somewhat safer, well down in the stack, but a single coding error for GOSUB or FOR/NEXT will crash that.

The amusing thing is that OSI's error-protection in its stack-handling works superbly to protect the NMI jump from being overwritten from beneath — but waits until the jump has actually been overwritten from above before issuing its OM ERROR!

Series 2 — some surprises

Tom Graves

When we first saw the new C1 Series 2, we took an instant dislike to it. It looked wrong, it felt wrong. The firmware had all the old infuriating bugs in it. Nothing seemed to have been done at all. The old package was dressed up in an even worse-looking guise than before.

Now we've had time to look at it more closely, we have to revise our opinions! The firmware may be unchanged, but the hardware is deceptively different — and very much improved.

I bought a Series 2 Superboard for a specific OEM purpose: a code-translator from ASCII to the mangled version of TTS that my new typesetter uses. As a result of my somewhat non-standard interest, I looked very closely at the (as usual, blurred) schematics supplied with the unit. The better-known new features, like the D-to-A converter, the power-on-reset and the two-way screen format were there in their muddled detail. But so were a few complete surprises:

⭑ even on the Superboard, the RS-232 is fully implemented. Adding a three-way switch gives not just cassette, but RS-232 and/or modem at an alternative baud rate.

⭑ the colour option, in principle handled by the still-not-yet-ready 630 board, is in fact largely implemented on the Superboard. All the signals are produced or interpreted by the Superboard, and only the encoding is handled externally.

⭑ all of the BASIC ROM sockets are ready-wired as 8K sockets (for a 2664 ROM, but equally a batch of 2716s on a piggyback board).

⭑ altering a set of links gives the option of using the existing BASIC ROM space for four independent 8K blocks with the same nominal address space, switched by a *software* switch.

⭑ and the usual maze of links for all manner of unexplained purposes!

Of these, the two most interesting (if only because undescribed in any way!) are the colour and 'block select'. The colour circuit design is rather like that on the existing 540 board used on the C4/C8 series, and with a little imagination the encoder described by Richard Elen could be adapted to work on the Superboard. All the colour signals go through a separate connector — not the usual Molex but a 16-pin DIL socket just below the character generator. Outputs are: colour enable from the software switch; colour data from the 2114 next to the main screen memory (boosted by an 8T28 just below it); and an ill-defined signal called CHLD (generated by U19, an LS20 at 4A4 in the schematics). A link cut at 4A2 isolates the video as VID output, and returns as the VID1 input from the encoder. And externally generated composite sync and SVID are also input through the header (although they appear to be marked as outputs!). In general, most of the control information is there — so if anyone does feel like building a PAL encoder, let us know!

More interesting from my own point of view as a machine-code nut is the block-switching option. Very little is involved in this in the way of hardware: it's all there when a handful of links are cut. Normally the chip-enables go direct from A11 and A12 to pin 20 of each BASIC chip, via a somewhat bizarre juggling act with a 3-into-8 LS138 and a set of inverters in U16. The inverters can be isolated by links for each chip (unlabelled, of course). The third line of the 3-into-8 comes from the R/W line, suggesting that the read cycle goes to the BASIC chips, while the write cycle addresses a nonexistent chip somewhere from $8000 to $9FFF.

Pins 18 and 20 on the BASIC chips are normally tied high. But the option is there on the board for *all* the chips to have A11 and A12 going direct to pins 18 and 20 — at the same time. This option is taken up only when the block-select option is used: the block-select lines from the software switch drive the existing 3-into-8 LS138 for the chip selects instead

of A11 and A12. Look at the schematic — 2A4 and 2B4, together with the software switch U74 at 4C1 — for the circuitry behind this. It's quite a tangle, but very clever.

The next question is — what do OSI intend to do with it? We've heard whispers of a 'mini-Pascal-in-ROM' and a few other hints, but nothing definite has appeared. In the meantime, the circuitry now exists on the board to drive 'language cards' as discussed in the last editorial. Anyone building an EPROM card to plug into these 2664-type sockets, please let us know!

The software switch — addressed at $D800, 55926 — controls most of the new features: bit 0 controls the screen format, bit 1 controls colour, and bit 4 the sound/DAC, according to the C1 manual. (When set, the screen is wide, the colour is on and the sound is on, respectively.) The software switch is wired with six data lines in; but since pin 10 of the LS174 goes nowhere, it would suggest that the bits controlling the block switching would be bits 2 and 3 — decimal values 0, 4, 8 and 12. If anyone cares to experiment, again, let us know!

The D/A (digital/analogue) converter is a fairly crude but reasonably effective eight-bit port, using an array of resistors strung across the keyboard lines to control the output level of a 'noise' line, coming out of the J4 two-pin Molex socket to the left of the keyboard. In BASIC, it can only really produce noise — you'll hear a buzz if you plug an amp into the line after you POKE 55296,16 to enable the port. If you POKE a value into the keyboard port, or if you press keys, the background of the buzz will change. Within a BASIC program, the value placed into the keyboard port will be latched, but the 'CTRL-C' check must be disabled (POKE 530,1) for it to stay there, as with all software scanning of the keyboard. For anything resembling a true digital/analogue control, however, the port has to be driven continuously, in machine-code: OSI give a sample program in the C1 manual, producing a square wave with an increasing frequency. Apparently they also produce 'music generation' software for the C1P-MF Series 2, though we have not yet seen it.

All in all, there's a lot more in the Series 2 than we thought. As usual, although OSI's software and firmware may leave much to be desired, their hardware is excellent, well thought out, versatile and reliable. We do wish, though, that they would bother to describe all these features in the manuals!

A NEW LOOK FOR OSI

R. Elen

Things, it might seem, are changing down in Aurora. Maybe it would have happened anyway: maybe it's because OSI is 'under new management'. Whatever the reason, the new C4P Users Manual, if it is an indication of what we should expect from OSI in the future, is more than welcome. If you've been permanently upset by the almost legendary standard of OSI's previous documentation — the poor photocopies, the errors, the illogical layout, the 'preliminary' message that seems to last forever — take a look at this manual. If you have a C2 or a newer C4, this manual will be a godsend. While, in essence, it is a properly printed version of the previous preliminary C4P Users Manual, this new book — no doubt by now available from all good dealers — includes a lot more besides. Over 152 neatly laid-out and printed pages describe both disk- and cassette-based systems, and while Appendix F only describes PEEKs and POKEs to disk BASIC, there is little else to complain about. Along with a full description of machine-specific functions and operational hints, quite clear, full and easily understood discussions of such things as the operation of a parallel port are considered, plus the options available via the 16-pin bus system in the form of the CA-Series boards and their functions. The main headings in the manual speak for themselves!

Introduction: Video display connection; Connecting the floppy or cassette; Starting the machine; Running a canned program; Basic programming; Graphics; Sound; Storing files on cassette or disc; Advanced features; Joysticks and keypads; AC remote control; Parallel I/O; Connection of 16-pin bus devices; Modem and terminal connections; Printer communications; Advanced topics.

This last section includes details on OSI's new Plot BASIC...a BASIC version designed to ease graphics use. This will be a useful addition to the previously somewhat crude POKE method we have all had to live with! The range of functions now available for the C4 definitely puts it in a class unmatched by other machines in the price range. In addition to the above chapter headings, no less than 12 appendices cover further topics:

Troubleshooting and machine organisation; Detailed A15 pin connections (for the connection of I/O devices); Memory-map and mini-floppy organisation; Disc BASIC statements and error listings (this includes both disk and ROM BASIC error codes); POKE and PEEK list; Piano keyboard (variable values for use in music programs); Disk utilities; Hex-to-decimal tutor and conversion tables; ASCII conversion chart; Character graphics and video screen layout; OS-65D user's guide; 65V machine-code monitor; the USR function; Executing machine-language programs; Indirect files (the best description anywhere of this invaluable 65D function); BEXEC; I/O distribution.

There is also a respectable index.

Although the book is still somewhat biased towards disk-based systems, there is still plenty there for the cassette user, and with the price of disk drives coming down all the time (and it looks like there's about to be a big breakthrough here any moment!), the information should become more useful to the C4 owner as time goes by.

It looks very much as if OSI are about to bring their documentation standard up to a level of excellence matched only by their hardware, and for that they deserve a good hearty pat on the back!

BASIC Renumberer

Tony Parsons

Here is a BASIC renumber routine for all BASIC-in-ROM systems — UK101 and OSI — and written in BASIC programs. It has been throughly tested and debugged.

The routine loads at the end of the program to be renumbered with line 63000 acting as a buffer between the two. 63000 was chosen as it is well above any line numbers likely to be encountered in normal programming.

The program allows for selection any first line number and any "difference between line values" value. It adjusts all GOTO's, GOSUB's and IF/THEN's etc. encountered in the program. When its task is complete it finishes by listing the renumbered program.

The renumbering task can be rather slow with this program, sometimes up to 15 minutes depending on the size of the program to be renumbered and the number of GOTO's etc; included as a confidence check therefore, is line 63081 which prints the line of the program that is currently being worked on.

Should the renumbering process produce a GOTO 'line marker' an order of magnitude greater than the original, the whole program has to be moved up in memory to allow insertion of the extra digit in the line. This action will cause an 'R' in the buffer line 63000 to be removed.

R's should be reinserted in line 63000 before the program is run again to allow adequate buffer space.

```
63000 REMRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
63010 A=769: INPUT"1st line no, diff"; B,C
63015 POKE15,42
63020 D=PEEK(A)+256*PEEK(A+1)
63030 J=PEEK(A+2)+256*PEEK(A+3)
63040 IF J=63000 THEN E3=A+4: GOTO 63060
63050 A=D: GOTO 63020
63060 A=769
63070 D=PEEK(A)+256*PEEK(A+1): A4=A
63080 J=PEEK(A+2)+256*PEEK(A+3)
63081 PRINT J
63085 IF J=63000 THEN GOTO 63500
63090 H=PEEK(A+4): I=PEEK(A=5): K=PEEK(A+6)
63100 IF H=0 THEN A=D: GOTO 63070
63110 IF H=160 AND I<58 AND I>47 THEN GOTO 63150
63120 IF H=160 AND K=32 AND I<58 AND I>47 THEN 63150
63130 IF H=136 OR H=140 THEN GOTO 63150
63140 A=A+1: GOTO 63090
63150 H=PEEK(A+4)
63160IF H>57 OR H<48 THEN A=A+1: GOTO 63150
63170 O=0
63175 H=PEEK(A+4)
63180 IF H>57 OR H<48 THEN GOTO 63300
63185 O=(O*10)+(H-48): A=A+1: GOTO 63175
63200 X=X+1: GOTO 63180
63300 O$=STR$(O): A7=A
63310 A1=769: N1=0
63320 D1=PEEK(A1)+256*PEEK(A1+1)
63330 J1=PEEK(A1+2)+256*PEEK(A1+3)
63340 IF J1=O THEN N=(N1*C)+B: GOTO 63360
63350 A1=D1: N1=N1+1: GOTO 63360
63360 N$=STR$(N)
63365 IF LEN(N$)<>LEN(O$) THEN GOSUB 63700
63400 FOR X=1 TO LEN(N$)
63402 JIM=(ASC(MID$(N$,X+1,1)))
63404 POKE((A+4+X)–(LEN(N$))),JIM
63406 NEXT
63411 IF LEN(N$)>=LEN(O$) THEN GOTO 63420
63412 A=A7: GOTO 63081
63500 A=769: E=0
63510 D=PEEK(A)+256*PEEK(A+1)
63520 J=PEEK(A+2)+256*PEEK(A+3)
63522 IF J=63000 THEN GOTO 63600
63530 E=INT(B/256): B=B-(E*256)
63540 POKE A+2,B: POKE A+3,E
63550 A=D: B=B+C+(256*E): GOTO 63510
63600 POKE 15,72: LIST-62999
63610 END
63700 OZ=LEN(N$)-LEN(O$)
63702 IF LEN(N$)>LEN(O$)THEN T=OZ: GOTO 63730
63710 RETURN
63730 FOR Y=1TOT
```



```
63735 A2=A4
63740 FOR X=E3TOA+4STEP-1
63750 W=PEEK(X): POKE X+1,W: NEXT
63760 E3=E3+1: D=D+1: A=A+1
63765 A2=A4
63770 P=PEEK(A2)+256*PEEK(A2+1): P=P+1
63772 P1=INT(P/256): P2=P-P1*256
63780 POKE A2,P2: POKE A2+1,P1
63790 D2=PEEK(A2)+256*PEEK(A2+1)
63800 J2=PEEK(D2+2)+256*PEEK(D2+3)
63802 IF J2=63000 THEN GOTO 63850
63810 A2=D2: NEXT X
63850 NEXT Y
63855 RETURN
```

Disk Notes

48×32 conversion under 65D

A note from *D. Amyes*: Here is a short routine to modify the screen handling routines, which I've added to the BEXEC* supplied on the system disk. It allows printing on the correct baseline, instead of halfway up the screen!

```
8 FOR I=9666 TO 9804
9 IF PEEK (I)<>0 GOTO 13
10 =I+1
11 IF PEEK(1)<>211 GOTO 13
12 POKE I,125
13 NEXT
14 POKE 9801,215
15 POKE 9811,215
```

Running 65D disk programs on BASIC-in-ROM

Another note from *D. Amyes*, sent with CEGMON in mind, but applicable generally to BASIC-in-ROM systems with disks.

Load from disk as normal. Reset (BREAK), and type 'C' for BASIC-in-ROM's cold start. Answer 'MEMORY SIZE?' with the maximum available (eg. 24000) — if you just hit 'RETURN' you will lose the program!

Reset again and type 'M' to enter the monitor. Inspect locations 3279 and 327A; note their contents, and place them on 0079 and 007A respectively.

Reset and type W. The program can then be run under BASIC-in-ROM, as if it had been loaded from tape — but with rather less memory space!

Re-installing Centronics driver in 65D

A note from *M.J. Bues*: In the 5" floppy version for the C1P, OSI have deleted the parallel printer port driver (device 4). As my OKI printer is supplied with a "Centronics port" input, I have had to build a parallel port as per the 8" version for the C2 etc. at $F400. The DOS has the initialization already but I had to add a change to give CB2 as an output for the data strobe. The printer routine I have put on place of the serial CA10 ('device 8') port. The user can either PRINT#8 or change the output distributor to vector PRINT#4 to the No.8 port.

Hardware Modifications: Double-Sided Drives

by Richard Elen

For those of you who are considering upgrading to disk, or even those of you who already have, but need more storage in limited space, double-sided drives offer a relatively inexpensive option. Each surface of an 8in disk, for example, can handle over 250K of storage, yet a double-sided drive doubles your capacity without requiring any more physical space, and at a far lower cost than adding another single-sided drive (typical prices on the U.K.,for a good-quality 8in drive, are £325 for s/s and £385 for d/s ex VAT).

The only disadvantage of a single d/s drive is that you can't create a new disk with operating system as simply as you can with two drives: you still need 'single drive copy' utilities, as before. But you have twice the storage (and only one side needs to have the DOS on it!). Incidentally, your s/s disks can still be used on a d/s drive (but not the other way round).

OSI have a technical note available, number 27, which describes the conversion for a 470 board (the 505 modification is similar) and the A12 ribbon-cable header card for double-sided operation (pages 16 to 18). The 470 board conversion is shown in figs 1 and 2. Figs 3 and 4 show the conversion to the A12 header card. This does, in fact, work. However, the jumpering of the select lines is not strictly correct, and this could lead to problems with more than one drive.

Device selection under OSI's disk operating systems makes use of both the SELECT and SIDE SELECT lines to access devices A to D. There are four SELECT lines, SELECT/0 to /3, corresponding to drive edge-connector (ribbon cable) lines 26, 28, 30, and 32 (odd-numbered lines go to ground), and the SIDE SELECT line appearsd at pin 14. A drive is usually jumpered (at the radial select jumper socket) to pick up SELECT/0. However, the jumpering suggested by OSI ties this line (pin 26) to ground, while SELECT/3 (pin 32) changes state merrily, unacknowledgewd by the drive.

The result is as follows:

Device selected by DOS	A	B	C	D
Side of disk selected	0	0	1	1

While this is fine for one drive, what if you want to add another? (Note, incidentally, that OSI intend you to have Devices A and C as sides 0 and 1 of the first drive, where Devices B and D are the corresponding sides of the second drive.I suppose this gives you continuity with single-sided systems, in which Devices A and B would be the 0 (only) sides of the two drives.)

The solution is much simpler than the description of what is wrong. You simply swap the A12 board jumpers to ribbon cable pins 26 and 32. If you're using two drives, you won't need SELECT/3, so the fact that it will now be low all the time matters not a whit. What does matter is that Pin 26 will now go up and down as shown below (SELECT/1 and /2 are held high, and are also unimportant):

Select:	Pin	Device A	Device B	Device C	Device D
0	26	0	1	0	1
Side	14	0	0	1	1
Drive	Side	0	-	1	-

Note that the selection of Device B or D in this configuration, with a single drive, will produce an error (of the form 'THIS DRIVE DOESN'T EXIST YET ERROR'). When you institute a second double-sided drive, all you need to do is invert the SELECT/0 line, and jumper your second drive to suit. For example, you could install a 7404 on the 470 board, and re-jumper, say SELECT/1 to pick up an inverted version of the SELECT/0 line: then you would simply jumper your second drive to pick up SELECT/1, typically by jumpering pins 3 and 20 on the Radial Select option pins.

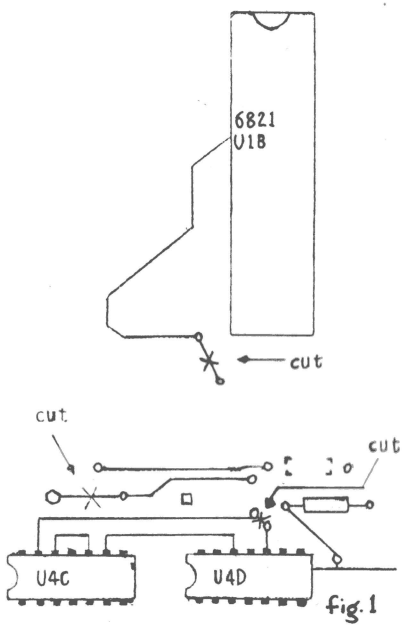


fig.1

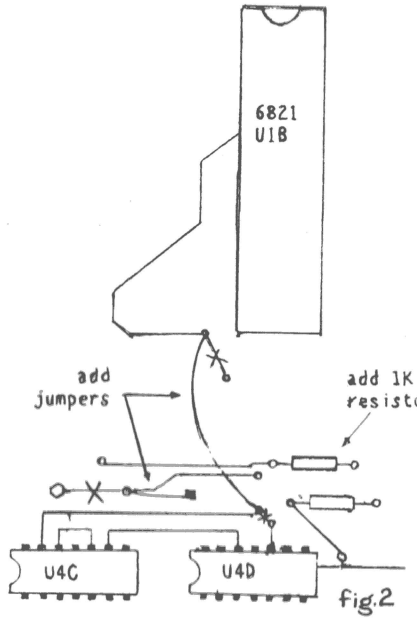


fig.2

Another, far simpler, solution is, of course, to fit a jumper on your first drive to pick up the SELECT/3 line (Radial Select pins 7 and 16). However, as you have to modify the A12 header board anyway, it might as well be 'correct'. Fig 5 shows the jumpering needed on the underside of the A12 to use SELECT/0 instead of SELECT/3 (Fig.3 also applies). However you decide to solve the problem, it is as well to be aware that the problem exists, so that you avoid confusion.

Floppy Connector Board Modified For Dual Sided Drives

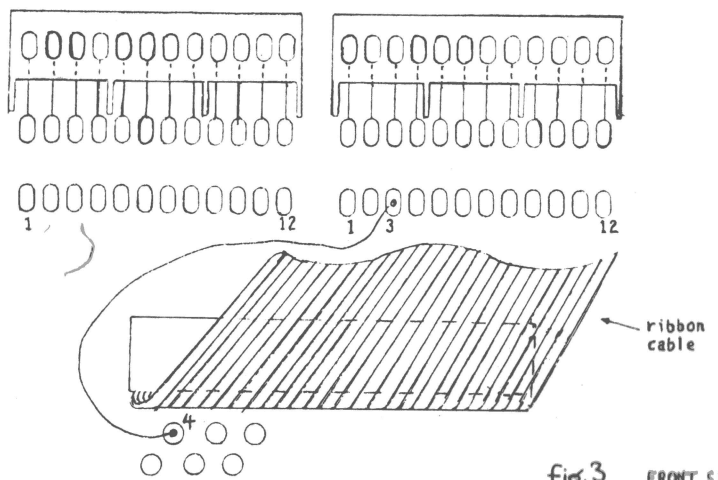


fig.3 FRONT SIDE of board

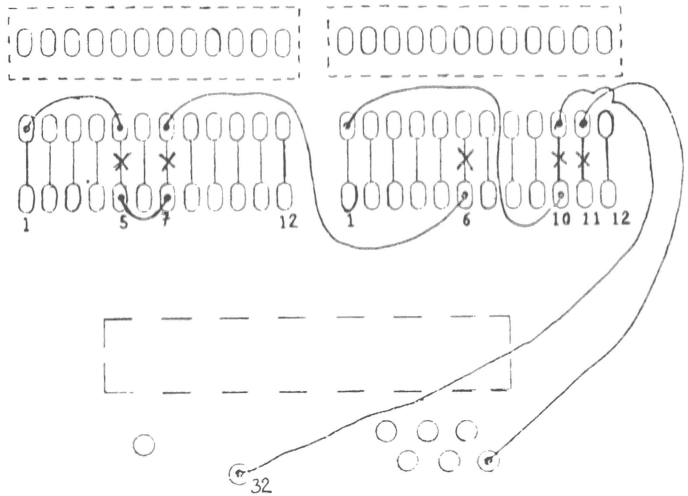


fig.4 BACK SIDE of board as suggested by OSI

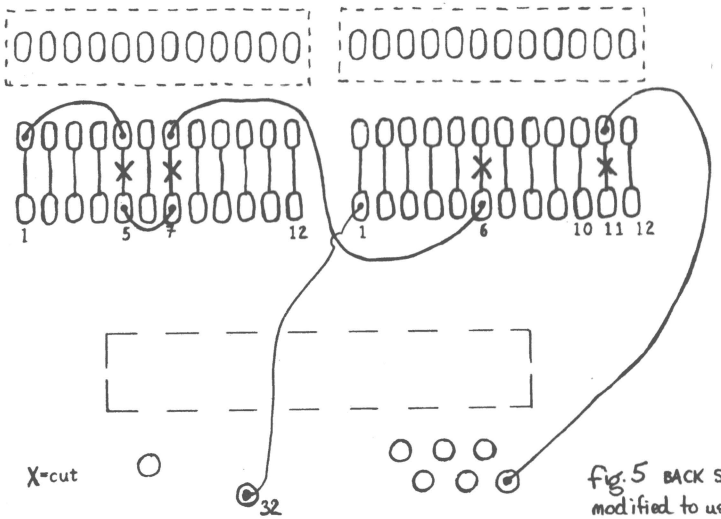


fig.5 BACK SIDE of board modified to use SELECT/0

In passing, it is interesting to consider exactly what OSI intended in the way of drives. The 470 board appears to have been designed for single-sided drives, while the ribbon cable seems to suit double-sided equipment (you need no modification to the drive edge-connector for d/s operation: a single-sided drive, however, needs extensive re-connection at this point). The A12 header card on the other hand, needs one modification even for s/s drives (but more for d/s systems). Odd. [As always — *Ed.*]

Dealer Notes

All busy on the dealer front

Plenty of things are brewing on the OSI/UK101 dealer front. For starters, some kind of price war seems to be going on at the Superboard/UK101 end of the market: Comp have cut their kit price to £149, and are throwing in their new monitor with the kit as part of the deal; Watford Electronics cut their Superboard price to £149 (the 'official' price is supposed to be £159), and have released a 4K monitor at £20; Mutek followed the Superboard price cut, but include the new Series II CEGMON in their price of £149; and CTS have issued a monitor of their own at £27.50. (Talking of monitors, we're obviously biased, since we wrote CEGMON — but would a user of Watford's WEMON care to send us a review?)

Premier have their board system on the market now, with a whole string of price cuts phoned through at the last minute. OS/UK have signed up a number of new dealers — supposedly thirty now, RisComp in Princes Risborough and Kram in Leicester among them — let us know who they are! And what was recently Beaver Systems of Thame has now become Avalon Computers, down the road from us in Street — the new name should go with a slightly less chaotic service, we hope!

An extension to word-processing

Your editor's firm Wordsmiths has finally completed its computer-to-typesetter interface system — which they call their Anvil, a new way to forge words into shape! Based on OSI computers (of course) and using a modified Superboard as a dedicated code-converter from standard ASCII to the bizarre TTS code used by typesetter systems, it gives OSI users access to a very powerful 'printer' — 200-odd characters per second, 135 sizes from 1/20" to 1", and over twenty type styles giving over 3000 characters overall. The system is designed to be used for complex non-conventional print work, combining data-processing with type-setting for specialist applications; but the OSI interfaces mean that it can accept any OSI text or listings, on standard cassette format, 5¼" or 8" disk, OS-65D or OS-65U, or the WP-2, WP-3 and WP-6502 word-processors. Modem links and interfaces for other computer systems will be available shortly; more details from Tom Graves on Street (0458) 45359.

This also means that for the Newsletter we can typeset text, listings or source code of programs direct from tape or disc — saving a lot of work, and reducing the number of bugs reintroduced to your work by retyping! If you want to send a contribution to the Newsletter, you'll help us a lot by sending it in these standard OSI formats.

How not to design a system

No, this time it *isn't* OSI. Back in October, Wordsmiths took delivery of a new typesetter system: the CRTronic, by Linotype (Germany). Very nice, superbly flexible, large character set, small, compact and quiet. For the computer buff, twin Z80s for foreground and background, plus an 8x300 dedicated disc-controller; twin 5¼" double-density floppies storing 160K each; and a total of 160K of RAM. For the software buff, an intellectual tour-de-force: the typesetting 'printer' works by splitting character shapes into lines, and drawing t[illegible]onto photographic paper with a projection CRT.

But the designers somehow forgot that anyone had to *use* the system. It works, very well — as long as you never make *any* mistake. It has virtually no error protection — make any kind of mistake, and it justs stops. The keyboard is brilliantly designed to slow the operator down, neatly angled the wrong way to be uncomfortable, and extra-easy to hit either the wrong key or several keys at once. The shiftlock latches on without warning. All the control functions depend on the state of the shift-keys: commands are only accessible in lower case, for example, while command letters have then to be typed in in caps. For good measure, most of the lethal commands (delete text, get directory and the like) have no safety checks at all, and many are set up as the shift-values of much-used keys — kill-text, the

shift value of the 'tab' key, being a particular case in point. Its hyphenation routines, as you can see in this Newsletter, are diabolical; but because of a bug in the system (still not even acknowledged, let alone corrected, after eight months) the hyphenation can't be switched off without worse errors occurring. And so on. Stupid design faults making a brilliant system almost unusable; a classic, and very expensive, example of how not to design a system.

Wordsmiths are replacing it with a much faster but much more expensive British system — a Linotron 202 (now the core of their Anvil system — see above) — so this should be the last issue of the Newsletter produced on it. But it's worth bearing in mind that design blunders on this scale are none too rare in the big-computer industry, and in this case are in a desktop system priced at about *fifteen thousand* pounds. That's a hundred times the price of a Superboard; twice the price of even the most expensive OSI machine, and with nothing like the facilities. OSI may have some failings and limitations, perhaps; but by comparison with the monumental blunders we see elsewhere, they do get a remarkable amount right!

A few 'User Notes' from dealers

Mutek rang up to say that a few UK101 users had had problems with the BASIC 3 chip — Comp had played about with the BASIC/monitor link to handle RUBOUT just enough to stop the replacement BASIC 3 working on early UK101s only. The problem does not arise under CEGMON, as the delete is handled within the screen-handler, without needing a monitor patch. Also a note from one of our members, *P. Godwin*, who found that CEGMON gives a rubout problem (again — OSI's blasted rubout routine!) on his UK101; it turns out that his machine was one of the first hundred from Comp, which uses 28_{10} as rubout, rather than 95_{10} like all OSI and later Comp systems. If you have one of these systems, and want CEGMON, say Mutek, let them know so that a 'special' version of the PROM can be blown.

A note from Ian Richardson of *Northern Micro* on his 8K+16K expansion unit: The first 20 units were sent out with inadequate paperwork. He says he has rectified the mistakes and redone the documentation, and most of those twenty have been contacted. He still needs, however, to get in touch with Messrs. *Adamson, Ball, Davies, Scales and Dr Peppiat* — if any are members, please get in touch with him on 0484 89 2062.

The indefatigable David Hardman of *Jayman Electro Devices* (see their ad below) has come up with a particularly useful-looking board: a Centronics parallel interface for any OSI-based system, that pretends to be the system's ACIA — so no software changes are required. He says that the ACIA and parallel ports are swapped in and out with a POKE, so the ACIA is always available. It's still in the pre-production phase, with a few extra features being added, but should cost around £25 when released. He's also arranged a very good price on a complete 8300 Comet 40/80/132 column printer (rrp £440, User Group £280), and on the same printer mechanism with case and Centronics driver board (£180).

SMALL ADS

Have[illegible]rdware Prograph Board for Superboard and UK101. Plugs in to give 128 user-define[illegible]haracters + ASCII standard; switchable to give original set with single command. Kit £39.50 +VAT, ready-built and tested £47.00 +VAT.
Includes demo tape and instructions.

Software cassettes for C1E and UK101:
Patience, Mancala, Othello: — 8K £2.00 each.
Invaders, Fruit Machine, Solitaire: — £2.00 per pair.

Hardware:— Resistors 1p; DIL sockets (pins/pence): 8/8, 14/9, 16/10, 18/13, 24/19, 40/25.

Hardware Add-ons
from Northern Micro

48×32 video enhancement

A must for all users — gives a better display than the Pet, Apple, Sharp or Trash-80!
For both Superboard II and Compukit (state which).
Gives a true 48×32 display at 50Hz.
Doubles video memory and makes the characters the correct shape.
Available with either a software patch to relocate the screen's vectors or with a special CEGMON which does this in firmware.
Comes with all components and full and easy to follow instructions.
Requires soldering ability, or you can use our fitting service (72 hour turn-round).

Display Kit + instructions **£15**, *48×32 CEGMON* **£29.50**, *or* ***both only*** **£42**, *fitting* **£12.**

Low Cost Memory Expansion Board for UK101 and all Superboards

8K RAM and 4K, 8K or 16K of EPROM.
Plugs into 40-pin Expansion Socket.
Fully static — uses fast 2114 L 300ns RAM which allows 2MHz operation.
Very high quality PCB — double sided with plated thro' holes, silk screen layout and solder resist.
Fully buffered address lines and all memory is bank selectable in 8K blocks.
All decoding done 'on board'.

Kit comprises: — 40 pin DIL plug/connector, PCB, 4K of 2114L 300ns RAM, 7 decoding and buffering ICs, various IC sockets and capacitors, full instructions.
Price: Kit **£42.95**, *fully built, tested and guaranteed* **£49.95**, *bare PCB and instr.* **£17.50**.
(*N.B.* A 'Toolkit' EPROM will be released shortly to fit on this board, costing less than £20).

We offer a full back-up service on all our products and kits and will repair or maintain any Superboard or UK101 at very reasonable rates.

Programs For All 48×32 Boards and Standard Superboards

Chess 1.9 — Play against your computer. 2 levels and speeds of play (55 secs max). Choice of computer playing several different openings. High speed load. **£8.00**
Draughts — Super BASIC program, difficult to beat, nice display. **£3.50**
Pinball — Great simulation of a Pinball machine (48×32 only). **£3.00**
Dodgems — Another great arcade type game, you must try to beat the computer's car driven by a crazy Italian (also only 48×32). **£3.00**
Golf — For one to four players. 18 hole course, requires skill and judgment but without that nasty walking. **£4.00**
Air Attack — Bomb New York flat and land your plane — very difficult. **£2.50**
Space Invaders — Very fast machine code program — eight skill levels — as seen in pubs, clubs, schools, canteens, chip shops etc. (not 48×32). **£4.50**
Assembler Editor — A must for the serious machine-code programmer. **£24.00**

Compare our prices — we don't think they can be beaten — others are twice the price.

NORTHERN MICRO
29 Moorcroft Park, New Mill, Huddersfield. *Tel:* Holmfirth (048 489) 2062

SAE for full catalogue:
Haven Hardware, 4 Asby Road, Asby, Workington, Cumbria CA14 4RR.

Supercase for Superboards by Jayman Electro Devices. A two-part painted heavy-gauge aluminium case, similar to the C1 Series I case but with hinged top cover. 16¼" wide × 5¼" high by 17" deep. Suitable for Superboard, UK101 and others; separate keyboard cutouts for Superboard, UK101, or solid blank for 'homebrew' systems. The computer board is mounted on a rigid inclined support plate punched for two 3A regulators, and leaving room for 610 board and other add-ons below. Backplate punched with finger-proof ventilation slots and holes for coax socket, 3 DIN sockets, fuse holder, mains switch, and cable clamp; also removable 8"×3½" plate for drilling your own holes! Underside punched at front with six 2¼"×⅜" holes for ventilation and cables.
Case only £28.00; 'mains kit' with switch, fuse-holder etc £2.50 +26p per metre cable; p&p on case or case/mains kit £1.50, kit only 60p (all prices excluding VAT).
Jayman Electro Devices, 85 Lees Road, Oldham, Lancs; tel. 061-652 1604.

STOP!

Now read this:

Is writing software in EPROMS expensive? Well it depends upon how you go around doing it!

Forget about the programming services that cost £2 to £3 per K (kilobyte). ***Forget*** also the £100-plus Eprom programmers! Because, here we have an EPROM programmer which plugs into UK101, Superboard etc. J1 expansion socket, resides at $F1Fx and costs ..... wait for it **ONLY £38!** (no joke!!!).

If you have a parallel I/O port you can program your EPROMs for ***just*** £28 (uses 12 I/O lines, Gnd & 5+5).

Alternatively give your Micro an I/O port (for that relay, Lights, Printer, motors etc.), and an EPROM programmer for **ONLY £43** (the EPROM programmer board plugs into the I/O board when required).
What can the programmer do? Well how's this:

(1) Test BLANK (erased), EPROMS.
(2) Read EPROMs into memory.
(3) Program EPROMs.
(4) Copy EPROMs (as many as you want!).
(5) Verify EPROMs.

FULL software provided with the hardware (not just examples!), well documented and completely **ASSEMBLED** on PCB (not veroboard!).

Now you can fix that BASIC 3, BASIC 1, put EXMON in EPROM, write another system monitor, add new commands / language etc.

ALSO available: Four slot J1 bus board £20; Light pen £14; Digital to Analogue converter £23; Analogue to Digital converter £25; Programmable sound generator £32; Dual Monitor board £8; Dual Character set board £9.50; 8K RAM plus 8/16K EPROM board (just plug in Memory chips) £60.30.

Information Sheet.
How to add at least 8K RAM plus 8K EPROM **on board** without sweat! £3.

ALL THIS AVAILABLE NOW (not next week, month...). All orders First-In First-Out only.
Add £1.50 to orders less than £50.
Send enquiries/cheques/P.Os to:

B. Mistry, 75 St.Margret's Road, Bradford, West Yorkshire.
Members of the OSI USER GROUP deduct £2 per item (greater than £15)

STOP PRESS

Disc patch for CEGMON

The main limitation of Justin Johnson's CEGMON-65D linker is that it only works with BASIC; separate linkers are needed for ExMon and (particularly) the Assembler. Steve Hanlan of Beaver (now Avalon Computers) has tackled a different approach, working within 65D itself. We've only space for a brief comment here; fuller details next issue.

The idea is that, just as the keyboard routine swaps the locations $0213-$0216 in and out to avoid corrupting BASIC or the Assembler, the video routine can swap the entire page 2 stores in and out. Since the CEGMON video routines are used instead of 65D's, the 65D video routine space is used as the swapper store. This does make video handling significantly slower — about 700 cycles longer per character output to the screen; but it is within the operating system rather than BASIC, so it is independant of the software running at the time.

Known difficulties are with OSI's backspace (again!) which is easy to patch, and with indirect files. More details next issue.

Disk handling in 48-character mode on Series 2 Superboard

We've hit against a strange problem in 65D on the new Series 2 Superboard. It loads from disk happily enough in 48-character mode; but it hangs immediately any attempt is made to PUT or SAVE to disk. It appears to scramble the track-header. We can't work this one out — any ideas, anyone?

The BASICs of machine-code
Part 4: Binary juggling

Tom Graves

Last issue we dealt with the use of the disassembler, and judging from the comments people have sent in — and their suggested modifications to the rather limited program of last time — this has at least been useful. But we have had many comments from others that they are still having difficulty with BASIC, let alone machine-code! So the discussion on 'hooks into BASIC' — adding your own commands to the interpreter — will be postponed till next issue; for this part of the series we will combine the general theme of machine-code with the problems of working direct to screen memory on BASIC, by using a number-base conversion routine as a demonstrator for both.

The program that follows was developed (or rather knocked up over a bleary-eyed evening) by *Steve Hanlan* of Beaver Systems, from a very rough program of my own (which George Chkiantz insists on referring to as 'Binary Beans', for some reason known only by himself!). The original program showed a travelling cursor under a row of boxes representing the individual bits of a byte. At the cursor, to '1' or, if already '1', to '0'. The resultant value was shown below in decimal and hexadecimal. Here Steve has adapted the routine so that the values can be changed in any of the number bases, showing the effect on the other number bases.

The other difference is that the sceen itself is used as the memory — the values are not stored in BASIC variables, but actually as their ASCII values on the screen — a principle which is particularly suitable for many real-time and graphic games in BASIC. This does make the routine rather slower than if BASIC variables had been used, but this is acceptable in a demonstration program!

```
10 FOR Q=1 TO 30: PRINT: NEXT
20 POKE 56900,0
30 DIM B(7)
40 K=57088: L=32: IF PEEK(K)<128 THEN L=64: S=8
```

This section does a general set-up: 10 clears the screen, 20 defines the C2 and C4 screen-format as 32×32 (ignored on C1/Superboards); 30 sets up the array for the binary value store used later; and 40 adjusts the program for different *standard* machines — L is the nominal width of the screen (32 on standard C1 series, 64 on C2/C4 series), and S is the distance on from the left margin. The routine checks this by looking at the keyboard port, which returns a value >127 on a standard Superboard or C1; on Series 2 Superboards (the latest version) and on modified systems like the C1E L may need to be set to 64 — check, because this test is used later in the program.

```
50 PT=53446+L*6: PRINT SPC(S) "Binary" SPC(4) "Dec" SPC(2) "Hex"
55 For Q=1 TO 11: PRINT: NEXT
60 PRINT "To move the cursor use: PRINT: PRINT SPC(4) "the SHIFT keys"
70 PRINT: PRINT SPC(1) "To increment a digit"
8[illegible]RINT: PRINT SPC(3) "use the space bar"
9[illegible]INT: PRINT SPC(4) "To exit use 'p'"
```

These obviously print up the instructions. I've used SPC(n) statements here because the typesetter can't easily show the exact number of spaces used, and here they do happen to be important; they can all be replaced by simple spaces except for the SPC(S) in 50, which places the labels in the correct positions for the different screen formats.

```
100 FOR Q=PT+1 TO PT+8: POKE Q,217: POKE Q+2*L,215: POKE Q+L,48: NEXT
```

```
110 FOR Q=PT+12 TO PT+14: POKE Q,217: POKE Q+2*L,215: POKE Q+L,48: NEXT
120 FOR Q=PT+18 TO PT+19: POKE Q,217: POKE Q+2*L,215: POKE Q+L,48: NEXT
130 CU=PT+3*L+8: POKE CU,94
```

These set up the number boxes and their contents — first the binary, then decimal, then hexadecimal. Note that the values are POKEd direct into memory, in defined positions: this allows us to use them to store the actual totals in each case, as memory. The three lines 100 to 120 place first the top-line of each section, then the bottom line of the boxes two screen lines down, then the '0' within each box; the actual placement could have been done more neatly with a GOSUB, but then this was written at one o'clock in the morning! Line 130 defines the cursor position, one screen line below the base of the boxes, and prints it direct to screen as an ' ↑ ' symbol.

```
140 RO=254: R1=253: IF PEEK (K)<127 THEN RO=1: R1=2
150 POKE 530,1
160 POKE K,RO: C=PEEK(K): IF C<128 THEN C=255-C
```

These get the key values — R0 and R1 refer to row-0 and row-1 of keyboard respectively.

The shuffling of values in 140 and 160 allows a direct comparison regardless of which machine — and therefore keyboard type — is being used. The POKE 530,1 in 150 disables the 'control-C' escape — you can only exit the program by pressing 'P' (dealt with in lines 190-200) or, of course, BREAK.

```
170 IF C=250 THEN GOSUB 260
180 IF C=252 THEN GOSUB 290
```

These pick out left-shift and right-shift respectively.

```
190 POKE K,R1: C=PEEK(K): IF C<128 THEN C=255-C
200 IF C=253 THEN FOR Q=1 TO 30: PRINT: NEXT: END
```

This ends the program on finding the 'P' key pressed.

```
210 IF C<>239 THEN FOR P=1 TO 150: NEXT: GOTO 160
220 ST=PT+3*L
230 IF (CU>ST) AND (CU-9<ST THEN 320
240 IF (CU>ST+11) AND (CU<ST+15) THEN 490
250 IF (CU>ST+17) AND (CU<ST+20) THEN 550
```

Line 210 checks for the 'space' bar; if not, then the scan loop starts again after a short delay, else the space bar must have been pressed to call for a value change. Lines 220 to 250 check the position of the travelling cursor: ST is the beginning of the cursor line. If the 230 check succeeds, the cursor is below the binary set, and the decimal and hex must also be changed (in 320 on); if 240 succeeds, the cursor is under decimal, and calls for changes to binary and hex; if 250, binary and decimal must be changed.

```
260 POKE CU,32: CU=CU-1: IF CU=PT+3*L THEN CU=CU+19
270 IF PEEK (CU-*L(=32 THEN 260
280 POKE CU,94: RETURN
290 POKE CU,32: CU=CU+1: IF CU=PT+3*L+20 THEN CU=CU-20
300 IF PEEK (CU-3*L)=32 THEN 290
310 POKE CU,94: RETURN
```

These are the cursor-shift subroutines called by left-shift (170) and right-shift (180): the cursor is blanked, and its position is decremented or incremented respectively; if it overshoots either end, the cursor 'wraps round' to the other end (CU=CU+19, CU=CU-20 — the difference is because CU runs from 0 to 19!). If the point above the cursor is a space

— in other words is between the stores — the routine loop round until a digit *is* found, so that the cursor effectively jumps between the stores.

```
320 CH=PEEK(CU-2*L): CH=CH+1: IF CH=50 THEN CH=48
330 POKE CU–2*L,CH: N=0
340 FOR Q=0 TO 7: PEEK(PT+L+8-Q): IF X=49 THEN N=N+2 ↑ Q
350 NEXT
```

This part scans the binary section of the display: it changes the value of the digit above the cursor (CU-2*L), converting it back to '0' if it is already '1' (320). 340 and 350 scan the binary section, converting it to a total value stored in N.

```
360 N(2)=INT(N/100): N=N-N(2)*100: N(1)=INT(N/10): N(0)=N-N(1)/10
370 FOR Q=0 TO 2: POKE PT+L+14-Q,N(Q)+48: NEXT
380 IF W=1 THEN RETURN
```

These print out the decimal part of the display, converting the total in N into three decimal digits, and POKEing their ASCII values direct to the screen by adding 48 (check this with the graphics character table in your manual if you're not sure how this works). Line 380 allows this to be used as a 'sometimes subroutine', called by line 610 later.

```
390 N=0: FOR Q=0 TO 3: X=PEEK(PT+8+L–Q): IF X=49 THEN N=N+2 ↑ Q
400 NEXT
410 IF N<10 THEN CH=N+48
420 IF N>=10 THEN CH=N+55
430 POKE PT+L+19,CH
440 N=0: FOR Q=4 TO 7: X=PEEK(PT+8+L–Q): IF X=49 THEN N=N+2 ↑ (Q–4
450 NEXT: IF N<10 THEN CH=N+48
460 IF N>=10 THEN CH=N+55
470 POKE PT+L+18,CH
480 GOTO 160
```

These convert the binary value to two hexadecimal digits, by scanning the respective lower (bits 0 to 3, counting from the *right*) and upper four-bit 'nibbles': 390-430 deal with the lower nibble, 440-470 deal with the upper, and 480 returns to the scanning loop. Note the hexadecimal-to-ASCII adjustment in 410-420 and 450-460, converting values of 10 and above to the letters A-F; again, check with the character-graphics chart if you don't understand how this works.

```
490 CH=PEEK(CU-2*L): CH=CH+1: IF CH=58 THEN CH=48
500 POKE CU-2*L,CH: ST=PT+L+12: N=0
510 FOR P=0 TO 2: CH=PEEK(ST+P)–48: N=N+CH*10 ↑ (2–P))
520 NEXT P: IF N>=256 THEN CH=PEEK(CU-2*L)-1: GOTO 500
```

This routine jumps to here if the cursor is under the decimal section of the display. Here the value is incremented rather than simply flipped (because decimal has ten possible values rather than two!); note also the check in 520 to make sure that the maximum total value of 255 cannot be exceeded. If if has, the routine loops back to decrement the digit again; if not, the routine continues.

```
530 FOR Q=0 TO 7: B(Q)=N-INT(N/2)*2: N=INT(N/2): POKE PT+L+8-Q,B(Q)+48
540 NEXT: GOTO 390
```

The binary values are now dealt with; the routine then jumps to 390, to change the displayed hexadecimal value, and thence back to the scanning loop.

```
550 CH=PEEK(CU-2*L): CH=CH+1: IF CH=71 THEN CH=48
560 IF CH=58 THEN CH=65
```

The routine jumps to here if the cursor is under the hex section. Again, the digit is incremented; note the checks to handle the ASCII jump from F to 0 (550) and 9 to A (560).

```
570 POKE CU-2*L,CH: ST=PT+L=18: N=0
580 CH=PEEK(ST): IF CH>65 THEN CH=CH-7
590 N=N+(CH-48)*16
600 CH=PEEK(ST+1): IF CH>=65 THEN CH=CH-7
610 N=N+CH-48: T=N: W=1: GOSUB 360: W=0: N=T: GOTO 530
```

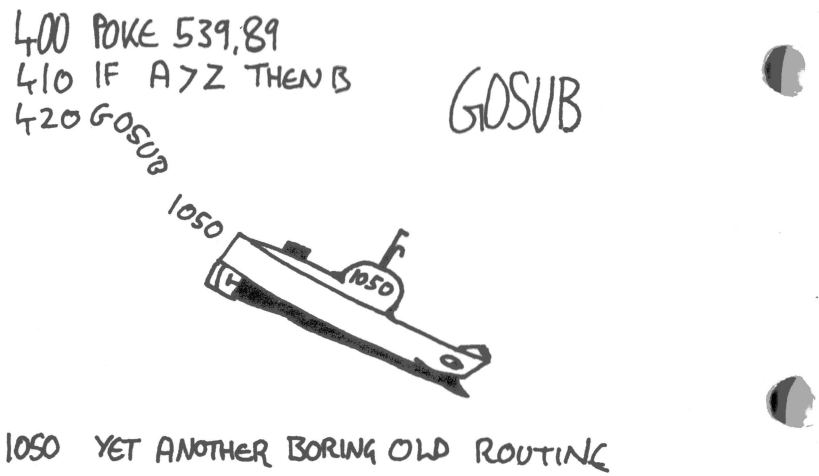
Here the routine calculates the new total value; again, note the checks needed in the ASCII to hex conversions (IF CH>=65 THEN CH=CH-7). 610 calls the 'sometimes subroutine' (360-380) to print up the decimal value, saving the N total in a temporary store T; it then exits by printing up the binary value.

The whole program is somewhat slow, and could do with some general tidying! But it is interesting to watch how changing a value — particularly a decimal one — causes a change to ripple through the other number-base displays. Steve is intending, incidentally, to expand this rough program into a full machine-code tutor, showing the effects of arithmetic and shift operations on the values, the 'carry' bit and the status flags; so we'll publish the relevant amendments in later parts of this series.

We have wandered somewhat from our original intention in this series of describing machine-code concepts in terms of BASIC! But we did feel that understanding some of the concepts needed some background as to what machine-code actually does, from the machine's point of view — hence the disassembler of last issue, and the BASIC program of this. Next issue we'll return to this series' original theme, and develop a BASIC description of machine code.

The main difficulty in explaining machine-code via BASIC is in its very extensive use of 'flags': the status register's flag-bits are set and cleared with almost every operation, something that would not normally be done without extensive subroutines. We'll discuss ways round this difficulty in the next issue.

Elsewhere next issue we'll deal with extensions to BASIC itself, as promised. A couple of members have already sent in suggestions, complete with the code to implement them for either BASIC-in-ROM or in 65D; any further suggestions would be most welcome!



8-inch floppies for the Superboard

David Kaye

Expanding my Superboard to a disk system would have entailed the adding of a 610 expansion board along with a standard 5¼in minifloppy drive to make a standard configuration. However, as the 610 board and minifloppy drive are not cheap, and there was a possibility of an 8in drive being acquired, I decided to attempt to attach a full-size floppy disk to my system.

I obtained a copy of the Howard Sams C1 manual, which showed that the circuitry used open-collector techniques for driving the disk unit. This suited my DRI Model 74 (Orbis 74) unit; however, the labels for the J3 connections on the 610 board are not given on the diagram! *[J3 connects the 610 board to the drive header cable. — Ed.]*

I contacted Mutek to obtain a 610 board, and they kindly provided information on the disk-driver circuitry for the C3 and C4. The C3 diagrams not only provide the labels for the I/O lines but also the changes required to drive 8in disks. As far as the disk driver itself is concerned, the circuit is identical, except that R20, which is connectd from the +5v line to the RxData trimmer R19 and thus to U70 pin 14 via a .001 capacitor C39, must be changed from its current 18K ohms value to 4K7 ohms to suit 8in drives. The pulse width may then be adjusted (with R19) to 2.5 microseconds via the procedure given in the Sams manual.

This change is all that is required to read 8in disks. However, as the data-clock rate is 250kHz for 8in and 125kHz for 5¼in, the TxCLK line needs to be moved from U12 pin 8 to U12 pin 9. This can be easily done on the solder pads provided, firstly cutting the printed link.

Having obtained the 610 board, I increased the total system memory to 24K (32K is possible, including the Superboard's own memory) and obtained a header card to plug into J3. The header card is designed, of course, to connect up to two minifloppy drives to the system. It does not contain any active components, as it merely acts as a connecting board, but it showed that a number of I/O lines were permanently set as shown in this table:

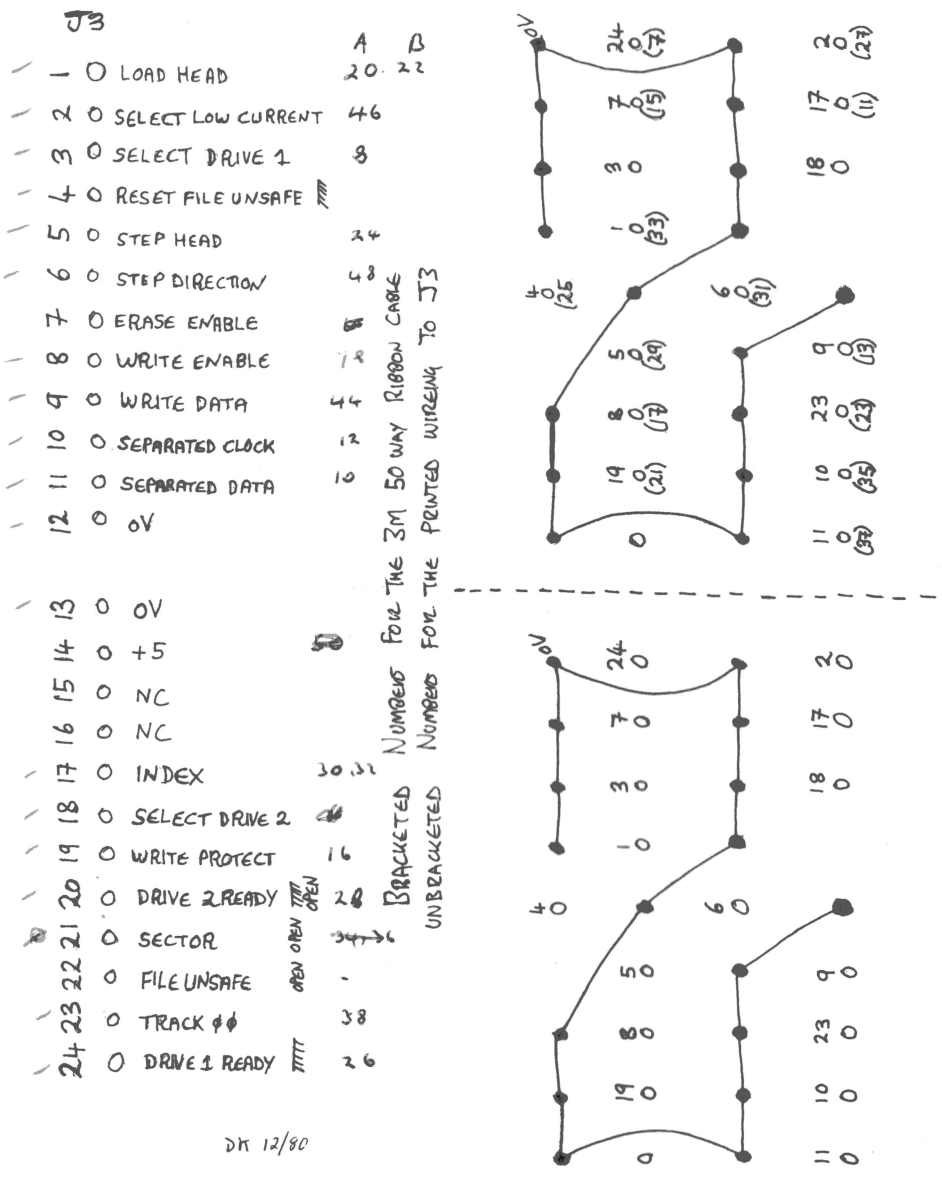
J3	*On header card*	*Mnemonic*
4	is tied low (active)	Reset file unsafe
20	is tied low (active)	Drive 2 ready
21	is not connected	Sector
22	is not connected	File unsafe
24	is tied low (active)	Drive 1 ready

The header card is used only as a wiring platform for the 50-way ribbon cable I used on my 8in drive (see figure 1). I have, however, added DIL switches on the drive to enable the above lines to be set.

The following table shows the lines to the 50-way connector on the Orbis drive:

Signal	*Pin in J1*	*Disk*	*I/F*
Reserved	1		
Return	2		
Key	3		
Key	4		

Header Card



Signal	*Pin in J1*	*Disk*		*I/F*
Read data	5		→	
Return	6		+	
Ready	7		→	
Return	8			
Sector*	9		→	
Return	10			
Index	11		→	
Return	12			
Write data	13		←	
Return	14			
Erase gate	15		←	
Return	16			
Write gate	17		←	
Return	18			
File unsafe	19		→	
Return	20			
File protect*	21		→	
Return	22			
Track 00	23	→+24v		50
Return	24			
Unsafe reset	25		←	
Return	26			
Low current	27		←	
Return	28			
Step	29		←	
Return	30			
In (direction)	31		←	
Return	32			
Load head	33		←	
Return	34			
Sep clock*	35		→	
Return	36			
Sep data*	37		→	
Return	38			
Reserved	39			
Return	40			
+5 volts	41			
+5 volts	42			
−12 volts	43			
0 volts	44			
+24 volts return	45			
+24 volts return	46			
+24 volts return	47			
+24 volts	48			
+24 volts	49			

CEGMON Notes

Justin Johnson's disk linker

A number of points have arisen from this — not least being the surprisingly large number of disk systems around!

For starters, there is a bug in the listing given last issue. Delete line 120 (120 DATA 4353) — this should not be used, as the linker crashes on DIM statements if it is used.

Rather more important, you must keep at least one disk with the original unlinked operating-system — or else you won't be able to copy disks or work on track Zero! The CEGMON/SYNMON pointers are swapped into Page 2, from $0200 upwards, and are used by all input/output calls, including those from the operating system. The 65D track-zero/copy utility, however, occupies the same space — CALL 0200=13,1 (CALL 0200= 01,2 on 8" systems) — and neatly overwrites the lot, crashing all input/output and plenty more besides! As mentioned above, keep at least one master copy of 65D unlinked.

The routine as published applies to Justin's own homebrew display; the DATA for the page-2 locations has to be changed for different machines. There was a note to this effect at the tag-end of the Dealer Notes section of last issue; but in case you missed it, here it is again: after the DATA 8 in line 1030 the values are machine-specific, and should be copied from locations 64434-64462 ($FBB2-$FBCE) on your own version of CEGMON. (The locations are different — 64464-64490, $FBDO-$FBEA — on Superboard Series 2/Superboard III CEGMONs: see below). Line 1070 seems to be redundant, in that the values of that line (stored in $0235-$0240) are not actually used by CEGMON.

Because of differences in the assembly, the linker for the new CEGMON version for Series 2 Superboards is rather different. Steve Hanlan of Beaver Systems rewrote the linker for us:

```
10 X=PEEK(10950):POKE 8993,X:POKE 8994,X
15 POKE 741,76: POKE 750,78
20 POKE 2073,173: POKE 2893,55: POKE 2894,8
25 L=PEEK(133): POKE 133,L-1: REM reserve top page of memory
30 POKE 8960,L-1: FOR J=712 TO 767: A=PEEK(J)
35 POKE J+(L-2)*256,A :NEXT :REM swap look-up to top page
40 READ N: IF N=999 Then J=511: GOTO 60
50 POKE N+2,L: GOTO 40 REM insert links into BASIC
60 READ N: IF N=-1 THEN 80
70 J=j+1: POKE J,N: GOTO 60: REM swap CEGMON pointers into page 2
80 POKE 1419,2556: POKE 11430,255
85 POKE 8979,54: POKE 8980,248: REM series 2 screen handler
90 POKE 9522,209: POKE 9523,250: REM series 2 input
95 POKE 1382,5: PRINT CHR$(26);
100 PRINT "CEGMON linked .... System open": NEW
110 DATA 2037,2041,8908,999
120 DATA 0,32,10,0,1,0,0,8,43,11,35
140 DATA 10,10,0: REM SYNMON-type pointers
150 DATA 180,250,155,255,57,255,112,254,123,254: REM vectors
160 DATA 47,138,208,138,211: REM screen table values
170 DATA 189,138,208,157,138,211,202,96
180 DATA 0,64,138,208,-1
```

CEGMON for Series 2 Superboard

After a mammoth re-assembly, CEGMON for Series 2 has finally arrived. There was just one byte free in the original CEGMON — we had to find just over sixty for the new routines! The screen-format swap is now supported by CHR$(6) (or CTRL-F if BASIC's masking is bypassed); you can also 'freeze' listing on the screen by holding down the 'CTRL' and 'REPEAT' at the same time. The only feature dropped is the former User jump in the machine-code monitor. An awkward point for some users — particularly software houses — is that almost all of the routines have been moved. Not surprising, really — 'inventing' more than forty bytes in a major system demands a fair bit of reshuffling!

CEGMON and Intamon

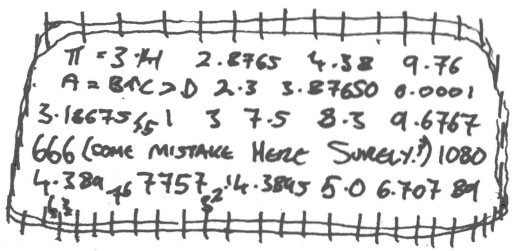
CTS in Lancashire are distributing a new monitor for the Series 2 Superboard, written by a software house by the name of Intasoft. It adds a new range of BASIC commands (mostly for screen-handling, such as .CLS), a twin-cursor screen-editor, and a programmable screen-handler; it does, however, have only limited machine-code facilities. Sounds familiar? Yes, you're right. On our estimate, just under 20% of the code is original — exchanging CEGMON's machine-code monitor for one of *Micro*'s 'hooks' into BASIC. The rest is mostly CEGMON original — just over 55% — including the screen-handler (unchanged), the editor (unchanged except for the actual key values — awkwardly using the 'repeat' key), the CEGMON keyboard unscramble and, amusingly, the special patch to allow CEGMONs to run on UK101s (Intamon, however, is not designed to run on the UK101s...). 15% of the code has even been copied byte-for-byte, let alone line-for-line — all of it CEGMON additions rather than OSI original. We are not exactly happy about this; neither, we are glad to say, is Pete Fawthrop of CTS, who was under the impression that Intasoft were producing an entirely new monitor with features like PRINT USING — which it doesn't have. Pete Fawthrop has a reputation for service and straight dealing, and we gather that some kind of licensing deal is being arranged.

Annoying though this is for us, it does mean that Intamon is almost exactly software-compatible with CEGMON. The notes in this column will, in almost all cases, apply to systems running under Intamon.

CEGMON-D

The trials and tribulations of CEGMON-D continue! As expected, OSI's software for disc systems is almost perfectly incompatible with itself — 65D, for example, has two separate backspace routines and three entirely different methods of printing messages; while 65U, a very impressive operating system, continues to confuse us, because it consists of a maze of machine-code patches into the already heavily-patched code of 65D BASIC. The CEGMON-D package will eventually consist of the PROM together with a small board carrying the extra circuitry for its own independent RAM. Prototypes are available, but the long job of weeding out OSI's incompatibilities will evidently continue for a while yet.

DATA FIELD



Utilisation of unused video RAM on the C1E

Chris Forecast

The Mutek 48 by 32 display conversion of the C1P requires the implementation of 2K video RAM. However it is obvious that a 48 by 32 display only requires 1536 bytes of RAM. So it is clear that 512 bytes of the video RAM remain unused.

If we look at the video memory map of the 48 by 32 format display we can see that the unused memory occupies a block "to the right of the screen".

Unfortunately this memory does not occupy consecutive locations but 32 blocks of 16 bytes at 64 byte intervals.

A very useful way of utilizing this memory is to define a function FNA(X);

```
DEF FNA(X)=53295+X+48*INT((X-1)/16)
```

This gives us a function such that FNA(1) is the first location of the unused RAM and FNA(512) is the last location. So we have a way of consecutively labelling the spare RAM. This can be very useful where we need to use a large array containing integers (0 to 255), since we have an array of up to 512 elements which may be set up by POKE statements and read by PEEK statements.

Hence instead of saying something like A(4)=62 we use POKE(FNA(4)),62 and likewise an expression like P=A(4) becomes P=PEEK(FNA(4)). Whilst this method is only useful for an array of dimension 511 or less, containing integers from 0 to 255, it still provides a tremendous memory saving when used.

Now to the disadvantages. Because the spare RAM is video RAM, scrolling must be avoided, as this would rearrange the contents of the proposed array. This means that the system is only really suitable for those machines using CEGMON, where video RAM areas outside the current 'window' are not scrolled.

It should be noted at this stage that ?CHR$(26) clears the entire array. So assuming we're using the full screen window as bound at start-up then screen clears should be executed using ?CHR$(30).

A similar technique could be used on the UK101 where 768 bytes of RAM are used for the video display, leaving 256 bytes of unused RAM. And also on the Superboard II where 625 bytes of RAM are used for the display, leaving 399 bytes of free RAM. However in the case of the Superboard only 244 bytes (to the 'right' of the screen) may be used in this manner.

The appropriate values for the UK101 and Superboard are:

UK101: DEF FNA(X)=53295+X+48*INT((X-1)/16) 0<X≤256

Superboard: DEF FNA(X)=53275+X+25*INT((X-1)/7) 0<X≤224

HELP!

We do try to satisfy everyone's needs, but there is a definite limit to what we can do ourselves in the way of research and experimentation — and we overshot that limit some months ago! We need help in sorting out the many queries that arise, and in researching a number of specific items that have arisen over the past few months.

A few issues back I asked for volunteers to help check out queries, particularly on the C1 and UK101. A lot of members replied — and to them I must apologise, because while I managed to acknowledge a few, I haven't been able to invent the time to sort out and distribute the mass of queries! In the meantime, plenty more queries have come in....

We also have the problem of this Newsletter's direction. We (George, Richard and myself) now have some two years' experience of OSI systems behind us; we are apt to forget

that some of our members may have only had their machines for a couple of weeks by the time they receive their first Newsletter. We are perhaps too deeply involved in machine-code — an interest reflected by many members, but certainly not by all! We do feel that the proper role of the Newsletter is one of documentation and description — we tend to prefer utility programs to games, for example, on the assumption that these will be of wider use — and that lenghty programs are best described in the 'comics' like *PCW* and *PC*. For UK101 members, there is also the *UK101 Users Group* (it advertises itself as 'Computer Aids' in the comics), which specialises in general software. But you may feel otherwise: should we limit the Newsletter to dealing only with BASIC-in-ROM programs, for example?

In any case, here are a number of specific items that now urgently need research:

New monitors — There are now a bewildering variety of monitors for the OSI and Comp machines, particularly for the C1/Superboard and UK101. The software houses are going quietly crazy trying to make their software automatically adjust for the different input/output routines, screen handling and God only knows what else! We're getting a similar stream of queries from members, mostly on the lines of "My existing software won't run on *nnn*Mon, why not?" So we need to collate a list of the start locations of all manner of routines in all of the monitors: not just the input/output vectors, but points like the machine-code monitor's load entry point, which is used for auto-load of machine-code in mixed BASIC/machine-code programs. We already have listings of OSI's SYNMON, SYN600 and 65A (hard disk) ROMs, of the CEGMON variants and of CTS's depressingly similar IntaMon; so we particularly need to know about Watford's WEMON, the range of Comp monitors from the original UKMon to NewMon02, and the ROM issued by the author of the *Practical Electronics* editor program — and any other monitor you may have come across. We would like to do a feature on this next issue — but please send us a simple list of entry points, or else we may find ourselves landed with a copyright action if we publish more!

WAIT — WAIT is the only BASIC-in-ROM instruction left that has not been covered either in the original documentation or in this Newsletter. We particularly need to know about uses for its ORing and EORing sections on the full WAIT instruction WAIT I,J,K.

Disk BASIC — in earlier issues of this Newsletter we dealt with the zero-page pointers and storage formats of ROM BASIC; we have not yet had the time to complete the list, or to do anything of the kind for the disk BASICs. In addition to information on zero page usage, we also need to update the somewhat limited POKE list into disk BASIC itself.

Assembler utilities — the Assembler is short of some useful utilities which would make it a really powerful tool. One which know I need in my work is a 'search' facility — one which would list out the line numbers in which a given label appears. Easy enough in BASIC, not so easy in Machine-code — any takers?

Software conversions: OSI/Pet/Apple/Acorn and others — most 6502-based machines use POKE-type graphics for one reason or another. It would be useful to have a video memory map for the non-OSI machines, and especially — if anyone fancies tackling it — a program to convert graphics from one type to another.

The back page program

By popular request(!) we're including a regular short-program feature, the 'Back Page Program'. As it happens, this one, by *C. Hurt* of Bristol, is a CEGMON utility; but what we'd like for this section is lighter items — games, puzzles and the like. Keep the program — BASIC or machine-code — under *fifty* lines total, so it will fit on the back page; and please send it to us either as a *clean* print-out or on cassette or disk — that way there's less chance of typo errors creeping in. Let's see what you can send us for next issue — all contributions will be acknowledged on this page!

```
10 REM * UK 101 - 32 X 48 FORMAT *
20 REM * CEGMON MONITOR 101 E *
30 CL$=CHR$(26):PRINTCL$;
40 PRINT:PRINT:POKE518,30
50 PRINTTAB(5)"PROGRAM TO CALCULATE"
60 PRINT:PRINT
70 PRINTTAB(8)"LOW & HIGH BYTE NUMBERS"
80 PRINT:PRINT
90 PRINTTAB(11)"FOR SCREEN-HANDLER"
100 PRINT:PRINT:PRINT
110 PRINTTAB(7)"Columns - 1 to 47  (CL)"
120 PRINT:PRINT
130 PRINTTAB(7)"Lines   - 1 to 32  (X top)"
140 PRINT
150 PRINTTAB(26)"(Y base)"
160 PRINT:PRINT:PRINT:PRINT
170 PRINTTAB(5)"After first - press space bar"
180 PRINT
190 PRINTTAB(18)"For next"
200 FORP=1TO7000:NEXTP
210 PRINTCL$;
220 PRINT
230 INPUT"  WINDOW WIDTH";W
240 PRINT
250 INPUT"  COLUMN NUMBER";CL
260 IFW+CL>47THENGOTO230
270 IFCL<1ORCL>47THENGOTO250
280 PRINT:PRINT:PRINT
290 INPUT"  START - No.TOP LINE";X
300 IFX<1ORX>32THENGOTO290
310 GOSUB520:GOSUB620
320 TL=K+CL-1
330 PRINT
340 PRINT" WINDOW -    TOP LEFT(Bytes)    LOW    HIGH"
350 PRINT
360 PRINTTAB(31);TL;"  ";M
370 MT=M
380 PRINT:PRINT:PRINT:PRINT
390 INPUT"  START - No.BOTTOM LINE";Y
400 IFY<1ORY>32THENGOTO390
410 IFY<XTHENGOTO390
420 GOSUB570:GOSUB630
430 BL=L+CL-1
440 PRINT
450 PRINT" WINDOW - BOTTOM LEFT(Bytes)    LOW    HIGH"
460 PRINT
470 PRINTTAB(31);BL;"  ";M
480 MB=M:GOTO650
490 POKE530,1:POKE57088,253
500 IFPEEK(57088)<>239THENGOTO490
510 IFPEEK(57088)=239THENGOTO210
520 IFX/4=INT(X/4)THENK=205
530 IF(X+1)/4=INT((X+1)/4)THENK=141
540 IF(X+2)/4=INT((X+2)/4)THENK=77
550 IF(X+3)/4=INT((X+3)/4)THENK=13
560 LB(1)=K:RETURN
570 IFY/4=INT(Y/4)THENL=205
580 IF(Y+1)/4=INT((Y+1)/4)THENL=141
590 IF(Y+2)/4=INT((Y+2)/4)THENL=77
600 IF(Y+3)/4=INT((Y+3)/4)THENL=13
610 LB(2)=L:RETURN
620 C=INT(X/4.1)+1:GOTO640
630 C=INT(Y/4.1)+1
640 M=207+C:RETURN
650 PRINT:PRINT
660 PRINTTAB(5)"POKE546,";W
670 PRINTTAB(5)"POKE547,";TL;":POKE548,";MT
680 PRINTTAB(5)"POKE549,";BL;":POKE550,";MB
690 GOTO490
```