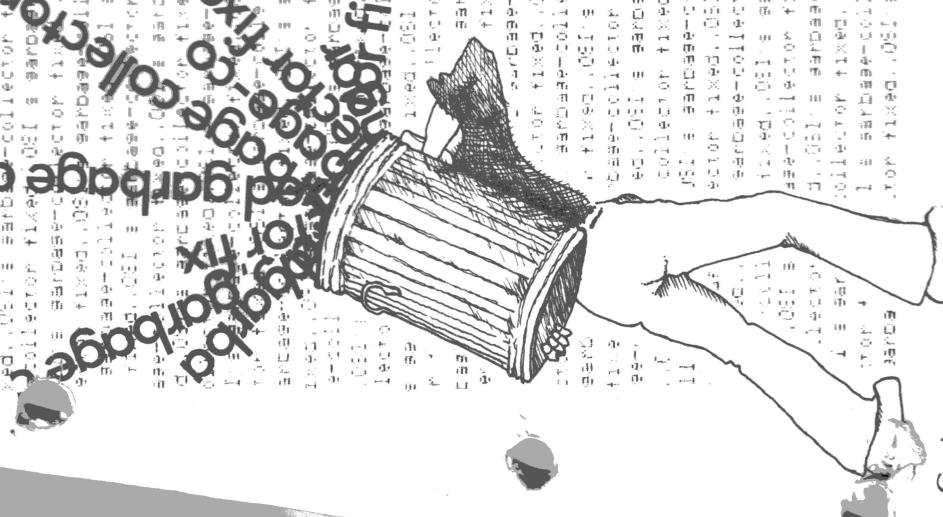


OSI UK User Group Newsletter

Vol.1 No.4

Vol.1 No.4



Garganee

Editorial

In case you thought we'd forgotten you, we're back again, with the usual miscellany of notes and ideas. Major items this time are an article by Ray Fox on RS-232 interfaces for printers, and a feature on the dreaded 'garbage collector' problem in BASIC's string-handling (solved at last by Dick Stibbons).

Following on from the notes in the last issue, we've been very busy incorporating everyone's ideas and requests into a new ROM for all the OSI and UK101 BASIC-in-ROM machines — and driving ourselves more than halfway round the bend in the process! (Its long pre-production testing period is the main reason why this issue is slightly late). But CEGMON, the new ROM, is complete and very extensively tested, and has been available for a couple of weeks now as I write this. More details elsewhere in this issue; but since it does contain all of the features that everybody asked for — such as true rubout, editing, screen-clear and machine-code save — and quite a lot of the other ideas — like a programmable screen-handler, and a fair chunk of the Extended Monitor now in ROM — we feel it ought to 'ease our members firmware headaches' as promised!

This issue is the last of the present volume, so if you were one of the many members whose subscription started with or was backdated to the beginning, you'll find a renewal/resubscription form in this copy of the Newsletter. Everyone else will be getting theirs as their four-issue sub runs out.

The good news is that we are going bimonthly — judging from the number of phone calls I get as each issue approaches, three months is too long to wait between 'fixes' for many of you! We also now have enough members to make bimonthly production feasible. The bad news, of course, is that the six-issue sub will be twice the original one, namely £10 — partly because of the higher effective production cost per issue, partly because of ever-increasing postal charges (up again in November, and partly because inflation and the like hurt us as much as everyone else! But we hope that the increase will not be a painful one, and that you believe it to be worthwhile.

We're very glad to see that this Newsletter is indeed becoming everyone's, rather than merely a product of your editor's pen. In addition to the articles here, we also received two implementations of FORTH for UK101 (from Bill Powell of the British FigFORTH group, and from Roger Cuthbert), on which more next issue; a complete text editor, again for the UK101, from Peter Maughan; games from Neil Cannon and Dave Caine; and many phone calls and other comments for which I have, of course, lost the few notes that I made. Many thanks to everyone — it makes our work worthwhile as far as we are concerned, and we hope it does for you too.

We do definitely want to hear from you, and to know what you are doing. But some people seem unclear as to our addresses and what we do, so:

Hardware/technical development:

George Chkiantz and Richard Elen at 12 Bennerley Road, London SW11 6DS.

Editor/documentation development:

Tom Graves at 19a West End, Street, Somerset BA16 0LQ.

Documentation Corner

Continuing CLEAR

Jack Pike says "I use CLEAR to clear variables and strings when trying to find out how much RAM the program text occupies. More importantly, though, I have used it for checking whether the program will run with restricted RAM (eg. when squeezing an 8K program into 4K) or to test 'error capture' in programs which are protected against having insufficient RAM to run. A typical substitute for RUN would be: CLEAR: DIM DUMMY(FRE(0)/4 -4): GOTO first program line

I am aware that input of any unused line number (eg. 0) also 'CLEARs' the variables etc. but I prefer to use CLEAR. I think it is definitely NULL for the booby prize!"

Peter Maughan comments that he uses CLEAR in order to wipe all existing values when 'chaining' sub-programs, so that all-too-commonly-used labels like I, T, X and Y start afresh each time rather than picking up any value they may have had in the previous sub-program.

INPUT again

Jack Pike's problem mentioned last issue, that of including commas and colons within a string INPUT, is still unsolved; but several members (Roger Beaumont among them) have pointed out that you can build an 'INPUT' via a USR call to the keyboard, looking for your own delimiter rather than for commas, colons, quotes or whatever. The basic sequence is:

```
1000 POKE 11,0: POKE 12,253 : REM set up USR call to keyboard subroutine  
1010 A$ = "" : REM clear the 'INPUT' string  
1020 X = USR(X): B$ = CHR$(PEEK(531)) : REM CHR$(PEEK(533)) under CEGMON  
1030 IF B$ = delimiter THEN RETURN  
1040 IF LEN(A$) > 254 THEN RETURN : REM or do your own error trap  
1050 A$ = A$ + B$  
1060 GOTO 1020
```

The danger with this dummy 'INPUT' is our old friend the 'garbage collector' — if you have any string arrays in your program, the reshuffling of string space that this routine will demand will cause the garbage-collector bug in BASIC-in-ROM to trash the program.

Jack Pike sent in another comment — not a puzzle this time! — to remind us that INPUT supports the use of multiple commas — „„„ — to INPUT zeroes or, more importantly, empty strings. This feature is useful, he says, when trying to INPUT a variable number of significant values or strings to a program, for by holding the comma key down after the last significant value, the auto-repeat rapidly gives more than enough commas to satisfy the INPUT. The extra commas are ignored, so the string length is not critical. But it is a pity, he comments, that the „ construct was not arranged to support a 'null' data input which left the variable values unaltered.

The other OM ERROR — out of memory in the stack

David Cannon writes: "Being short on line numbers, I extended a program by a GOSUB. Later I modified the subroutine to GOTO part of the main program. All went well until the eighth or ninth time through the game, and then 'OM ERROR' — out of stack space!"

"Being used to programming in CORAL 66 which allows you to jump out of

subroutines, it took me ages to sort out this bug. Rub the point home to the other members and save them time."

GOSUBs, and also FOR:NEXT loops, push their return addresses on the stack. If you jump out of a GOSUB, and in certain cases out of a FOR:NEXT loop, these return addresses are left untouched; the next time round pushes the same return address, and so on up the stack. The out-of-memory check for the stack is a little bizarre in BASIC-in-ROM because of the way OSI has organised its IRQ and NMI locations (right in the middle of the stack!) — for those interested, the error check is at \$A212.

The reason why you get an OM ERROR after many immediate-mode actions (like a POKE) after warm-start is because BREAK resets the stack to \$(01)28; warm-start, however, expects the stack to be at \$(01)FF! — and that is where it resets it after its OM ERROR call, or after any ERROR. If you want to do a string of POKEs after a warm-start, force a SN ERROR (syntax error) first, by typing a non-command such as RUBBISH!

Aligning output on the decimal point

In last issue Matthew Soar gave us a function to align numerical output on the decimal point. It was, of course, reproduced without one of its brackets! Two other functions to do the same job, but for all numbers, were sent in by J.R. Parkes and Ray Fox. J.R. Parkes' function is:

```
PRINT TAB(D-INT(LOG(X)*0.47+3));X
```

Ray Fox says that to cater for numbers greater than or equal to 0, use the same routine as last issue, but with the following function:

```
DEF FNP(X) = -LEN(STR$(INT(X)))-(ABS(X)<1)+(X)
```

However, he says, the function can be made very general, catering for numbers whose values range from negative through to positive. Then:

```
DEF FNP(X) = -LEN(STR$(SGN(X)*INT(ABS(X))))-(ABS(X)<1)+(X=0)
```

where incidentally SGN(X)*INT(ABS(X)) equates to FIX(X) — a command not available in OSI BASIC.

'Missing' characters in C2-to-Base-2 printer interface

It may be found as in my case that when outputting a stream of data via the RS-232 interface to the printer which includes CR/LF or Vertical Tabs etc. (i.e. functions that cause the printer to a mechanical operation apart from actually printing) followed immediately by data, that the first character of the data is lost. Whilst setting NULLS will solve the problem at the beginning of each new line, it does not do so where Vertical Tabs are concerned. I have found it necessary to delay the CRT print rate by POKEing a delay into \$0206 (518₁₀) — this does not actually slow down the SAVE print rate but allows the printer to return a busy signal before the computer packs a second character into the double-buffered ACIA. The values I use are: POKE 518,3 when dumping at 4800 baud; or POKE 518,80 when dumping at 300 baud. My C2 is operating at 2MHz however, so it will be necessary to change these values for 1MHz machines — if in fact the problem exists at that speed. I understand C1 users have had no problems. Incidentally, to operate the ACIA at 4800 baud on a C2 do POKE 64512,179: POKE 64512,176 in immediate-mode.

Ed: The reason why the character buffer can be over-written can be seen in the machine-code of the output loop in the monitor ROM and in BASIC's PRINT and output loops. As soon as the ACIA has taken the character at the end of the loop, another character is thrown at the screen and then to the ACIA, without any internal

delay at all. The only delay is NULL, collected after a carriage-return. In some cases it would seem that, particularly at 2MHz machine speed and 300 baud transmission speed, the next character is thrown into the ACIA as soon as the old character has been taken away — in other words before the printer has had any chance to process it and, in the case of mechanical operations, to send out a 'busy' signal. POKEing 518 with a delay (which operates within the screen-handler) delays the effective output to the ACIA to the point where the Base-2 does have time to catch up; but too long a delay will take the problem the other way, with characters being repeated in the main run. Experiment with delay values as required — particularly if you've built a 'home-brew' interface!

BASIC block-delete

One of the more annoying absences from the command list in OSI's BASIC is a block-delete of program lines. David Caine sent us this routine, which works by 'sending' BASIC a long string of 'empty' line numbers. It's a little slow for clearing large or widely spaced blocks — but it's better than laboriously typing numbers in by hand!

```
62000 FOR I = 7936 TO 8026: READ J: POKE I,J: NEXT  
62001 DATA 162,199,189,131,162,157,33,2,202,208,247,173,89  
62002 DATA 31,141,202,2,173,90,31,141,203,2,169,13,32  
62003 DATA 45,191,165,240,133,173,165,241,133,174,32,98,185  
62004 DATA 162,0,134,14,189,64,215,201,95,240,6,149,19  
62005 DATA 232,76,43,31,169,0,149,19,133,196,169,18,133  
62006 DATA 195,76,34,2,230,241,208,2,230,240,56,165,243  
62007 DATA 229,241,165,242,229,240,16,193,76,116,162,69,31  
62010 PRINT: PRINT "Block Delete": INPUT "Start, Stop";I,J  
62011 L1 = INT(I/256): POKE 240,L1: L1 = I-(256*L1): POKE 241,L1  
62012 H1 = INT(J/256): POKE 242,H1: H1 = J-(256*H1): POKE 243,H1  
62013 POKE 11,0: POKE 12,31: X = USR(X)
```

The routine as written is for an 8K C2, and will need some adaptation for other machines — as it stands it will not run on a C1, or under CEGMON (part of the routine starts at \$0222, where CEGMON's screen look-up table starts). Two data bytes in line 62004 (64, 215) are the screen address where the cursor appears after a carriage return on a C2 (40, D7 hex) — these will need to be changed accordingly for C1/Superboard and UK101.

Notes on RND

The notes on RND which we mentioned in last issue were from John Partridge. Following on from the reference we made to random-number generation in Newsletter 2, he says:

The random number is stored as a 4-byte binary number (see Newsletter 2), and this is changed when RND is called. The first random number loaded on Cold-Start is 80 4F C7 52 — and exactly the same sequence will be followed after each Cold-Start. The first 938 random numbers produced do not repeat, but after this a loop of 1861 different numbers is produced, which then repeats continuously.

Changing RND(1) to RND(2) or any other positive integer does not affect the sequence at all. Using RND(0) will give a repeat of the last random number called. Using RND(X), where X is a negative integer, will start a new and different number sequence, which in most cases degenerates into the same 1861-number loop as

before. However, a few of these degenerate into a loop of only 279 different numbers. The best that I have found is started by calling RND(-5) before using RND(1) in the usual way: this gives about 5000 different numbers. On the other hand, calling RND(-79) before RND(1) results in only 403 numbers.

The RND routine is located at \$BBC0 and can be called from a machine-code program, with a new number being taken from \$D4-D7 If a new range of random numbers from \$00-FF is required, then \$D6 or \$D7 should be used. A smaller range can be produced by using the AND operation. For example:

```
LDA $D6
AND $07
```

TAY

will leave in Y a random number in the range 0 to 7. I have used this in a machine-code program to produce random movement on the screen (see later). For those interested, the repeating sequences can be shown with this short program:

```
10 R=RND(-79)
20 R=RND(1): IF R>0.999 THEN PRINT X,R
30 X=X+1: GOTO 20
```

— but the exact entry to the loop of numbers takes more finding!

Finally, a short program to show Brownian movement, using running entirely in machine code and using BASIC's RND function. The BASIC program is the loader for the machine-code; it is given here as for an 8K machine.

```
120 REM - Brownian Movement - M/c Mk.III
130 REM - Runs entirely in machine-code once going
140 REM - To stop, press BREAK
1000 FOR X=7936 TO 8016: READ A: POKE X,A: NEXT
1004 REM - N=Number of dots - do not exceed 36
1005 N=32
1010 A=0: POKE 7986,2*N
1014 REM - Puts start locations in page-0
1015 FOR X=0 TO N: POKE 14+2*X,208+A: A=A+1: IF A>7 THEN A=0
1020 NEXT
1030 POKE 11,0: POKE 12,31
1040 FOR X=0 TO 30: PRINT: NEXT
1050 P=USR(P)
2000 DATA 174,80,31,169,32,129,19,165
2010 DATA 214,41,14,168,24,216,185,64
2020 DATA 31,117,19,149,19,232,200,185
2030 DATA 64,31,117,19,201,208,16,2
2040 DATA 169,215,201,216,48,2,169,208
2050 DATA 149,19,202,169,166,129,19,232
2060 DATA 232,224,64,208,2,162,0,142
2070 DATA 80,31,32,192,187,76,0,31
2080 DATA 1,0,63,0,64,0,65,0
2090 DATA 255,255,191,255,192,255,193,255
2100 DATA 0
2120 REM
2130 REM - For an interesting variation, include 1024 POKE 7940,171
```

Relocating the Extended Monitor (ExMon)

David Butler and Michael Whittle both wrote in about relocating ExMon — since the standard version supplied by OSI and Comp is located at the end of the first 4K, for an assumed 4K machine. This means that the standard version is right in the middle of the RAM on an 8K machine, and incidentally also in the middle of the Assembler. ExMon does have a 'relocate' function of its own, which corrects all subroutine calls and jump addresses, but look-up tables and the like are either scrambled (if they appear to be jumps or JSRs) or left untouched. To move ExMon to anywhere else in memory, its jump-table must be changed by hand after using the 'relocate' — the jump table resides at \$0960-0999 in the standard version. The jumps are in pairs, with the low byte first as usual. To move ExMon to the top of an 8K memory, for example, \$D10 must be added to every high byte — the contents of \$0961, 0963, 0965 and so on. \$0962-0963 are the address for ExMon's 'A' routine (print contents of Accumulator), \$0964-0965 is the address for 'B', and so on to 'Z'.

Michael Whittle included several other comments on ExMon in his letter. One was a complete patch to allow ExMon to 'Save' in the ROM Monitor's hex-digit format rather than the strangely unreliable checksum — the listing is below. This dump routine is shorter than the checksum dumper, so — as he says — 'there is room for another goody', namely a routine to restore the vital (for BASIC) addresses \$00D1-D6, and having done so, to jump to BASIC. In order to implement this routine, \$0994-5 (or their relocated equivalents) need to be set to point to the routine's start address, so that the spare 'Z' command will implement the jump from ExMon to BASIC. Further monitor enhancements are to use SPACE instead of CTRL-J to increment to the next line in 'a' and 'Q' modes (this is more important to UK101 users than those with OSI machines, since the standard UK101 monitor decodes the old LINE-FEED key (i.e. CTRL-J) as 'up-arrow' instead). The change is achieved by changing \$0A0 to \$20 at (standard) locations \$0B70 and \$0D2F. It is also helpful, says Michael, to make the quotes (for ASCII) advance to the next line, enabling an ASCII string to be listed rapidly. This is achieved by changing \$0B81 from \$60 to \$8B.

Listing to change ExMon's checksum save to digit-pair save (addresses as for ExMon relocated at top of 8K memory space).

1EC3	207FFF	JSR \$FFF7	; 'S' entry point — set SAVE flag
1EC6	201C1B	JSR \$1B1C	; get start and stop addresses — store in DC-DF
1EC9	A93A	LDA \$3A	; ':' prompt for jump address
1ECB	2069FF	JSR \$FF69	; output routine — adjust on C2 or under CEGMON
1ECE	20A31A	JSR \$1AA3	; get byte — address high
1ED1	85C1	STA \$C1	; store in C1
1ED3	20A31A	JSR \$1AA3	; get address low
1ED6	85C0	STA \$C0	; store in C0
1ED8	A92E	LDA \$2E	; ':' for address mode
1EDA	2069FF	JSR \$FF69	; output
1EDD	A5DD	LDA \$DD	; start address — high byte
1EDF	20AC1A	JSR \$1AAC	; output as hex pair
1EE2	A5DC	LDA \$DC	; start address — low byte
1EE4	20AC1A	JSR \$1AAC	; output as hex pair
1EE7	A92F	LDA \$2F	; '/' for data mode
1EF9	2069FF	JSR \$FF69	; output
1EC	A200	LDX \$00	; clear X register as pointer
1EEE	A1DC	LDA (\$DC,X)	; get next data byte
1EF0	20AC1A	JSR \$1AAC	; output data byte as hex pair

1EF3	A90D	LDA D\$0D	; 'CR'
1EF5	20B1FC	JSR \$FCB1	; output CR to cassette only — JSR \$BF15 on C2
1EF8	E6DC	INC \$DC	; increment low byte of address pointer
1EFA	D002	BNE \$1EEF	; skip next instruction if low-byte not zero
1EFC	E6DD	INC \$DD	; increment high-byte of address if low-byte was zero
1EFE	A5DE	LDA \$DE	; get end-address, low byte
1FO0	C5DC	CMP \$DC	; compare against current address pointer, low-byte
1FO2	D0E8	BNE \$1EEC	; loop back for next data byte if no match
1FO4	A5DF	LDA \$DF	; get high-byte of end address
1FO6	C5DD	CMP \$DD	; compare with current address pointer, high byte
1FO8	D0E2	BNE \$1EEC	; loop back for next data byte if no match (i.e. not end)
1FOA	A92E	LDA D\$2E	; ?' for address mode (for restart/jump address)
1FOC	2069FF	JSR \$FF69	; output
1FOF	A5C1	LDA \$C1	; get restart/jump address, high byte
1F11	20AC1A	JSR \$AAC	; output as hex pair
1F14	A5C0	LDA \$C0	; get restart/jump address, low byte
1F16	20AC1A	JSR \$AAC	; output as hex pair
1F19	A947	LDA D\$47	; 'G' for ROM monitor 'go' command
1F1B	2069FF	JSR \$FF69	; output
1F1E	A900	LDA D\$00	; get null
1F20	8D0502	STA \$0205	; clear SAVE flag
1F23	4C0918	JMP \$1809	; end — jump to ExMon warm-start
1F26	A206	LDX D\$06	; 'Z' entry point — set counter for 6 bytes
1F28	BD321F	LDA \$1F32,X	; load data saved before overwrite by disassembler
1F2B	95D0	STA \$D0,X	; restore to D1-D6
1F2D	CA	DEX	; decrement byte counter
1F2E	D0F8	BNE \$1F28	; loop back until all restored
1F30	2074A2	JSR \$A274	; jump to BASIC warm-start
1F33	E9D0	SBC D\$D0	; data only — tail-end of BASIC's 00BC subroutine
1F35	60	RTS	; data
1F36	80	???	; data
1F37	4F	???	; data
1F38	C7	???	; data
1F39	EA	NOP	;
1F3A	EA	NOP	; pad to start of ExMon 'V' routine at 1F3B

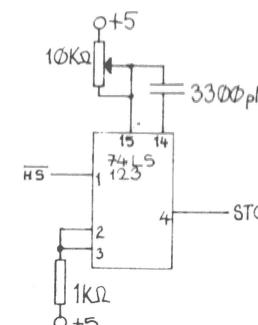
Video display mod

killing 'overscan' on C1/Superboard and UK101

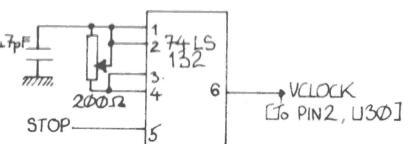
Here is a simple way of achieving the aspiration of many Superboard/UK101 owners — that of getting rid of the 'overscan' limitation to the number of characters per TV line. The solution, whilst not elegant, is extremely simple, costing less than £5.00, using three or four chips and a few passive components. I will only describe the method in general terms, allowing for all the variants of the systems readers may have. But anyone who basically understands the guts of their machine will be able to implement the mod in a few evenings. The question of using the mod to its full advantage is a bit trickier; I use a RAM-based operating so I can use the full line length; others stuck with ROM systems will only be able to use the expanded screen through POKEs during games. [Special versions of CEGMON can be 'blown' if required, however — Ed.]

The crucial aspect of the solution is the provision of two clocks: one for the processor and cassette interface, the other for the video. Inspection of the circuit diagram shows that the first part needs only two signals — the \emptyset_0 -in and the input to the TxCLK system (U57). These are readily provided by taking the 4MHz output of the crystal system (pin 3, U58) and putting it through a 74-163 (or dividing chain equivalent to it) which will provide 2MHz or 1MHz signals for the processor and a signal that will divide down to the equivalent of 600 baud for the cassette system. If 300 baud operation is needed then a further divide-by-two stage (e.g. a 74-74) is simply added.

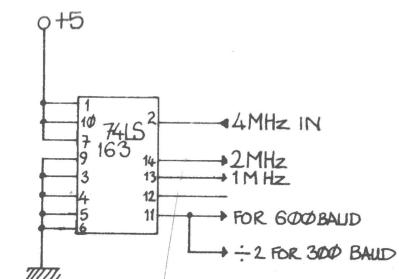
OVERSCAN DELAY GENERATOR



VIDEO CLOCK & GATE



DIVIDING CHAIN FOR PROCESSOR CLOCK



Now for the video part. This all uses the 'CLK' signal — i.e. only one input is required to operate it all. Assuming we have cut the CLK trace with the crystal controlling only the processor and cassette, we need only one other oscillator to drive the whole video system. In principle another quartz crystal would seem desirable but instead we provide an adjustable clock using a Schmitt trigger 74-132 gate with a controllable RC network. We can now get the characters coming out faster than the old system by taking the frequency above 4MHz, and hence get more characters per line. But there's one thing missing, the characters are still being spewed out in the overscan region. To solve this problem all we need do is to stop the clock when you're overscanning! This is very simple to do: when the horizontal sync pulse comes along we initiate a delay equal to the overscan time. The delay pulse gates the clock (using the same '132 gate chip as provides the clock) and all video addressing stops. Because the video system is completely isolated from the processor system, the processor functions perfectly normally; all this stopping and starting of the video clock at its own odd frequency has no effect on the processor or cassette.

It may sound from the description that to set all this up would require sophisticated scopes and timing gear! In fact I don't possess a scope, and all you need to do is get your resistors and capacitors in the right sort of area and then play with the resistances till you get the right combination of delay (so you lose no characters to the left of the screen) and of speed of video output (so you lose none to the right). Surprisingly the delay is very stable, even without crystal control, provided you use decent trim pots. Initial versions with untidy wiring showed some interference between various signals, causing jitter on the screen and, in one bad case, causing the processor to crash at 2MHz. But as soon as the wiring was tidied up (as tidy as Veroboard can be) everything behaved itself. The display 'locks in' to all TVs I've tried it on though, of course, minor adjustments had to be made to suit the differing overscans of the TVs.

Dr. S.J. Abbott

Notes

1 74-123 used because it's familiar, a '121 would do as well. The spare monostable on the '123 might always come in handy.

2 Resistances and capacitances fit my 64×32 Superboard with the resistances in the middle of their adjustable ranges.

3 'STOP' can also be connected to pin 10 of U56 to activate (DB) — this tidies up the screen during sync and delay.

4 Implements very simply on Veroboard.

CEGMON notes

As mentioned elsewhere, last issue's discussion on firmware has brought concrete results in the form of our new monitor, CEGMON. Ads will be appearing for it in various parts of the computing press at the same time as this issue is published, so there's not much point in describing it here — let's simply say that it does pretty well everything you asked for, and a lot more besides. Since its documentation went to press several interesting points have come up, and we therefore give them here.

Access to control and graphics characters

The new keyboard routine allows direct access to control characters and (on OSI machines only) graphics above 128₁₀, using the REPEAT key as a second control key. These do function as expected in the Assembler or in the machine-code monitor — CTRL-Z does clear the screen, for example. But BASIC normally masks out almost all characters below 32₁₀, and all characters above 124₁₀ — typing CTRL-Z in BASIC's immediate mode will not clear the screen, although PRINT CHR\$(26) will. This is explained in the CEGMON user notes, in several places; but judging by the number of phone calls we've received, a fair number of people seem to have fallen for the old trap of 'if all else fails, read the instructions'!

The masking is a limitation of BASIC, and not an error in CEGMON! But we have provided two ways round this: you can either call the keyboard routine (JSR \$FD00) or editor routine (JSR \$FABD) via a USR call from BASIC, which will return the key value without masking; or you can use the 'unmask' routine described in the User Notes, changing the BASIC input vector accordingly. Control characters can then be entered direct into program lines; but note that cursor home, screen or window clear and other embedded commands make LISTings a little bizarre! The only limits are that nulls (the 'racing car' graphic) are always masked off, and LINE-FEED cannot be used as the first character in a line — this is to prevent problems when loading from tape. Note also that the 'unmask' routine must be enabled before loading a tape with embedded control characters or graphics, or they will be ignored during LOAD.

When 'unmask' is in use, a limited 'single-key entry' of some BASIC commands and functions is available, on all OSI machines and on UK101s with the REPEAT key wired in. We didn't actually design this, and didn't realise that it was possible until Steve Hanlan of Beaver Systems pointed it out to us — it is a side-effect of accessing top-bit-set graphics, some of which are decoded by BASIC's tokenising routine as being delimiters of certain keywords. Usefully, these are all mnemonic, because of the way in which BASIC does this — REPEAT-A gives AND, REPEAT-P gives POKE, and so on. For example:

10 REPEAT-P 546,24

is equivalent to

10 POKE 546,24

and will appear as such when LISTed.

This only works with upper-case alphabetic characters, and then only with some of them. And not all the BASIC keywords are accessible in this way — BASIC translates them as being the first keyword in its table which starts with that letter. The complete list of 'single-key' keywords available is as follows:

REPEAT-A	AND
C	CONT
D	DATA
E	END
F	FOR
G	GOTO
I	INPUT
L	LET
M	MID\$
N	NEXT
O	ON
P	POKE
R	READ
S	STOP
T	TAB(— note the bracket!
U	USR
V	VAL
W	WAIT

Memory-fill function in machine-code monitor

The monitor's 'move' function will over-write the code to be moved if the new start address is between the old start and end addresses; but this can be turned to advantage in order to fill a block of memory with either a single value — such as nulls or spaces — or a repeating pattern, by deliberately 'over-writing' during the move.

For example, to fill the lower half of the Superboard's screen memory with nulls ('racing cars'), use the monitor to type:

```
D200/20 00 — i.e. enter a null  
MD200,D3FE>D201
```

This copies the first byte into the new start, which happens to be the next location; it then copies that into the next, which is the new 'next location', and so on. Note that the new start is thus one byte on, and the end of the old block one byte 'early'; adjust these accordingly ('n' bytes on and 'n' bytes early respectively) if you want to put a repeating pattern 'n' bytes long into a block of memory.

UK101 keyboard

The UK101 standard keyboard is not quite the same as that used on OSI systems: there is no ESCAPE key, the REPEAT key is changed into a second BREAK key, and LINE-FEED is changed into 'up-arrow'. The loss of the ESCAPE is no real hardship, and can be wired back in; the same goes for the REPEAT key, especially if the graphics and 'single-key entry' functions are required. But under CEGMON the LINE-FEED key is decoded as such in all its versions, including UK101. 'Up-arrow' is SHIFT-N, as on OSI systems.

Non-standard video systems

Because of the severe limitations of the C1/Superboard standard display, a lot of people have done 'home-brew' improvements, some of them with pretty weird display formats! If you've done that kind of mod, and would like to run your system under CEGMON, we can get a 'special' version blown for you (the User Group's 32-by-48 display mod is a 'standard' version, by the way). As long as it only involves

changes to either the display format, the command characters and/or the keyboard look-up table, 'specials' are quite easy, and will be available for a small surcharge — contact us, or Mutek or your local dealer, for details. Outside of these relatively simple table changes, any mod becomes a re-assembly job — and since there is just one byte unused in CEGMON, that will inevitably mean losing some function or functions, and will be expensive as well. But for those interested, we will be looking into the possibility of 'losing' the disc bootstrap in exchange for a 'stringy-floppy' boot — the Exatron system now being available for OSI equipment in the States.

Finally, we hope that those of you who've already bought CEGMON are enjoying the difference that it makes to their programming; and, in our usual cheeky fashion, we urge those of you who haven't got it to go out and buy it, especially as the machine-code series we'll be starting next issue will be making full use of its monitor facilities!

Velvet Software's peripheral control unit a hardware review

This comes as a kit complete with all components. It is very well documented in that the instructions are clear, the various diagrams well drawn and legible. There is however no circuit diagram. The data lines are taken from a socket next to the ACIA and partially decoded lines taken from the prototype area using a DIL socket. One chip (a decoder) has to be wired into the Superboard. The extension board consists of copper-strip Veroboard in which the strips have to be broken in the appropriate places. Whilst not difficult this is rather tedious and I think many people would prefer to pay the extra cost of a PCB.

There was one serious drawback with the kit I received. The pin connections of the four transistors used to operate the four reed relays were shown as for the BC212L whereas in fact the transistors supplied were BC212A (although not marked as such) which had different pin connections. It took a couple of hours of debugging before it was realised what had happened. The symptoms were the four relays being switched on irrespective of what was put on the address and data lines. In addition one of the relays turned out to be defective, but one would not expect this to be a common fault. Other control lines than those needed to operate the relays are taken to the extension board so that other peripherals can be connected to it. However without a circuit diagram it is not possible to know what addresses could be utilised or what further decoding would be necessary.

I would say that the kit does represent good value for money and is a very useful addition to the Superboard. I am personally quite satisfied with my purchase.

Michael Slifkin

Velvet Software produce the controller kit in a variety of configurations, including one with a programmable sound generator — price according to configuration. Their address is 26 Colesbourne Close, Worcester WR3 9XF; phone 056 885 453.

The 'garbage collector' bug

— the problem, and two solutions

It is interesting, but infuriating, that a serious bug still exists uncorrected in the version of Microsoft's 6502 BASIC used by OSI and Comp — the 'garbage collector' bug that converts everyone's word-processor into an unusable mess of garbage. Ohio Scientific, we're told, have no real plans to fix the problem, since 'the small benefits are totally out of proportion to the cost of masking new ROMs'; but CompShop are a little more amenable at the moment, of which more later.

Like most bugs, it's small, subtle, not often encountered, but almost invariably fatal to the program concerned. It only occurs when BASIC tries to reshuffle string arrays to remove redundant ones and 'invent' a little more room — hence the term 'garbage collection', and hence its fatal effect on word processor programs. The symptoms are well known: in the middle of handling a string array the program suddenly 'hangs', and the screen seems to 'pulse' about once every one and half seconds. Sometimes, but only sometimes, the system recovers — and then only after a long wait; but even then the contents of the string arrays will usually have been scrambled into garbage themselves. The same thing happens if the 'how much free memory' function — $Y=FRE(0)$ — is called.

This only happens with string arrays such as A(1)$, A(2)$, not with simple strings like $A1$$ and $A2$$; and it tends mostly to happen when string arrays are concatenated — such as by using A(X)=A$(X)+B$$ to build up a string, since these operations use up a vast amount of temporary storage space while the string is being built. To demonstrate what happens, George Chkiantz provided us with this modification of a routine originally published in Aardvark's *First Book of OSI*. Like the fast screen-clear published in Issue 2, it stores the strings in screen memory rather than in the normal program workspace; and then shows what happens as a string is built up.

First, change the string space pointers in immediate-mode (i.e. type in the POKEs without any line numbers). This can't be done within a program — it would lose all its variable and string pointers in the process!

POKE 123,0: POKE 124,209: POKE 133,0: POKE 134,212 (or POKE 134,216 for a 2K screen-memory display on a C2, C1E or the like).

Then use a program to clear the screen, build up a string array (in this case of the alphabetic characters a to z repeated for each element in the array), and halt between building each element by calling the keyboard routine, to wait until any key is pressed.

```
10 FOR I=1 TO 32:PRINT:NEXT — screen-clear (PRINT CHR$(26) on CEGMON)
20 DIM L$(20)
30 K=64
40 FOR I=1 TO 26
50 FOR J=1 TO K: L$(I)=L$(I)+CHR$(96+J)
60 REM see what happens if you insert Y=FRE(0) here!
70 NEXT J
80 POKE 11,0: POKE 12,253: X=USR(X) - wait until any key pressed to continue
90 NEXT I
```

When this program is run, the screen will fill with 'garbage' strings from the concatenation — the bottom string will be the final correct one. The program will then

wait for any key to be pressed, on which it will construct another string as the next element in the array. This will continue until the string space is full and the strings being stored meet up with the pointers at the top of the screen. If the garbage collector (or GC from now on) did its job properly, all the 'garbage' would be cleared as string space ran out, and only the real strings would remain, eventually causing an 'OM ERROR' (out of memory) when they ran into the pointers. In practice, with the original GC, the routine can only fill the area — it 'bombs out' as soon as the GC is called, either on running out of storage space, or if $FRE(0)$ is called. That is what goes wrong, and that is the problem with trying to write any kind of word-processor for the BASIC-in-ROM. Quite simply, it dies. There are a number of 'fixes' around, such as those published by Elcomp and Aardvark in their respective *First Book of Ohio Scientific* and *First Book of OSI*. Elcomp's routine does not work at all, while Aardvark's BASIC patch (when shorn of its published typing errors!) only improves the situation rather than resolving it. The only complete solution is to fix the problem at the machine-code level, of which more anon; but for the moment, here is the corrected Aardvark routine. As can be seen with the test program above, it runs a lot longer before expiring — probably good enough for many applications.

```
10 X=PEEK(133):Y=PEEK(134)
20 L=256*Y+X:L=L-262
30 Y=INT(L/256):X=L-256*Y
40 POKE 133,X:POKE 134,Y
50 POKE 11,X:POKE 12,Y
60 PRINT "POKE 11,":X;"POKE 12,":Y
70 PRINT L: A=45383:B=45644
80 K=L:FOR I=A TO B
90 IF I<>A+34 THEN 110
100 M=K+146:GOTO 240
110 IF I=A+59 THEN 130
120 M=K+141:GOTO 240
130 IF I=A+67 THEN POKE L,4: GOTO 230
140 IF I<>A+84 THEN 160
150 M=K+209:GOTO 240
160 IF I<>A+137 THEN 180
170 M=K+146:GOTO 240
180 IF I=A+216 THEN POKE L,2:GOTO 230
190 IF I=A+217 THEN POKE L,24:GOTO 230
200 IF I<>A+261 THEN 220
210 M=K+4:GOTO 240
220 X=PEEK(I):POKE L,X
230 L=L+1:NEXT:PRINT "Location":END
240 Y=INT(M/256):X=M-256*Y
250 POKE L,Y:POKE L-1,X
260 GOTO 230
```

Using an external 'patch' may work well enough in some cases, but it still isn't good enough for many others. The problem really needs to be resolved at the machine-code level, in the BASIC ROMs — in BASIC3, to be precise. Dick Stibbons, one of

our members, has been through the 'garbage collector' routine with very carefullly indeed, not just identifying where the bug is and why, but providing a complete (and also shorter!) solution as well. The following are his notes on the problem and its solution.

The complete Garbage Collector (GC) routine at \$B147 to \$B24C will move one string at each pass as follows:

Reset the \$81, 82 pointer to be equal to the \$85, 86 pointer, thus making all memory available.

Using these two pointers to define a window, search every current string pointer pointing within this area and find the one with the highest value.

Move that string to the top of memory (the only string it can over-write is itself).

Loop to find the next-highest string and repeat.

When no pointers remain within the window which is left, the routine is complete.

The routine has two distinct parts:

- 1 Find the next string to move.
- 2 Move it and update the pointers.

There are three types of string which need to be checked. These are as follows:

1 The Descriptor Stack

BASIC waits until it has used up the last byte of free memory before calling the GC routine. This means that it almost certain to be part-way through creating a new string. After garbage collection, it continues from where it left off, and finishes the string, so steps must be taken to ensure that garbage collection preserves the substrings which were being worked on and updates their pointers.

These are defined in the descriptor stack, nine bytes in page-0 from \$0068-71, divided into three groups of three, each capable of defining a substring in the form: length; address low; address high.

The number of descriptors in use (three maximum) is indicated by the descriptor stack pointer at \$0065, which contains:

- 68 0 descriptors
- 6B 1 descriptor
- 6E 2 descriptors
- 71 3 descriptors — note the intervals of three.

2 String Variables

The definitions of these start at the address pointed to by \$7B, 7C. Every variable — string or numeric — is defined in six bytes [see Issue 2 — Ed.]. If it is a string, the form is:

- 1 Name (first letter, in ASCII)
- 2 Name (second letter, in ASCII, with bit 7 set to denote 'string')
- 3 Length of the string
- 4 Pointer, low byte (low of actual address of string)
- 5 Pointer, high byte
- 6 Null

User Group Notes

Contacts

Last issue we commented that several members wanted the 'club' aspect of the Group developed, for informal sub-groups and meetings. We asked for people who were interested to send in their names and addresses; here are those who've put themselves forward so far.

David Webster, 99 Edmondstown Road, Edmondstown, Rhondda, S. Wales. Phone (work): Caerphilly 885911, Ext.30.

Dick Stibbons, 3 Mansfield Drive, Hayes, Middx. UB4 8DZ. Phone. 01-848 9926.

David Cannon, 91 Glenfield Frith Drive, Glenfield, Leicester LE3 8PU. Phone: 0533 871140.

Kevin Johns, 77 Feeches Road, Prittlewell, Southend-on-Sea, Essex SS2 6TE. Phone (work): Southend 49431 Ext.434.

Kirklees Computer Club: meets every Monday at 7.30pm in The White Swan, 14 Kirkgate, Huddersfield — about a dozen of its members have Superboards.

Eric Wilson of ACS Cleaning Services, at 1 Raglan Court, 31 Balaclava Road, Bitterne, Southampton (phone: 0703 464611), would like any experienced members in the Hampshire area who would be interested in working on a fairly large data-matching project to get in touch with him.

Member **Ian Wales** has some equipment for sale — the old type 400 series system, consisting of 400 CPU board, 440B video (with colour), two 420B 4K RAM boards, and the associated documentation. He's at Koenigsbergerstr. 10, 6107 Reinheim, Odenwald 1, West Germany; but he'll be back in the UK for a week at the end of October, and could bring them with him then.

Finally, we've all noticed the dearth of women in the computing field generally, so this note from **Veronica Leach** seemed particularly apposite:

"One thing puzzles me, there don't seem to be many women into home computing. Friends at work regard me as strange, and consider that, on the whole, the money would be better spent on a Kenwood food mixer. This seems to be an awful shame, since I realised that you don't have to be a whiz at maths in order to do programming. I have become rather computer orientated; my job for instance would be a cinch... for every bus operator who wants to take another bus on his licence I have to check four different books and refer to two card indexes. This takes ages, and moves me in about ten points of the compass each time — Bah! Even the engineers (mechanical, not electrical) seem impressed. 'Do you understand this?' seems to be the favourite question as they thumb through the mags.

"All I want now is a T-shirt with PEEK & POKE on. Husband sez he won't be seen in the streets with me if I do, 'what about RAM & ROM?'. No way!"

THE WORLD'S FINEST HOBBY COMPUTING SOFTWARE

MASTER
PACK

UTILITIES PACK for
Compukit UK101 and
Ohio Superboard
(all screen formats)

Sixteen utility programs that will revolutionise your programming techniques. All programs feature new logical screen address system (line 1 column 1 is address 101) with FULL protection against under/over poking.

* Simple and complex graphics created with single GOSUB calls * Inputs displayed at any screen address without scrolling * Strings displayed at any screen address without scrolling * Full page of strings displayed by defining just one variable * TEXTRA text display - a full screenful of text displayed direct from the keyboard * Graphics Design Toolkit - 'Graphics Underlay' and 'Screen Address Indicator' to speed your graphics design * Precision Random Number Generator - a great improvement on Microsoft's RND * Instant clear and fill screen and other invaluable routines * Modular design to minimise RAM needed (full pack 1300 bytes - 500 - 600 bytes in typical applications) * Written entirely in BASIC for easy customisation * Comprehensive operating instructions and demonstration program.

Our best-selling UK101/SUPERBOARD program pack:
NOW ONLY £14.95 including VAT

TO ORDER: Enjoy the ultimate demonstration of program quality - in your own home on your own computer, with the security of our 10-day money-back guarantee of satisfaction. Just send cheque/PO to include 50p to cover post, packing and insurance.

PREMIER software is available ONLY direct from PREMIER PUBLICATIONS
We will be pleased to send you details of our software range for your computer - phone or write today

from Premier Publications

12 Kingscote Road Addiscombe Croydon Surrey Telephone 01-656 6156
Britain's biggest hobby software specialist-over 90 000 programs sold to date!

MASTER
PACK

STRATEGY GAMES PACK
for TRS 80 (16K), Video Genie,
Compukit UK101 (8K),
Ohio Superboard
(standard screen, 8K),
and Sharp MZ-80K

Three extra-special games
GUARANTEED to appeal to enthusiasts
who want something a little more thought-provoking than Space Invaders!

* SQUARE SOLITAIRE - Solitaire brought up to date. Unique REPLAY feature gives you a slow-motion replay of all your moves, and allows you to resume play at any point, helping you to develop winning strategy. Incredible graphics!

* NINE-IN-A-LINE - The age-old game of Reverse with new and challenging variations to keep you engrossed for hours.

* EXECUTIVE JIGSAW - An entirely new game that's as frustrating as it is fascinating. Use your skill to exactly fill the jigsaw frame. Great fun (even if you don't like ordinary jigsaws).

Other leading software publishers would probably ask £8-£12 for just one of these 'Rolls-Royce' games. But PREMIER's value-for-money price is only £12.95 for all THREE, and that includes VAT.

OVERSEAS: Please deduct VAT (divide price by 1.15) and add postage for 200 grams weight OR send two International Reply Coupons for quotation/program details.

Orders normally despatched within five working days
PLEASE SPECIFY YOUR COMPUTER WHEN ORDERING

Dealer Notes

Again a collection of new dealers (new to us, at any rate!), plus a few comments from others on new items and other things they are doing.

Northern Micro, 29 Moorcroft Park Drive, New Mill, Huddersfield.
Tel: Holmfirth (0484 89) 2062.

"We are a small concern who began trading on 15th September, offering the Superboard with various modifications and add-ons, such as the 48×32 Superboard, an upgrading service for existing models, and also a kit which will include a 'fix it' service for those who experience difficulty. We are hardware based and as a result we are only offering a few programs for sale, most of which are in machine-code; these include a Space Invaders program in 3½K and a Extended Monitor which displays 120 bytes on screen, etc. We expect to sell mainly to private individuals and we therefore intend to have a technician available during the evening and at weekends to answer queries and give demonstrations, as this is when most people are wanting service."

Premier Publications, 12 Kingscote Road, Addiscombe, Croydon.
Tel: 01-656 6156.

"Premier Publications, Britain's biggest hobby software specialist, has a rapidly expanding range of high quality software for the Ohio Superboard and Compukit UK101. To continue this expansion programme, we are urgently seeking freelance part-time programmers to join us. We pay generous royalties, and Premier marketing assures you of a wide market for your programs. We are interested in hearing from programmers of ready-written software for sale, and from programmers who would prefer to write software based on ideas and program briefs provided by us. Fluency in BASIC or machine-code is assumed, but all programmers receive advice on 'house style' and standard subroutines. For details please write, or preferably telephone, to the above address and/or phone number above."

See Premier's ad elsewhere in this issue for more details about them and their current range of software.

Ing. W. Hofacker GmbH, 8 München 75, Postfach 437, West Germany.
Tel: 08024/7331.

Hofacker are the European distributors of the American group Elcomp — software, hardware add-ons, books and technical notes for OSI and also for Pet and TRS-80 — under the Elcomp and Silver Spur trade-names. Winfried Hofacker let us use part of his stand at the recent PCW Show, and had a very good range there, including useful items like an eight-way joystick (i.e. with vertical and rotary movement); we'll be reviewing Elcomp's First and Second Book of Ohio Scientific in the next issue.

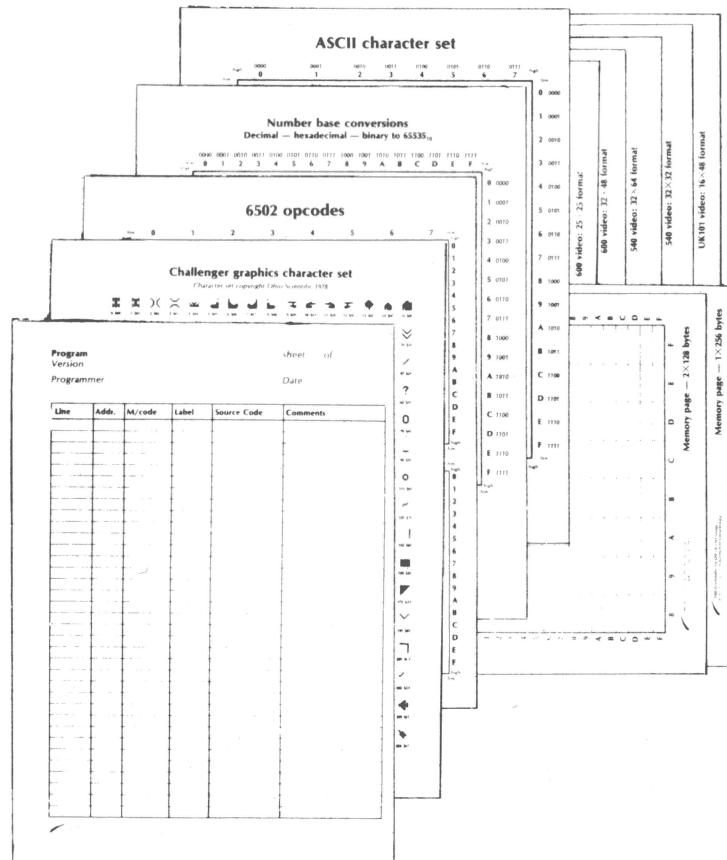
J E D Ltd, 15 Ashgrove, Springhead, Oldham, Lancs. Tel: 061 652 1604.

Our member David Hardman rang up to suggest a number of offers his firm could make for other members. They deal mostly in printers and printer mechanisms: for example a 21-column (2¼" standard paper) mechanism, without electronics, for

Planning cards

A complete range of planning and programming cards and pads for users of OSI and UK101 systems.

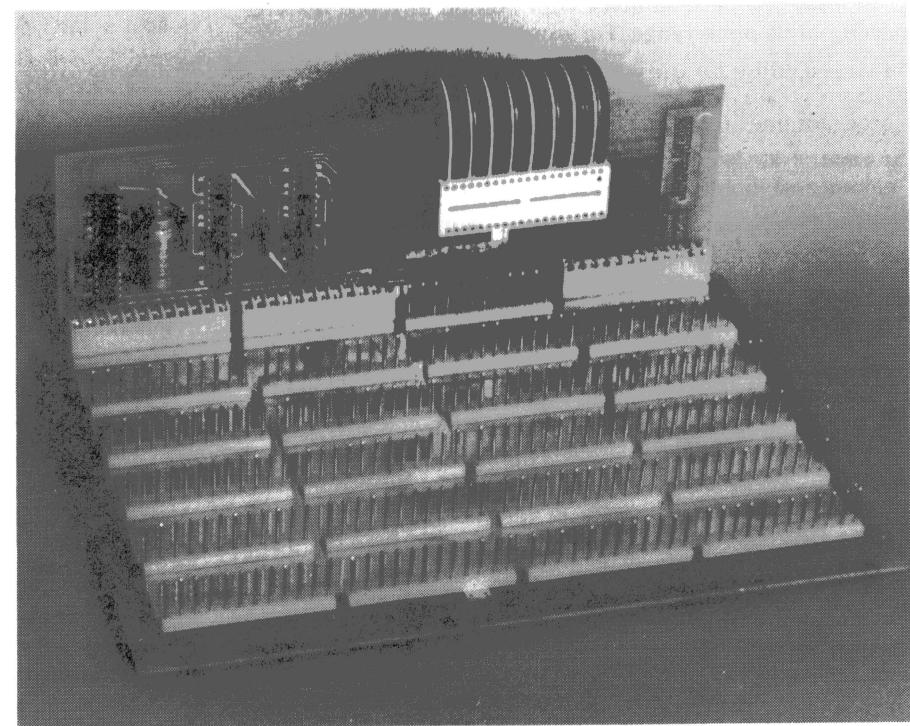
BASIC machine-code video charts opcodes graphics



Available to User Group members at the 'trade rate' of £1.50 for 100-sheet pads,
50p for laminated cards.

For further details and complete list of pads and cards, contact *Wordsmiths* at
19a West End, Street, Somerset BA16 0LQ.

£44; and an 80-column, 7-needle, sprocket-fed mechanism for about £160, depending on the number of orders. Anadex and Epson heads at considerably less than the 'official' price (new, or will recondition your existing printer head). Also metal case for Superboard/UK101, available with choice of three different heights of lift-off top, starting at £23 for the low-line version.



Zen Computer Services, 71 Manor Avenue, Sale, Cheshire M33 5JQ.
Tel: 061 962 3251.

Produces a variety of hardware items for OSI equipment, including a very useful own version of OSI's 620 board backplane unit (see photo). This is a six-slot 48-line backplane for OSI's big-system boards, with a 40-pin DIL header for the Superboard/C1 or UK101 expansion socket — allowing you to use the 470 floppy-controller for 8" floppies on a Superboard, for example. The backplane/expansion interface set is £43.70 ex-stock, built and tested. Also available are a '50Hz' conversion kit (£5) and a useful OSI-to-Acorn adaptor (£3).

CEGMON

'the best thing for OSI systems since OSI itself'

CEGMON is a new monitor PROM for all Ohio Scientific and UK101 BASIC-in-ROM systems. Written by the organisers of the OSI UK User Group, with the user's needs in mind, CEGMON gives you the kind of firmware support you need to get the most out of your system, and gives your computer more features and flexibility than anything in its price range. For example:

- a **screen editor** for use with BASIC or Assembler programs, linked directly to the system's calls for keyboard input; allows copying and alteration of text or program lines from anywhere on the screen.
- a **revised keyboard routine**, giving typewriter-like response, true ASCII key-values and direct access to most graphics.
- a completely **new screen handler** — output to the screen is via user-definable 'windows', easily programmed to allow free mixing of text and graphics, protected non-scrolling areas, and multiple scrolling and non-scrolling zones. Cursor controls and separate 'window'- and screen-clear commands are also included; text is printed from the top of the current 'window'.
- **full machine-code monitor**, co-resident with BASIC and Assembler. Includes machine-code *load* and *save* in auto-start format; *memory modify* allows input of text and graphics as well as hexadecimal instructions; *tabular display of memory contents* in hexadecimal; *memory block move/copy*; and *breakpoint handler* for debugging programs. Most subroutines are available for use in your own programs.
- **disc bootstrap** — OSI-compatible floppy-disc bootstrap available on all versions.
- **input and output** from BASIC or Assembler vectored through RAM, allowing direct linkage to user-defined I/O routines.
- **compatible design** — designed for the maximum practicable compatibility with the standard monitor, CEGMON will run your existing software with little or no alteration.

All this packed into a single replacement PROM! Five versions of CEGMON are available:

C1 — standard OSI Superboard or C1.

C1E — 32×48 display Superboard/C1, C2-type keyboard scan (Mutek conversion).

C1U — 32×48 display Superboard, C1 or UK101, standard (invert) keyboard scan.

UK101 — standard 16×48 display UK101.

C2 — OSI C2 and C4 systems (requires small hardware mod to address full 2K ROM).

Price: £29.50 (excluding VAT) includes full documentation with program examples and reference card.

Further details from UK distributor: Mutek, Quarry Hill, Box, Wilts. Tel: Bath (0225) 743289

More details from Easicomp of Norfolk and Beaver Systems of Thame:

Easicomp, 57 Parana Court, Sprowston, Norwich. Tel: 0508 46484.

Mentioned in last issue, but we've had a bundle of leaflets from them since then. They do their own 'cased Superboard', called the 'Easicomp Companion'; a variety of software and other assorted items like the Microcase; and also an interesting-looking programmable sound-generator board for Superboard/C1 and UK101.

Beaver Systems, Norlett House, Dormer Road, Thame, Oxon OX9 3UC. Tel: Thame (084 421) 5020.

Sells the full 'personal' OSI range — Superboard to C8P — and also the Mutek-type 'enhanced' C1, the C1E; but specialises in software, partly bought-in (from OSI, Aardvark and others), partly their own publication (they will publish any good software for OSI/UK101 kit), and partly their own productions — including the most amazing version of *Life* that I've seen yet, and a very neat set of utilities. Steve Hanlan very kindly allowed us to use part of his stand at the PCW Show to demonstrate our new monitor — many thanks.

The arrival of CEGMON

We're pleased to say that our new Monitor EPROM for OSI BASIC-in-ROM systems and the UK101 is now ready. (The name CEGMON is based on our initials, but don't let that put you off!). We've included as many of your suggestions and requests as we could, but we couldn't put *all* of them into a 2K ROM, of course! That's why you won't find a 'named file' cassette handler on board, for example — to do the job properly would have taken an inordinate amount of space — and neither is there a 'data save and load' system because, once again, it takes a lot of space to do it (and in fact the best way to handle data files is in BASIC itself, where it only takes a couple of lines!). What you *will* find, however, are a range of useful functions which make your machine far more flexible than before, yet without losing compatibility with existing software. You'll find full details in the ads in *Practical Computing* and elsewhere; but briefly we've included an improved version of the *Sirius Cybernetics* screen editor; a revised keyboard routine that ends the ludicrous juggling-act with the Shift-keys for lower-case, with proper decode for Return, Line-feed, Escape and Rubout (which now does what it says on OSI systems!), and using the Repeat key (on OSI systems only) as a second control to access graphics; a new fully-programmable screen-handler like that on the new Super-PET, with screen-clear, 'window'-clear and cursor controls; expansion of the machine-code monitor to include not just machine-code *save* (also compatible with the Assembler - at last!), but virtually all of the OSI Extended Monitor (bar disassembler or search) as well. We've not only retained the disc bootstrap for the Superboard/C1 and UK101, but added one for the C2/C4 series, which also now have the former systems' user-definable input-output vectors. We had to use some of the old 'free RAM' in page-2, from \$0222-0234, for the editor's and screen-handler's stores and tables; but these can be disabled with a single POKE, to allow existing machine-code routines starting at \$0222 to run. We've been very careful about compatibility: almost all of the former routines' start points have been retained, and the editor and Monitor both work happily with either BASIC or Assembler.

At about £34 after the dreaded 15%, CEGMON isn't particularly cheap (and no, there ain't no User Group discount!); but there's an awful lot in there! Not just the firmware, but a decent bit of documentation for it as well — 20 pages plus a reference card. Main distributors are Mutek, at Quarry Hill, Box, Wilts; other dealers such as Premier Publications and Beaver Systems should also have it by the time you get this Newsletter.

CEGMON and WP-6502 word-processor

CEGMON is, as far as we know, happily compatible with BASIC, the Assembler and ExMon. The other major software package which a lot of people now have is the WP-6502 word-processor from 'those Chinese guys', Dwo Quong Fok Lok Sow — which will run under CEGMON, but only with the new screen-handler and editor disabled. Its main jump-table is stored at \$0222 onwards, so that hitting BREAK will scramble it, and its contents will have to be restored each time. Steve Hanlan of Beaver Systems has been discussing this with DQFLS, and they are willing to produce a new version of WP-6502 which will not only resolve the BREAK problem but should use some of CEGMON's editing and screen facilities as well as the improved keyboard. This will involve a re-assembly of the source code, but Steve reckons it should be here in time for Christmas.

3 Arrays

The definitions start at the address pointed to by \$7D, 7E. Each string array definition has the form:

- 1 Name (first letter)
- 2 Name (second letter, with bit 7 set)
- 3 and 4 Length of pointer block, including definition section
- 5 Number of dimensions
- 6 Null
- 7 Size of last dimension
- 8 Null
- 9 Size of penultimate dimension
- 10 Null

— and so on until the first dimension of the array is reached.

The distance between the 'initial letter' byte and the start of the element pointers themselves is given by $(N^2) + 5$, where N is the number of dimensions (see \$B1B7, B1B8).

The element pointers have three bytes each, again in the form: length; address low, address high. Importantly, there is a null between each element pointer.

So, back to the garbage collector itself. Part 1 is built round a subroutine, \$B1D1-B215, which examines any one string and decides whether it qualifies for a move. The tests are:

- \$B1D4 Is this variable a string? (Variables only).
- \$B1D9 Is its length >0?
- \$B1E4 Is it in the area currently defined as free memory?
- \$B1EE Is its pointer higher than any one checked so far?

If all the answers are 'yes', the 'core' of the subroutine is reached (\$B1F6-B205) and the string position is recorded in \$AA, AB; its pointer position in \$9C, 9D; and a number to indicate its type in \$A2.

This subroutine is called from three areas in the program, each devoted to a particular type of string. In each case, a working pointer (\$71, 72) is set up for it, and the increment between pointers is stored in \$A0.

The subroutine is called from within a loop and itself increments the pointer, ready for the next call. It is the duty of the calling routine to check when all strings, of the type in which it deals, have been examined. String types are checked in the order Descriptors, Variables, Arrays, and the increment for each case is 3, 6, 4.

The principal bug in the existing routine is that the increment given for arrays is not 4, but 3 (\$B188). The search thus attempts to treat spurious bytes as pointers, and crashes. [The 'glitch' on the screen every one-and-a-half seconds is this confused search running through the screen memory each time round its loop — Ed.] Changing the increment to four does solve this problem, but creates two more in the process!

When all the strings have been checked, the routine then jumps to Part 2, \$B197, the 'move' routine. The first tests to see if a string qualified for a move during the search. If not (i.e. \$9D=0), garbage collection is complete, and it branches back to pick up the RTS at \$B215.

There then follows some arithmetic to calculate the distance between the pointer pointer (\$9C, 9D), and the 'Length' byte of the string definition. The figures (which

arise purely because of the way the search routine is set up) need to be:

Descriptors 0

Variables 2

Arrays 0

and will be stored in \$A2. The original system noted the increment value in the subroutine 'core' (\$A0-A2) and used this to work out the new increment — fine if there are only two possible primary increments, 6 and 3; but there are now three: 6, 3 and 4. Easy if you've got bytes to spare — but we haven't.

On the other hand, the set-up implies some redundancy. If we found the string to know it needs moving, how come we need more arithmetic to work out how to move it? Looking back into the core of the routine, we find that Y will be 2 for descriptors and arrays, and 4 for variables. Decrement it twice and store it in \$A2 and we have the new increments for later use in one easy move — and no extra bytes!

In fact, we can now scrap all of \$B21C-B223 and replace it with just one 'clear carry'. Is that it? Well, nearly.

At the entry to the whole routine (\$B147), \$A0 is expected to be 3 (set by the 'Cold Start' procedure at \$BD62) and the old garbage collector leaves it at 3. We've now changed it to 4, so instead of crashing at the array search, we crash at the descriptor search instead! However, we now have bytes to play with, so a DEC \$A0 before we leave the routine (\$B216) leaves everything perfect — honest!

A final tidy-up is to ditch a bit of assembler inefficiency at \$B15D (Y is already 0 and X doesn't matter), and shuffle everything up. We now have a garbage collector that not only works, but uses 15 cycles — and 5 bytes — fewer.

Addresses used

7B, 7C Start of variables

7D, 7E Start of arrays

7F, 80 Start of free memory

81, 82 Start of strings

85, 86 End of memory

71, 72 Working pointer to find the various string definitions

9C, 9D Pointer to the old pointer. 9D also serves as a flag: a search completed without any qualifying strings being found leaves 0 in \$9D.

A0 Contents are the increment between the pointer to one string pointer and the next. All the bugs live here!

For descriptor block, =3 (or should)

For string variables, =6 (and does)

For string arrays, =4 (not 3!)

A2 Used to determine which type of string is about to be moved (\$B21C). The final contents are the distance between the 71, 72 pointer and the 'string length' byte in the string definition.

For descriptor block, =0

For string variables, =2

For string arrays, =0

AA, AB Used to record the 'highest pointer so far'. Left pointing to the string to be moved and used as such by the \$A1D6 'move' routine.

A4, A5 a) Used to point to the end of the block under examination (variables and arrays only);

b) Used for the move routine to point at the byte at which the newly-moved string will end (\$A1D6 works backwards).

A6, A7 Used by the move routine to point to the end of the string in its old position.

65 Pointer to the string-building descriptor block of the highest order in use at the time the GC routine was called.

68, 69, 6A; 6B, 6C, 6D; 6E, 6F, 70 3-level descriptor stack used in building strings. Each triplet may contain a string definition, in the form: length, low address, high address. All three need to be included in the garbage collection in case they define substrings which were being assembled at the instant the GC routine was called. A1D6 subroutine Moves a block of memory, \$AA, AB - A6, A7 to a new area, ending at \$A4, A5. Returns with X=0 and Y=0 and \$A4, A5 pointing to the byte which would have been used next (but note that \$A5 is one less than its true value).

The decision to call the GC routine is made in the 'string space check' subroutine at \$A21F. The 'OM ERROR' caller is at the end (\$A24C) and this runs on into the Warm-Start location, \$A274. \$A21F is called from:

A1CF 'Record string' routine (latter part, from \$A1D6 on, is used by GC itself).

AE8D 'Build array' routine (called twice, at \$AEA4 and \$AEF3).

BD11 'Cold-Start' calls \$A21F for 'Memory Size' check (\$BE09).

The GC is also called direct by the STR\$ function at \$B13D and the FRE function at \$AFAD.

Also interesting (but irrelevant) is the fact that the 'string temporaries' handling at \$B0AE has a 'number of orders' test at \$BE0F, and the 'ST ERROR' caller is at \$B0F3.

Descriptors

B147	A685	LDX \$85	; initial entry
B149	A586	LDA \$86	; re-entry for further passes (update string-space ptr)
B14B	8681	STX \$81	
B14D	8582	STA \$82	
B14F	A000	LDY □000	; becomes a flag for 'GC complete' (see B21A)
B151	849D	STY \$9D	
B153	A57F	LDA \$7F	
B155	A680	LDX \$80	
B157	85AA	STA \$AA	; becomes 'bottom of string space' pointer
B159	86AB	STX \$AB	
B15B	A968	LDA □\$68	; to search descriptor block
B15D	8571	STA \$71	; (minor change — see notes above)
B15F	8472	STY \$72	
B161	C565	CMP \$65	; descriptor stack pointer
B163	F005	BEQ \$B16A	
B165	20D7B1	JSR \$B1D7	; 'test this string' subroutine
B168	F0F7	BEQ \$B161	; always branch

Variables

B16A	A906	LDA □\$06	; search increment
B16C	85A0	STA \$A0	
B16E	A57B	LDA \$7B	
B170	A67C	LDX \$7C	
B172	8571	STA \$71	
B174	8672	STX \$72	

B176	E47E	CPX \$7E	; 'have all variables been searched?' (If yes, go to B183)
B178	D004	BNE \$B17E	
B17A	C57D	CMP \$7D	
B17C	F005	BEQ \$B183	
B17E	20D1B1	JSR \$B1D1	
B181	F0F3	BEQ \$B176	

B183	85A4	STA \$A4	
B185	86A5	STX \$A5	; (71,72) will be left pointing to
B187	A904	LDA □\$04	; first array block — store here
B189	85A0	STA \$A0	; search increment — change, see notes above
B18B	A5A4	LDA \$A4	
B18D	A6A5	LDX \$A5	; = start of arrays
B18F	E480	CPX \$80	
B191	D007	BNE \$B19A	; any (more) arrays?
B193	C57F	CMP \$7F	
B195	D003	BNE \$B19A	
B197	4C16B2	JMP \$B216	

B19A	8571	STA \$71	
B19C	8672	STX \$72	
B19E	A001	LDY □\$01	
B1A0	B171	LDA (71),Y	; = '\$ or numeric' byte
B1A2	08	PHP	; i.e. push value of top bit, for '\$ or numeric'
B1A3	C8	INY	
B1A4	B171	LDA (71),Y	; = 'length of array block', low byte
B1A6	65A4	ADC \$A4	
B1A8	85A4	STA \$A4	
B1AA	C8	INY	
B1AB	B171	LDA (71),Y	; ditto, high byte
B1AD	65A5	ADC \$A5	; (A4,A5) now pointing to start of next array block
B1AF	85A5	STA \$A5	
B1B1	28	PLP	
B1B2	10D7	BPL \$B18B	; if plus, this array is numeric — branch to look at next
B1B4	C8	INY	
B1B5	B171	LDA (71),Y	; = 'no. of dimensions' byte
B1B7	0A	ASL A	; ×2 + 5 gives position of first array element
B1B8	6905	ADC □\$05	
B1BA	6571	ADC \$71	
B1BC	8571	STA \$71	
B1BE	9002	BCC \$B1C2	
B1C0	E672	INC \$72	
B1C2	A672	LDX \$72	
B1C4	E4A5	CPX \$A5	
B1C6	D004	BNE \$B1CC	
B1C8	C5A4	CMP \$A4	
B1CA	F0C3	BEQ \$B18F	
B1CC	20D7B1	JSR \$B1D7	
B1CF	F0F3	BEQ \$B1C4	

B1D1	C8	INY	
B1D2	B171	LDA (71),Y	; entry for variables
B1D4	1030	BPL \$B206	; is it a string variable?
B1D6	C8	INY	; branch if not
B1D7	B171	LDA (71),Y	; (entry for others) is LEN\$ = 0?

Arrays

; earlier entry checks 'is this a \$ variable?'
; always branch

; exit point for Part 1 — highest \$ will have been found
; (all have been checked by here)
; set pointer for array search

; = '\$ or numeric' byte
; i.e. push value of top bit, for '\$ or numeric'

; = 'length of array block', low byte

; ditto, high byte
; (A4,A5) now pointing to start of next array block

; if plus, this array is numeric — branch to look at next
; = 'no. of dimensions' byte
; ×2 + 5 gives position of first array element

; update search pointer to point to it
; finished this array pointer block?

; 'test this string' subroutine
; always branch

'Test this string' subroutine

; entry for variables
; is it a string variable?
; branch if not

; (entry for others) is LEN\$ = 0?

B1D9	F02B	BEQ \$B206	; branch if yes
B1DB	INY		
B1DB	C8	INY	
B1DC	B171	LDA (71),Y	; set X, A as pointer to pointer
B1DE	AA	TAX	
B1DF	C8	INY	

B1E0	B171	LDA (71),Y	
B1E2	C582	CMP \$82	; is this \$ in 'free memory'?
B1E4	9006	BCC \$B1EC	; branch out if not

B1E6	D01E	BNE \$B206	
B1E8	E481	CPX \$81	
B1EA	B01A	BCS \$B206	
B1EC	C5AB	CMP \$AB	
B1EE	9016	BCC \$B206	
B1FO	D004	BNE \$B1F6	
B1F2	E4AA	CPX \$AA	
B1F4	9010	BCC \$B206	

Core of string-test subroutine

; record string position in \$AA, AB

; and pointer position in \$9C, 9D

; set up \$A2 for 'move' routine — see notes

; regardless of whether this one 'won' or not,
; increment search pointer ready for the next test

B1F6	86AA	STX \$AA	
B1F8	85AB	STA \$AB	
B1FA	A571	LDA \$71	
B1FC	A672	LDX \$72	
B1FE	859C	STA \$9C	
B200	869D	STX \$9D	
B202	88	DEY	
B203	88	DEY	
B204	84A2	STY \$A2	
B206	A5A0	LDA \$A0	
B208	18	CLC	
B209	6571	ADC \$71	
B20B	8571	STA \$71	
B20D	9002	BCC \$B211	
B20F	E672	INC \$72	
B211	A672	LDX \$72	
B213	A000	LDY □\$00	
B215	60	RTS	

Part 2

; ready for next garbage collection — see notes
; if \$9D still 0 (see \$B151) no string has been found
; (i.e. garbage collection is finished)

; set up Y — see notes

; get LEN\$

; (AA, AB) = start of \$; +LEN\$ = end of \$;
; store this in (A6, A7)

; copy 'next free memory location' into (A4, A5)

B216	C6A0	DEC \$A0	
B218	A69D	LDX \$9D	
B21A	F0F5	BEQ \$B211	
B21C	A4A2	LDY \$A2	
B21E	18	CLC	
B21F	B19C	LDA (9C),Y	
B221	65AA	ADC \$AA	
B223	85A6	STA \$A6	
B225	A5AB	LDA \$AB	
B227	6900	ADC □\$00	
B229	85A7	STA \$A7	
B22B	A581	LDA \$81	
B22D	A682	LDX \$82	
B22F	85A4	STA \$A4	
B231	86A5	STX \$A5	
B233	20D6A1	JSR \$A1D6	
B236	A4A2	LDY \$A2	

; block-moves (AA, AB)-(A6, A7) to new position
; ending at (A4, A5) — updates latter

B238	C8	INY
B239	A5A4	LDA \$A4
B23B	919C	STA (9C),Y
B23D	AA	TAX
B23E	E6A5	INC \$A5
B240	A5A5	LDA \$A5
B242	C8	INY
B243	919C	STA (9C),Y
B245	4C4BB1	JMP \$B14B

; change '\$ pointer low' to new value
; (increment because of an oddity in \$A1D6 routine)
; change '\$ pointer high' to new value
; and back up to the top for another pass

Changes to the original:

Essential

B187	A904	LDA □\$04
B189	85A0	STA \$A0
B204	88	DEY
B205	88	DEY
B206	84A2	STY \$A2
B218	C6A0	DEC \$A0
B21A	A69D	LDX \$9D
B21C	F0F5	BEQ \$B213
B21D	EA	NOP
B21E	EA	NOP
B21F	EA	NOP
B220	18	CLC

Non-essential

B15F	8472	STY \$72
B161	EA	NOP
B162	EA	NOP

Shuffle as required, to lose the five NOPs, then:

B165	20D7B1	JSR \$B1D7
B17E	20D1B1	JSR \$B1D1
B197	4C16B2	JMP \$B216
B1CC	20D7B1	JSR \$B1D7

Disc System Notes

What with the 'specials' in this issue, we've run out of room to include the disc system notes. We've received two different single-drive copy routines for 65D, a fairly complete memory map of 65U, and a vast but disorganised and largely unreadable mass of disc notes from Aardvark's *Aardvark Journal* and the semi-official newsletter *Peek-65*. To make up for its absence this issue, we will be having a large Disc System Notes next issue — promise!

In the meantime, if you have any queries on 65D or 65U, or have anything to tell us (please!), please get in touch with us, and we'll do what we can!

Inexpensive hard-copy

For a relatively small sum (£15-£25) it is possible to purchase ex-GPO teleprinters which when interfaced to your computer provide a good quality output albeit with some limitations in speed and the character set. This article briefly describes the teleprinter operation and details the software and hardware necessary to interface specifically to a CREED 7B teleprinter and the Challenger C2-4P; however information is included for users of C1/Superboard and UK101 machines. Any Baudot based printer may be used but there may be small variations with the figures-mode character set which will have to be allowed for. When you obtain a teleprinter it is worth determining if possible the machine's operating code. Most use the 5-bit Baudot code, some have been configured to special codes and if you are really lucky you may find one ASCII coded.

Teleprinter operation

The Baudot code, being 5 bits long (plus 1 start bit, plus 1½ stop bits) only allows 32 (2^5) combinations of bits with which to convey information. However, in order to increase the effective character set, two of these combinations are used to define the mode in which the teleprinter operates, namely 'letters' or 'figures', which equate to lower and upper case on a typewriter. This effectively increases the character set to 60; however, some functions such as line-feed, carriage-return etc. are duplicated in both modes so the resulting number of characters reduces to 56. Table 1 lists the allocation of the 64 codes. When a teleprinter receives a mode command, it shifts to that mode and remains mechanically latched in that mode until it receives the alternative mode command.

Software

The software as listed is located between \$0240₁₆ and \$02E5 inclusive and uses location \$02FE to temporarily store the contents of the 'X' register and \$02FF to memorise the teleprinter mode. This area of RAM is not used by BASIC and is not overwritten when the machine is reset/cold-started. The program however may be located anywhere in RAM and has been written using relative branches so that the opcodes may be copied directly (with one exception) into any other area of RAM or even placed in ROM, but \$02FE-02FF will still be used as scratchpad locations. The exception is at \$0250 in the source code: LDA \$0287,X will need to be changed to LDA equivalent ,X as necessary. If it is more convenient to the user, \$02FE-02FF may be redefined by changing the lines in the source listing that are marked with an asterisk.

The program (\$0240-02A7) is written as a subroutine and expects to find an ASCII character code in the accumulator (A register). \$0240 simply calls the CRT driver routine in BASIC and displays the character on the VDU. The remainder of the program undertakes the ASCII to Baudot conversion by checking code limits and outputting a space if outside these limits, looking up the code to be output (from table \$02A8-02E5), checking for and outputting if appropriate the teleprinter mode and finally outputting the Baudot mode.

The look-up table needs some explanation. The program takes in the character code in the Accumulator and eventually places it in the 'X' register. This value, which is restricted by earlier checks to the range 21₁₆-5F₁₆, determines which location in the table relative to the base address \$0287 is to be loaded into the

Baudot value in Hex	Letters Mode		Figures Mode		Alternative Figures Mode	
	Character	ASCII	Character	ASCII	Character	ASCII
00	NUL	00	NUL	00	NUL	00
01	E	45	3	33	3	33
02	LF	0A	LF	0A	LF	0A
03	A	41	-	2D	-	2D
04	SPACE	20	SPACE	20	SPACE	20
05	S	53	'	27	"	22
06	I	49	8	38	8	38
07	U	55	7	37	7	37
08	CR	0D	CR	0D	CR	0D
09	D	44	WHO ARE YOU? (N/A)		\$	24
0A	R	52	4	34	4	34
0B	J	4A	BELL	07	*	2A
0C	N	4E	,	2C	,	2C
0D	F	46	%	25	;	3B
0E	C	43	:	3A	:	3A
0F	K	4B	(28	(28
10	T	54	5	35	5	35
11	Z	5A	+	2B	+	2B
12	L	4C)	29)	29
13	W	57	2	32	2	32
14	H	48	£	23	£	23
15	Y	59	6	36	6	36
16	P	50	0	30	0	30
17	Q	51	1	31	1	31
18	O	4F	9	39	9	39
19	B	42	?	3F	>	3E
1A	G	47	a	40	<	3C
1B	(Figures)		(Figures)		(Figures)	
1C	M	4D	.	2E	.	2E
1D	X	58	/	2F	/	2F
1E	V	56	=	3D	=	3D
1F	(Letters)		(Letters)		(Letters)	

Notes:

- Character Baudot value location in look-up table determined as ASCII value - 20_{16} .
- Value in look-up table is Baudot value plus 00 ('letters-mode' characters) or plus 80_{16} ('figures-mode' characters).
- Remaining positions in look-up table filled with 04_{16} (Baudot SPACE).

Accumulator. The values given in the table are such that bits 0-4 give the Baudot code to be ultimately output and bit 7 indicates the mode in which the teleprinter must be in order to receive the code. Bit 7 set to '1' indicates figures mode, and bit 7 set to '0' indicates letters mode.

The look-up table is listed for use with a CREED 7B fitted with a normal character set, therefore when any characters occur within the range 21_{16} - $5F_{16}$ which do not normally appear in the CREED set, a space is output. Table 1 however also lists my proposals for an alternative figures-mode set. This is based on my opinion of the most wanted characters for BASIC or machine code working and utilises codes not normally associated with characters. More on this later.

The program as listed is for the C2-4P under the standard OSI monitor; two changes may have to be made to run on different machines and/or under different monitors. For all CT1, Superboard and UK101 systems the serial output subroutine is at \$FCB1, so the calls at \$0273, \$0288, \$028F and \$02A0 should read JSR \$FCB1. Under Comp's New Monitor for the UK101 the CRT call at \$0240 should be JSR \$FA57; under all versions of CEGMON the CRT call may be unchanged or (for the new screen handler) changed to JSR \$F836.

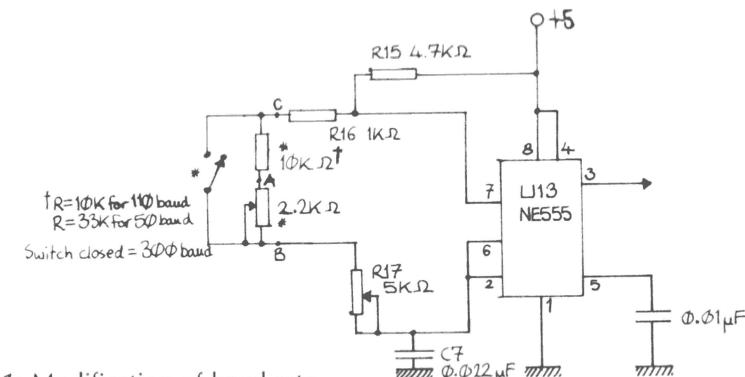


Figure 1: Modification of baud rate.
300 baud with switch closed; 50/110 baud (dependent on resistor value) with switch open.

Hardware

Teleprinters such as the CREED 7B operate at 50 baud whilst the computer normally outputs at 300 baud. The C2/C4 serial output can be easily modified to output at 50 baud by the addition of two resistors and a switch to select 300 or 50 baud operation as shown in Figure 1 — the additional components are asterisked. To effect the modification, locate R16 on the circuit board (see computer manual) and unsolder one end. Lift this end clear of the circuit board, and solder two wires, one to the free end and one to the pad on the circuit board from where R16 has been disconnected. Space is limited, so ensure there are no short circuits. The two wires are now routed to the back panel where a switch is mounted with the two additional resistors on its

```

0240 2020BF JSR $BF20
0243 48 PHA
* 0244 80FE02 STX $02FE
0247 C921 CMP #$21
0249 902E BCC $0279
024B C960 CMP #$60
024D B049 BCS $0298
024F AA TAX
0250 B08702 LDA $0287,X
0253 AA TAX
0254 2980 AND #$80
0256 D00F BNE $0267
* 0258 ADFF02 LDA $02FF
025B F000 BEQ $029A
025D A900 LDA #$00
* 025F 80FF02 STA $02FF
0262 A9FF LDA #$FF
0264 18 CLC
0265 900C BCC $0273
0267 ADFF02 LDA $02FF
026A D02E BNE $029A
026C A9FF LDA #$FF
026E 80FF02 STA $02FF
0271 A9FB LDA #$FB
0273 2015BF JSR $BF15
0276 18 CLC
0277 9021 BCC $029A
0279 C90A CMP #$0A
027B D005 BNE $0292
027D A202 LDW #$02
027F 18 CLC
0280 9018 BCC $029A
0282 C900 CMP #$00
0284 D012 BNE $0298
0286 A9E8 LDA #$E8
0288 2015BF JSR $BF15
028B A9E0 LDA #$E0
028D A203 LDW #$03
028F 2015BF JSR $EF15
0292 CA DEX
0293 D0FA BNE $029F
0295 18 CLC
0296 9002 BCC $029A
0298 A2E4 LDW #$E4
029A 8A TXA
029B 291F AND #$1F
029D 18 CLC
029E 69E0 ADC #$E0
02A0 2015BF JSR $BF15
* 02A2 80FE02 LDW $02FE
02A6 68 PLA
02A7 60 RTS

```

b) Conversion look-up table.

	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
02A8	04	85	94	04	8D	04	85	8F	92	04	91	8C	82	90	90	96
02B8	97	93	81	8A	90	95	87	86	98	8E	04	04	9E	04	99	9A
02C8	03	19	0E	09	01	0D	1A	14	06	0B	0F	12	1C	0C	18	16
02D8	17	0A	05	10	07	1E	13	1D	15	11	04	04	04	04	04	04

ASCII-to-Baudot routine

a) Assembly listing.

back. The setting of the 2.2K variable resistor is not critical so initially it may be set at its midpoint and adjusted as necessary. It should be noted that this modification also changes the cassette output frequency, thus the switch is required to allow the cassette to operate at 300 baud.

To output the data I implemented the transistor RS-232 interface by installing U31, Q2, R55, R56 and R57 (see Figure 2) and breaking the printed circuit track from pin 2 of U41, otherwise the output of U41 would be in contention with the output from Q2. It may be necessary to generate a -9V supply to R57, but this depends on the teleprinter driver circuitry. With the circuit given in Figure 3, a -9V supply is unnecessary (but acceptable if already fitted), so I connected the -9V rail to earth. Two points worth stating about Figure 3 are firstly the circuit expects the input current to be limited by the source, which in this instance is achieved by R56 in Figure 2; and secondly the opto-isolator on the input may appear to be a bit exotic, but the back-EMF from the teleprinter coil is several hundreds of volts so it is wise to ensure this cannot reach the computer.

C1/Superboard and UK101 users can readily implement their equivalent circuits to Figure 2, however, the cassette clocks for these computers are derived from the video divider chain. For UK101 users I suggest the use of a 7492 (divide-by-2 and divide-by-6) integrated circuit as indicated in Figure 4; however, not having any detailed knowledge of C1/Superboards I cannot make any suggestions, but it is possible that Figure 4 also applies.

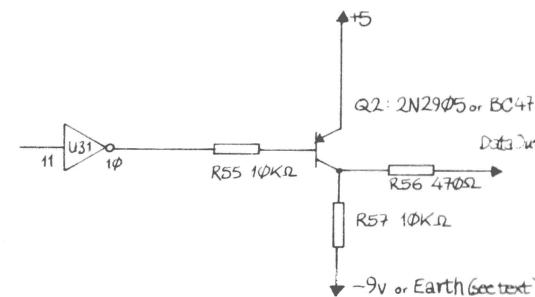


Figure 2: RS-232 interface in computer.

Using the software routine

At this point C1/Superboard and UK101 users (and also C2 users of CEGMON) have a distinct advantage over standard C2/C4 owners in that their input/output machine-code routines are accessed via vectors in page 2 of RAM. Thus to utilise the routine as written for BASIC and machine code working, their users can switch the computer's output from the CRT driver to the conversion routine by a POKE 538,64: POKE 539,2 — either within a program or in the immediate mode (in which the

POKEs must be in one line). All output to the VDU will now be echoed on the printer. For users of standard C2s all is not lost. The routine can be readily used with the Extended Monitor by using the built in monitor to change the contents of locations \$0862-0863 in ExMon from EE, FF₁₆ to 40₁₆, 02₁₆ respectively. To use the routine with BASIC however, is somewhat more complicated. I had originally planned to program an EPROM with a revised monitor incorporating the Baudot routine, however I think the most prudent solution now would be to invest in the User Group's new monitor ROM (CEGMON) which does vector BASIC's support routines through RAM. In all instances the routine can be initialised by outputting two characters utilising opposite modes in the teleprinter — this is necessary to synchronise the mode of the teleprinter with the program.

I will conclude this part of the article with three brief specific points on the teleprinter. Firstly the alternative characters suggested in *Table 1* are not available, but if you can obtain a spare character set, standard characters can be adapted with a bit of ingenuity, i.e. '9' changed to ';', 'K' adapted to '<', 'Z' to '>', 'S' to '\$', and 'X' to '×', although it is possible to obtain a proper '*'. Two of my proposed characters use the 'BELL' and 'Answer back' codes — the mechanisms on the printer associated with these codes will need to be disabled and the new character heads inserted into the corresponding empty locations in the print head. The second point is to try to obtain a printed fitted with a synchronous motor — this minimises timing problems. Finally I would recommend to anyone investing in a teleprinter to sight a copy of the Radio Society of Great Britain publication '*Teleprinter Handbook*'. This describes in great detail the workings of several teleprinter models and also the setting-up tolerances.

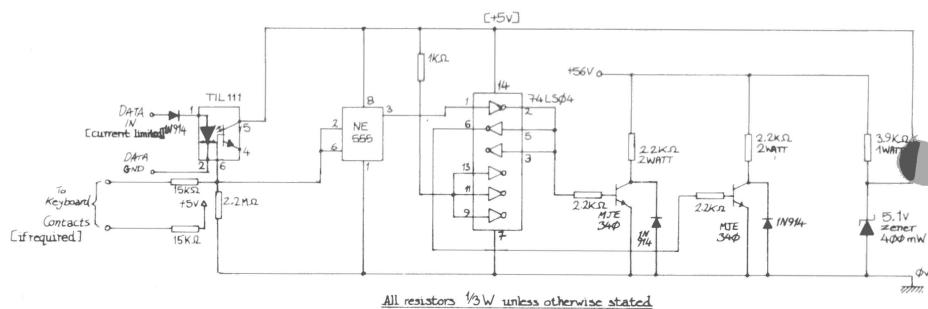


Figure 3: Teleprinter drive circuit.

- Figure 3. Teleprinter drive circuit.

 1. Keyboard contacts can be wired in as shown if required.
 2. Configure teleprinter electromagnet connections so that it is inactive when computer data output inert.
 3. Input current must be limited by source, e.g. R56 in Fig.2.

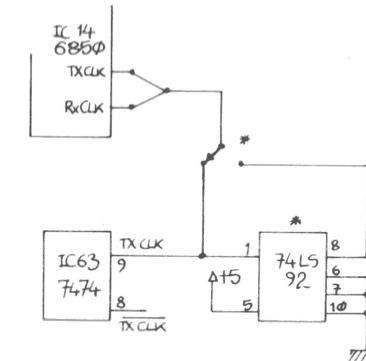


Figure 4: Suggested modification of baud rate for UK101 and possibly C1/Super-boards.

* Components additional

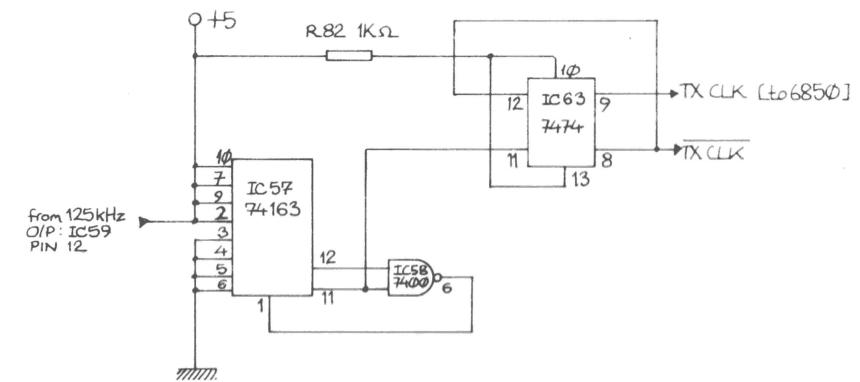


Figure 5: Baud rate generator as fitted in UK101

Component numbers relate to UK101; C1/Superboard users should be able to read across.

Suggestions for 110 baud Teletype operation (send only)

Hardware modification

C2/C4 computers: Baud rate modification can easily be implemented on these machines as described earlier, with the addition of the components marked with an asterisk in *Figure 1*. The closing/opening of the switch will select 300/110 baud operation respectively. The 2.2K linear preset should initially be set at a value of 1.77K using an ohmmeter or by eye approximately $\frac{3}{4}$ travel from end 'A' in *Figure 1*. It can then be adjusted if teletype operation is found unreliable. The baud range achievable is approximately 107 to 120 with this value of preset.

UK101 and C1/Superboard computers: On these machines the baud rate is derived from the video divider chain culminating with the circuit shown in *Figure 5*. The component numbers relate to the UK101, but hopefully C1/Superboard users can read across. Users of these machines now have two choices open to them, either to build a tidied-up version of *Figure 1* and implement it as shown in *Figure 6*, or implement the circuit shown in *Figure 7*. *Figure 6* — being derived ultimately from a crystal oscillator — needs no setting up, but will run slightly slow at 108.5 baud and will involve cutting a number of tracks on the circuit board.

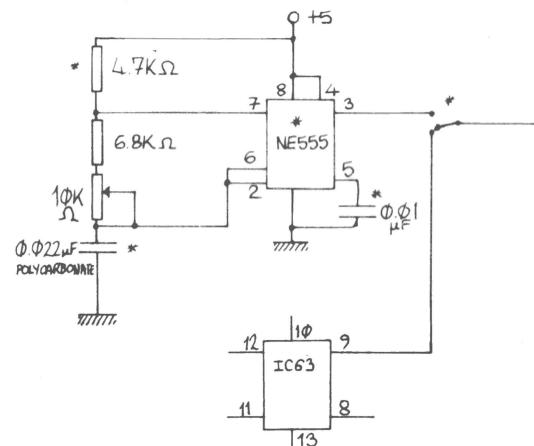


Figure 6: Proposal for 110 baud on UK101.

1. * Components additional.
2. 10K-ohm preset should be set at approximately mid-travel and then adjusted as necessary.
3. Similar modifications apply to C1/Superboards.
4. Remaining connections on IC63 unchanged.

Word format

Most teletypes operate with a word formatted with 1 start bit, 8 bits of information and 2 stop bits. All C1/Superboard, C2/C4 and UK101 systems set their ACIA to this condition on start-up (C2/C4 systems, and also UK101s under Comp's New Monitor, enable receive and transmit of interrupts; standard C1/Superboard and UK101s don't). If your teletype does not use this format, refer to Leventhal, 6502 Assembly Language Programming (Osborne/McGraw-Hill), p.11-111, or to the 6850 data-sheet, for details of how to set up the ACIA accordingly.

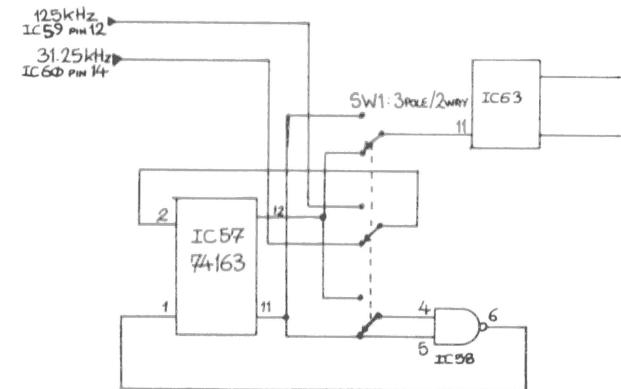


Figure 7: Alternative mod for 110 baud on UK101.

1. * Component, 3-pole 2-way switch additional.
2. Should apply to C1/Superboard.
3. Remaining connections on IC57, IC63, IC58 stay unchanged.

For both 20ma and RS-232 interfaces a -9V supply will be required.

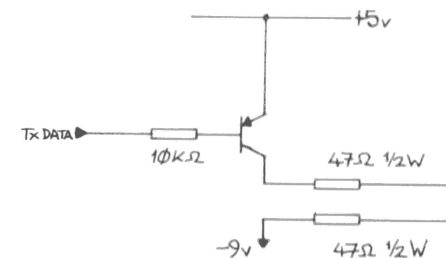


Figure 8: 20ma interface.

1. Install instead of RS-232 transistor.
2. Check current through resistors.

Interfaces

Teletypes are available fitted with one of three possible line units:

RS-232

20ma

80-0-80 volt Teleprinter Interface

RS-232 TxData can be implemented in accordance with the user manuals. 80-0-80 can be interfaced using the circuits of *Figures 2 and 3* of the earlier part of this article; however, it may be necessary to increase the supply voltage from 56V to 80V and change the 2.2K 2-watt resistors to 3.3K 3-watt, and the 3.9K 1-watt resistor to 5.6K or 4.7K 2-watt. For a 20mA interface it is tentatively suggested that the circuit of *Figure 8* is tried instead of the RS-232 TxData circuit in the manuals.

Ray Fox

Copyright 1980 OSI UK User Group, unless otherwise stated.

'Garbage collector' listing: main text copyright Microsoft Inc.; alterations/corrections 'public domain'.