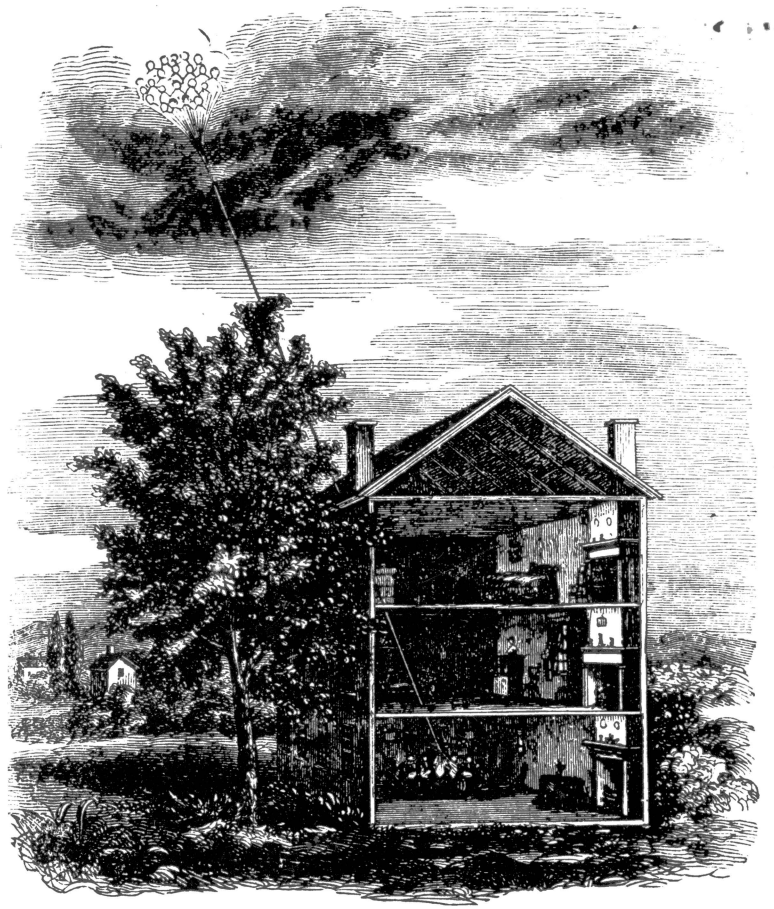
OSI UK User Group Newsletter

Volume 2 No. 6 November/December 1981



More on Communications — the Modem

Indirect files in ROM BASIC
Hooks into BASIC

New products from MUTEK

To get the year off to a good start and increase the capabilities of your system here are a few hardware products and some excellent software offerings.

One of the questions we are continually asked is "How do I put disk drives on a Superboard?" We now have available a very high quality board to do just that. The MCM9 board is a bus compatible 10 × 12 card giving space for up to 24K of static memory, a real time clock and an OSI standard floppy disc controller. The controller will operate with 5¼" or 8" drives but does not include the data separator required for most 5¼" units. Full details for interfacing are supplied with any version but we cannot offer advice on interfacing to other than Shugart type drives.

MCM9/A	Bare board	**£37.00**
MCM9/B	Floppy disk controller section populated	**£75.00**
MCM9	Complete card fully populated	**£185.00**
FL1	Header card for disk ribbon cable with connection to MCM9	**£6.00**
FLC4	Complete ribbon cable with connectors for 2 × 8" drivers and FL1 4 ft long.	**£32.00**
FL2	40 way ribon cable with DIL header at one end	**£4.50**
FL3	Molex connectors 12 way plug or socket	**£0.25**

We expect to have stocks of 8" drives available by the end of January. Prices will be around **£250**.

Communications

The M555 is a multiple communications and memory card and when fully populated gives 2 parallel and 5 serial ports and up to 8K of static memory. The ports are designed to support OSI's Centronics and Diablo printer drivers but can be used as general parallel I/O ports. The board is quite expensive when fully populated but we can offer this as a bare board complete with Molex connectors at very realistic price.

M555 board with documentation and Molex connectors **£37.00**

Both the M555 and the MCM9 are designed for use with the OSI 48 pin backplane but will connect to a Superboard or UK101 using a 40 pin header and ribbon cable and 4, 12 way molex plugs. A neater method is to use the Zen expansion board advertised elsewhere!

Software

A few good quality products this month, both utilitarian and good fun.

Mini-Compiler

This is a 5.8K basic program that converts a sub-set of Microsoft Basic into machine code. For those not familiar with compilers they tend to be large, slow and very expensive. This one is small, slow and fairly cheap and produces relocatable machine code which does not rely on any routines in ROM BASIC. The code generated will hence operate on any 6502 based machine with minimal changes.

Little knowledge of machine code is required but a working capability makes it more fun to use and provides a greater base for imaginative use. The advantage of a compiler is that programs may be easily produced in a high level language and then converted to machine code for very fast operation. This will add a new dimension to video action! The compiler does not support floating point arithmetic or string handling but does cope with the following:

Legal variables A-Z (positive integers 0 to 64K) A=N (0≤N<64K)

A=B	A=B OR C	POKE A,B	GOTO N
A=B+N	A=B AND C	A=PEEK (N)	A=D*B
A=A+C	A=B OR N	POKE A,N	A=D*N
A=B−C	A=B AND N	GOSUB N	A=B/C

IF A=B THEN GOTO	RETURN	X=USR(X)
IF A=B THEN GOSUB	STOP	FOR I=A TO B STEP C
REM	NEXT I	

and similarly for >, < and <>
and several others!!

The Price: **£15.50** cassette and documentation

See our separate software catalogue for details of our new range of software and firmware.

All prices quoted exclude VAT.

MUTEK

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

Contents

The **OSI/UK User Group Newsletter** is published by the OSI/UK Group, ©1981. Unless otherwise stated, copyright on each article is held jointly by the Group and the author. Requests for permission to reprint articles will normally be given where the article was origninated by the Group, if the request is made in writing. The *Newsletter* is published approximately bimonthly, subject to the limitations of our essentially 'spare time' operation. Subscriptions run from January to December: thus subscriptions received during the year are automatically back-dated and new members receive the year's issues to date. The UK subscription is £10 per annum; overseas subscriptions are £11 surface mail, £14 airmail. All subscriptions include six issues of the *Newsletter*. Volume 1, produced during our first year and consisting of four issues, is available as a complete set at £7 UK, £8 overseas, including postage. *Cheques should be made payable to the* **OSI/UK User Group**.

Advertising space is available in usual magazine format positions: rates are available on request. Small ads are accepted at a rate of 5p per word and will normally be published in the next available Newsletter.

The **OSI/UK User Group Newsletter** welcomes contributions of any length on subject which will interest members. Articles should preferably be typed or printed double-spaced on one side of the paper, or legibly handwritten. Articles will also be accepted on cassette or disk.

The **OSI/UK User Group** exists to assist and inform the user of Ohio Scientific and related computer systems, primarily via the Newsletter. Written queries are welcomed, although due to committments we can only guarantee to reply when we have time available. An SAE should be enclosed with all queries, and if data sheets etc. are required, the SAE should be sufficiently large!

All enquiries, subscriptions, articles etc. should be addressed to the **OSI/UK User Group**, 12 Bennerley Road, London SW11 6DS, England.

OSI/UK User Group Newsletter

Editorial

At last, the 'final 1981 issue'. We're gradually clawing back the lateness which developed around the middle of 1981, partially by working on two issues almost at once. As a result, we should be able to offer Volume 3 number 1 within a very short time from now. From then on, we should be back on sync.

Which brings us on to another point. The end of volume 2 (this issue) also marks the end of 1981 subscriptions. Now that we have rationalised the subscription system, they all end at once. We hope, therefore, that you have found the last six issues useful and interesting and worth the £10 you sent us. In 1982 we will, with your support and resubscription, be able to produce another six issues. You will have noticed that the last couple of issues have been larger than previously: we intend this to continue, yet we will not be raising the subscription. Inside this issue you will find a resubscription form; we very much hope that you will want to continue to be a member of the Group. There is no other publication, even in the United States, which offers so much information on OSI machines and related equipment. We also hope that our technical query service has been useful to you during the year, and that we have been able to help you, both personally and through the Newsletter, to get the best out of your system. We sincerely hope that you will want to continue to take advantage of what we are offering.

Already lined up for the coming issues of the Newsletter are a number of important features that will interest almost all users, whether their interests lie in hardware or software, BASIC or machine code. Next issue begins a major series on upgrading to disk: starting from first principles, the series will help you find your way through the jungle of options and systems, to get what you need at a price you can afford. Drives are getting cheaper all the time, and disk storage opens a whole new world of speed and flexibility for your sysem. Also slated for the coming months are articles on enhancing BASIC, and the usual complement of hints, programming tips, hardware modifications, BASIC and machine-code utilities and applications programs, disk notes plus further articles on getting your computer to communicate with the outside world. This year should also see our own Bulletin Board get off the ground. There are plenty of interesting and useful things due to become available to User Group members in 1982: we hope you'll continue to stay with us to enjoy them! Happy computing in 1982!

Richard Elen

String graphics

Roger Derry

This program is a development of Aardvark's screen-clear routine using the screen as string storage. I reasoned that the following would be true:

- If you can write blanks to the screen then any character can be written.
- Instead of strings of single characters, *text* could be written.
- Having told the computer that a particular string variable was in screen RAM, when that variable is called again the computer will look for it on the screen and find the *current* contents of that portion of the screen.

The program demonstrates this by printing centred text on the screen and then calling a simple graphics-drawing routine. At any point the contents of the screen can be stored or the store recalled. At first glance you might think that text would come out backwards: it doesn't, as BASIC stores text backwards going backwards down the memory!

```
10 REM * Experimental graphics package *
20 REM * Roger Derry *
30 REM * (for Compukit UK101 with MONUK01) *
35 GOTO 10000
40 SC=53251
50 GOSUB 2900: REM initialize and clear screen
90 POKE 15,255
100 FOR X=1 TO 255: A$=A$+"#": NEXT
110 GOSUB 1000
120 FOR DD=1 TO 3000: NEXT
130 GOSUB 3000: REM clear screen
140 POKE 11,0: POKE 12,253
145 REM PEEK(519) gives position of cursor along
146 REM line in New Monitor Compukit
147 REM Loc 512 can be used in old monitors
150 X=USR(X): KS=PEEK(531) IF PEEK(519)>46 THEN PRINT
151 IF KS=1 THEN 200
152 IF KS=2 THEN 300
153 IF KS=4 THEN ZZ=0
154 IF KS=5 THEN ZZ=127
155 IF KS=6 THEN ZZ=64
156 IF KS=7 THEN 500
157 IF PEEK(531)<33 THEN PRINT CHR$(PEEK(531));: GOTO 150
158 IF KS+ZZ>255 THEN 150
160 PRINT CHR$(PEEK(531)+ZZ);
170 GOTO 150
200 M1$=S1$: M2$=S2$: M3$=S3$: M4$=S4$: GOTO 130
300 A=PEEK(129): B=PEEK(130)
310 POKE 129,255: POKE 130,211
320 S1$=M1$: S2$=M2$: S3$=M3$: S4$=M4$
325 S4$=S4$+"": S3$=S3$+"": S2$=S2$+"": S1$=S1$+""
330 POKE 129,A: POKE 130,B
340 GOTO 150
```

```
500 GOSUB 3000
510 SAVE: POKE 15,255
520 PRINT M1$
530 PRINT M2$
540 PRINT M3$
550 PRINT M4$
630 GOSUB 3000: PRINT: PRINT "String transfer ended"
640 FOR DD=1 TO 4000: NEXT: GOSUB 3000
650 GOTO 300
1000 A=PEEK(129): B=PEEK(130)
1010 POKE 129,255: POKE 130,211
1025 S4$=A$+"": S3$=A$+"": S2$=A$+"": S1$=A$+""
1030 POKE 129,A: POKE 130,B
1040 RETURN
2000 GOSUB 4000
2001 GOSUB 3010: POKE 129,0: POKE 130,210: GOSUB 4000: GOSUB 3030
2005 FOR DD=1 TO 5000: NEXT
2010 PRINT: PRINT
2020 DATA"This program uses a development of the Aardvark
2030 DATA"screen clear routine. This works by setting
2040 DATA"the string storage pointers (locs 129, 130) to
2050 DATA"point to the screen RAM
2060 DATA"     If this is done a string addition
2070 DATA"will cause the string to
2080 DATA"be 'printed' to the screen. The text of the
2090 DATA"string will end at the address pointed to.
2100 DATA"This program first sets up a 255-character
2110 DATA"string and 'prints' it onto the screen as 4
2120 DATA"different variables, S1$, S2$, S3$, S4$",
2123 DATA"To make this clear the
2125 DATA"#
2130 DATA"character is used to fill the screen.
2135 DATA,"Press any key to continue                "
2140 DATA Once this has been done the four string
2150 DATA variables 'live' on the screen. Although the
2160 DATA screen's contents will change the computer will
2170 DATA look for the contents of those variables on the
2180 DATA screen. So if any other variable is made equal
2190 DATA to one of the screen variables (A$=S1$) then A$
2200 DATA will be loaded with the current contents of the
2210 DATA screen. This program will store the current
2220 DATA screen contents in four variables when
2230 DATA Ctrl-A is pressed.
2240 DATA Ctrl-B will fill the screen with the stored
2250 DATA contents (Ctrl-C stops the program as usual),
2255 DATA,"Press any key to continue                ",
2260 DATA Ctrl-D selects the keyboard to operate normally
2280 DATA Ctrl-E shifts the keyed character up
2290 DATA 64 ASCII codes
2300 DATA Ctrl-F shifts the keyed character up
2310 DATA 128 ASCII codes
2320 DATA Ctrl-G outputs the stored screen variables to
2330 DATA cassette,
2340 DATA With New Monitor Compukits the cursor controls
2345 DATA can be used
2500 DATA This program is only for demonstration
2510 DATA It has not been optimised
2530 DATA,"Press any key to continue                "
```



```
2898 END
2899 REM Initialise clear screen
2900 FOR X=1 TO 64: CS$=CS$+CHR$(32): NEXT
3000 REM Clear-screen (Aardvark)
3005 CS$=CS$
3010 A=PEEK(129): B=PEEK(130)
3020 POKE 129,0: POKE 130,212
3030 FOR O=1 TO 16: CS$=CS$+"": NEXT
3040 POKE 129,A: POKE 130,B: RETURN
4000 CS$="          STRING GRAPHICS          "     : RETURN
5000 :
6000 :
10000 REM * Print to screen using string storage *
10010 REM * by Roger Derry *
10020 REM
10080 :
10100 REM Print from data list
10110 S$=CHR$(32)
10120 FOR O=1 TO 30: B$=B$+S$: NEXT
10130 FOR Q=1 TO 16
10140 READ M$
10150 IF M$="#" THEN RUN 40
10160 GOSUB 10210
10170 NEXT
10180 POKE 11,0: POKE 12,253: X=USR(X)
10190 GOTO 10130
10200 END
10210 SC=53246
10220 :
10230 REM Find end of required line
10240 :
10250 AD=SC+(64*Q): A1=INT(AD/256): A2=AD-(A1*256)
10260 A3=PEEK(129): A4=PEEK(130): L=52-LEN(M$)
10270 M$=LEFT$(B$,L/2)+M$+LEFT$(B$,L/2)
10280 POKE 129,A2: POKE 130,A1: M$=M$+""
10290 POKE 129,A3: POKE 130,A4: RETURN
40000 DATA,#
```

CEGMON 'Print at'

C. Hurt

```
 1 REM * subroutine using CEGMON screen handler with UK101 *
 2 REM * to replace 'print-at' facility *
 3 REM * using X/Y co-ordinates to give *
 4 REM * line-number, column-number *
 5 REM * e.g. Line 12 Col 25 is PA=1225 *
 6 REM * e.g. Line 7 Col 8 is PA= 708 *
 7 REM * In use – e.g. [ PA=1225: GOSUB 10 ] *
 8 REM * end each PRINT$ with ';' *
 9 GOTO 20
10 PRINT CHR$(12);: X=INT(PA/100): Y=PA-(X*100)
11 IF X/4=INT(X/4) THEN K=205
12 IF (X+1)/4=INT((X+1)/4) THEN K=141
13 IF (X+2)/4=INT((X=2)/4) THEN K=77
14 IF (X+3)/4=INT((X+3)/4) THEN K=13
15 W=K+Y: Z=INT((X+4)-.1)+208
16 Z=INT((X/4)-.1)+208
```

```
 17 PRINT: POKE 555,W: POKE 556,Z
 18 RETURN
 20 REM * start of user program *
 30 CL$=CHR$(26)
 40 PRINT CL$;: GOSUB 200
 50 PA=1515: GOSUB 10: PRINT "This is the test";
 55 GOSUB 200
 60 PA=101: GOSUB 10: PRINT "Top left";
 65 GOSUB 200
 70 PA=3230: GOSUB 10: PRINT "Bottom right";
 75 GOSUB 200
 80 PA=130: GOSUB 10: PRINT "Top right";
 85 GOSUB 200
 90 PA=3201: GOSUB 10: PRINT "Bottom left";
 95 GOSUB 200
100 PA=(INT(RND(1)*30)+1)*100+(INT(RND(1)*25)+1
110 GOSUB 10: PRINT "< PA=";PA;
120 GOSUB 200: GOSUB 200: GOSUB 200
150 GOTO 40
200 FOR P=1 TO 1000: NEXT P: RETURN
```

Using WAIT

Jack Pike

[Unfortunately, when we printed this item in Volume 2 issue 4, we omitted the program, which enables programs to avoid programs with string delimiters. Here it is. —Ed.]

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

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

```
 10 POKE 133,0: POKE 134,8: CLEAR
 12 J=2048: DIMP(99)
 14 CA=61440: CB=61441: CC=57088
 16 POKE 530,1
 18 POKE CC,253
 20 WAIT CA,1
 30 P=PEEK(CB)
 34 IF PEEK(CC)=239 GOTO 100
 40 IF P>31 THEN 60
 45 IF I=1 GOTO 20
 50 P(J0)=J: J0=J0+1
 52 IF J0<2 GOTO 55
 53 FOR L=P(J0-2) TO P(J0-1)-1
 54 PRINT CHR$(PEEK(L));: NEXT L: PRINT
 55 I=1: GOTO 20
 60 IF I=1 AND (P>57 OR P<49) THEN 20
 65 I=0
 70 POKE J,P: J=J+1
 80 GOTO 20
100 POKE 530,0
105 SAVE
110 A$=CHR$(0)+CHR$(0)
120 FOR J=0 TO J0-1
130 PRINT A$;A$;A$;A$;CHR$(10);CHR$(13);
140 FOR L=P(J) TO P(J+1)-1
150 PRINT CHR$(PEEK(L));
160 NEXT L: PRINT
170 NEXT J
180 POKE 517,0
```


UK101 subroutine RETURN killer

Dave Woolcock

Jumping out of FOR-NEXT loops and GOSUBs before they are finished usually results in 'out of memory' errors as the stack gets clogged up with return addresses etc. The solution for loops is easy (once you see it).

```
Eg, in a loop   FOR I = 1 TO N
                ...
                ...
exit with       IF (whatever) THEN I=N (wherever)
                ...
                ...
                NEXT I
```

For subroutines there is no easy solution as RETURN is the only way out, and that will send the program back to where it was called from — not where we want to go to.

The program in **fig 1** will cancel a RETURN for UK101s with the new BASIC 1 chip just by CALLing it and then using GOTO.

For the original BASIC 1 chips use the program in **fig 2** by POKEing addresses 11 and 12 with its start address and doing an X=USR(X) from within the subroutine, then using GOTO.

To kill two or more levels of subroutines just CALL the program the required number of times.

If the program is not in a GOSUB at the time or is also in an unfinished loop (which hasn't been dealt with as above) then a new error code is generated: NR ERROR IN LINE ... –ie a NO RETURN error. Old BASIC 1 chips will produce squiggle.

The program can be located anywhere convenient (eg page 2) or if you have new BASIC 1 (or an input-vector 'unmask' routine) then you can embed the program in a REM statement. To do this enter line 0 REM """" line of quotes """" then enter the monitor and locate the start of the quotes (ASCII $22) which should be at $0306 for the first line of a program. Enter the program in hex from the next location, $0307, and warmstart. The REM line should now contain a few graphic nonsense characters plus a few spare quotes which can be edited away.

The reason for the quotes is so that the BASIC program REM line can be saved and loaded without the interpreter detokenising them. Line 0 is preferred as it can't be shifted by insertions of other BASIC program lines, in which case the routine is operated by CALL 775.

I find this routine useful for using the 'control' key for overriding the normal exits from keyboard polling subroutines.

As far as I am aware there is no reason why this shouldn't work on the Superboard.

fig 1 for use with CALL

```
entry   68          PLA
        68          PLA             ditch CALL return address
        68          PLA             pick up GOSUB token
        C9 8C       CMP #$8C
        D0 05       BNE error       error if not a GOSUB
        68          PLA
        68          PLA             ditch GOSUB's line number
        68          PLA
        68          PLA             ditch GOSUB's line pointer
        60          RTS             and exit
error   A2 03       LDX #$03
        4C 4E A2    JMP $A24E       fix error pointer and jump
```

fig 2 for use with X=USR(X)

```
entry   A2 0B       LDX #$ 0B
loop    68          PLA
        CA          DEX             ditch 10 USR stack levels
        D0 FC       BNE loop
        C9 8C       CMP #$8C        error if not GOSUB token
        D0 05       BNE error
        68          PLA
        68          PLA             ditch GOSUB's line number
        68          PLA
        68          PLA             ditch GOSUB's line pointer
        60          RTS             and exit
error   A2 03       LDX #$03
        4C 4E A2    JMP $A24E       fix error code and jump
```

BASIC Quickies

From Trevor Watson: This routine 'centre-justifies' a line of print:

```
1000 PRINT TAB ((44-LEN(Z$))/2);Z$:RETURN
```

For example:

```
10 Z$="I'm in the middle":GOSUB 1000
20 Z$="So am I":GOSUB 1000
```

In line 1000, 44 is the screen width, and should be adjusted appropriately.

The GETKEY in the August/September issue didn't work for me! I have always used this:

```
10 POKE 11,0: POKE 12,253
20 PRINT "Press any key"
30 X=USR(X): P=PEEK (531)
40 PRINT CHR$(P)" was pressed"
```

Or, for example in a games program, I might test for key A by:

```
40 IF PEEK(531)=65 THEN...
```

The USR jump POKEs only have to be specified once, of course.

DISK UPDATE

OS-65D with enhanced Superboard

David Anderson

I was recently called upon to modify a 5¼-inch disk system (OS-65D) to run a C1 with large screen. The master disk, OS-65D V3.1 as supplied by Mighty Micro, did not contain the extended monitor, as the supplied documentation would have one believe.

The modification is, however, straightforward and is outlined below:

1. Boot up DOS — from a copy diskette — *never write to your original diskette.* It doesn't matter which system you boot up into (C1 or C1 modified). If you boot into C1 modified then you will just have to put up with the 'funnies' on the screen until you have finished these mods.
2. UNLOCK — from the basic program BEXEC*
3. EXIT — from basic into DOS
4. CA 0200=13,1 — call in disk utilities
5. GO 0200 — execute disk utilities
6. 2 — select No. 2 from utilities
7. R4200 — read into $4200+ the contents of track zero
8. E — exit utilities
9. GO FE00 — execute monitor — *do not press BREAK and M*
10. Do the following changes. All the addresses are offset by 8K ($2000), hence when they are booted they will exist at $2200+ and not $4200+.

Address	*Originally*	*Change to*	*Remarks*
45C3	D3	D7	cursor start address high
45C6 *	7D	73	cursor end address low
45CD	D3	D7	
45D5	D3	D7	
45FB	1F	3F	test for 64-character line
45FD *	04	07	left overscan –1
4602 *	08	0C	left overscan + right overscan
4608	1F	3F	
460A	1D	3C	
460F *	07	0B	left + right overscan –1
4616	D3	D7	
461D	D3	D7	
4624	D3	D7	
4626 *	65	40	low address of start cursor
462A	65	40	
4630	D3	D7	
4637	7D	7C	
463A	D3	D7	
4648	20	40	number of addresses scrolled

4649	D0	D7
4653	D3	D7

* The values of these addresses may be changed to position cursor correctly.

11. .0200G re-enter diskette utilities
12. 2 select utility No. 2
13. W4200/2200,8 rewrite $4200+ to track zero with mods
14. Reboot disk.

If the modifications are made on a C1-modified, the screen should now behave itself. If modifications are made with a C1, the screen will be blank after boot. In this case, to test the disk run a program from the disk — preferably one which POKEs to the screen. This should cause the disk drive to click a few times before the pokes to the screen appear.

Disk notes

Himem on disk systems is found by looking for the first byte of RAM as opposed to ROM or nothing by working from $C000 downwards. This means that systems with a discontinuous block of RAM could be severely fooled!

From *Peek 65* or somewhere: JMP (03FF) will fail on all 6502s.

CEGMON and Pico-DOS

G. Heath writes: I have recently purchased a CEGMON monitor, and have found that it is not compatible with the Pico-DOS software. However the patch is straightforward:

Memory locations $25D9 changed to 46
$25DE changed to FB.

This is part of a subroutine which initialises the input vector. This is located on track zero of the Pico-DOS disk. The OS-65D track zero R/W Utility can be used. If track zero is read into locations $4200 on, locations $45D9 and $45DE must be changed, and the track rewritten.

For more general use, eg for other versions of CEGMON or other monitors, replace the contents of $24D9 and $25DE with the low and high bytes respectively of the address found at the input vector location (INVEC).

Hitachi CMOS RAM

Olof Swembel of Sweden writes: The new Hitachi 2K CMOS static RAMs, type *HM6116P-3* are compatible with standard 2716 EPROMs, but pins 18 ($\overline{CS}$) and 20 ($\overline{OE}$) have to be switched by jumpers to the socket or rewired on the memory board. Also, pin 21 ($\overline{WE}$) has to be wired to the common signal running to pin 10 on 2114 RAMs.

MACHINE-CODE PROGRAMMING

Displaying numbers in machine-code programs

by J.M. Leach

When you write a machine-code program it is quite difficult to work out a way of displaying intermediate numeric values, or producing meaningful output on the screen, although of course all this is handled within the BASIC interpreter. Volume 1 issue 1 gave the addresses of the necessary routines, but it is not quite so simple to get them actually to work. The following short programs show the way to using this information in practice.

2-byte unsigned hex numbers converted to decimal

Copy the 2 bytes to $AD,$AE with the most significant byte first, unlike memory reference in machine code. Then call $B962 to print the number, then $A86C if you want a CR/LF. The sample program prints all numbers from 0 to 65535, rather quickly. Use RESET to stop it.

Demonstration program to print 2 bytes as unsigned decimal number

```
1100 A900     LDA #$00      ; Initialise
1102 85E8     STA $E8
1104 85E9     STA $E9
1106 A5E8     LDA $E8       ; Copy E8,E9 to AD,AE
1108 85AD     STA $AD
110A A5E9     LDA $E9
110C 85AE     STA $AE
110E 2062B9   JSR $B962     ; Display number
1111 206CA8   JSR $A86C     ; CR/LF (optional)
1114 E6E9     INC $E9       ; Add 1 to 2-byte number
1116 D0EE     BNE $1106     ; and go back to display it
1118 E6E8     INC $E8
111A D0EA     BNE $1106
111C 00       BRK           ; Back to Monitor
```

Floating-point numbers

Volume 1 issue 2 described how FP numbers are stored in memory, but the print routine is not quite like that. $AC holds the exponent and $AD to $AF hold the mantissa. The difference is that the sign is given by $B0, ie negative if the most significant bit is set (≥ 80 hex).

Subroutine $B962 sets up what it finds in $AC-B0 as ASCII characters at the bottom of the stack, altering the contents of $AC-B0 as it does so. To display the number, the message printer, $A8C3, is called, as the following subroutine demonstrates. The contents of $E8-EC are used to hold your number and are copied to $AC-B0.

Relocatable subroutine to print floating-point number

```
1000 08       PHP
1001 A204     LDX #$04      ; Copy 5 bytes to FP accumulator from
1003 B5E8     LDA $E8,X     ; your own area
1005 95AC     STA $E8,X
1007 CA       DEX
1008 10F9     BPL $1003
100A 206EB9   JSR $B96E     ; Set up ASCII characters in stack
100D A001     LDY #$01      ; Set up address for message printer
100F A900     LDA #$00
1011 20C3A8   JSR $A8C3     ; Display number
1014 206CA8   JSR $A86C     ; CR/LF (option — otherwise replace with EAs)
1017 A900     LDA #$00      ; A fix is required to prevent an error message
1019 855F     STA $5F       ; on return from an X=USR(X) in BASIC
101B 28       PLP
101C 60       RTS
```

The following BASIC program will allow you to experiment with the floating-point routine, but don't try to list all possibilities as it might take quite a long time.

```
 10 FOR I = 4096 TO 4124 : REM Set up machine-code
 20 READ Z: POKE I,Z: NEXT I
 30 DATA 8,162,4,181,232,149,172,202,16,249,32,110,185,160,1
 40 DATA 169,0,32,195,168,32,108,168,169,0,133,95,40,96
 50 P=232: REM =E8, offset for data insertion
 60 POKE 11,0: POKE 12,16: REM Change if you relocate the subroutine
 70 FOR I=0 TO 4
 80 A=0: REM Try A=255
 90 POKE P+I,A
100 NEXT I
110 POKE P+1,128: REM Experimental; try < 128 for an eventual crash!
120 FOR I=120 TO 150
130 POKE P,I: REM Experimental display
140 FOR J=P TO P+4: PRINT PEEK (J);: NEXT J: PRINT "    ";: REM Show data
150 X=USR(X): REM Show result
160 NEXT I
```

Line 110 gives a clue to printing the numeric value of variables in BASIC, for example in a debugging aid that lists the current value of all variables. AND the content of $AD with $80 and copy to $B0. Then OR $AD with $80 to set the most significant byte. A further report later.

BASIC EOR

It is, in fact, possible to write a single-line Exclusive-OR routine in BASIC, *writes Steve Smith,* contrary to what was suggested in Part 6 of *BASICs of machine code,* without using a FOR...NEXT loop. The following function assigns the Exclusive-OR of A and B to C:

```
C=(A OR B) AND NOT (A AND B)
```

It is especially pleasing as it even 'reads right'!

Assembler Notes

A clever trick in the OSI assembler is that if you are lazy and type 0 as the line number, the new line will be placed in the file after the last line printed. This is useful in several ways. For example, it makes the formatting of the source code much easier to do so that you get a tidy program to read; it can save you key strokes; and it allows you to add the last few additional instructions in a file when you have already filled up the original gap left between line numbers! Now, thanks to OSUIN, yet another trick is revealed. If you have ever wished to have a library of assembler source code subroutines so that you could just feed them into the source code when they were needed instead of having to type them in, here's how to go about it. Load the Assembler with the relevant program module. Exit the assembler and use the ROM monitor (the extended monitor is advised on disk systems) to change locaion $12D3 from $0A to $00. Warm start the assembler and resequence your module. Exit the assembler as before and change location $12D3 back to $0A. Warm start again and observe that your code will print out without line numbers. Now save the source code on tape or disk. When you want to use the module in a program, all you have to do is to print up to the point at which the new code is to be inserted, download the module(s) from cassette or from the indirect file, and finally resequence the entire file. Your module should now be happily installed in its new home. It is unwise, to say the least, to attemp to edit or delete a source file which has lines numbered zero in it – you have been warned!

The above procedure has been checked out quite thoroughly with the disk-based systems and seems to work on cassette-based ones, although a few spurious graphics and odd line numbers may occur with the latter. As usual we would be delighted to hear from anyone who sorts this out. On disk systems, you would obviously have to load the program module into the indirect file first, then load the real file before finding the right place to download the module.

BASIC quickies

Bill Farmer

High-speed loading...

The reason for the double characters while loading BASIC at high baud rates is that the time taken for the screen to scroll causes an overrun error to be generated by the ACIA. It needs two reads of the control register to clear the error: one theoretically to read the error bit and do something about it, and one to get the next character. Thus you get a repeat of the first character after the scroll. This effect also makes loading horrid hex assembler tapes hard work.

...And a stack mystery

The funny 'look back up the stack' routine at $A1A1 in the BASIC ROMs is called by the FOR routine to see if a similar FOR is outstanding. If it is, the stack pointer is adjusted so that the new entry overwrites the old. So you can jump out of a FOR-NEXT loop, despite what they say. Try:

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 10
30 GOTO 10
```

This should fill up the stack and crash with an OM error, but it doesn't.

FEATURES

Indirect file handler
For ROM BASIC systems
George Chkiantz

This article describes an indirect file handler for ROM BASIC systems in some detail. I do not claim that the machine code is either the most elegant or the only way of attaching this potential to your system, but I hope that it will provide a good basis for experiment. The code emulates the handler used in the disk-based operating systems, although it differs on one important respect in that control of the system is trapped during output rather than on input, unlike the DOS. This allows the handler to be used by BASIC programs as well as in the direct mode, so that BASIC programs can use memory as an input or output device. An added extra is that <CTRL S> will halt screen output until some other key is pressed, as in the disk systems.

The program listing is in assembler format and has been vectored to start at your favourite location for machine code patches, viz $0240. Originally assembled and tested for a *C2* running CEGMON, the program is easily modified to run on a *C1/UK101*,as the only changes required relate to the inverted keyboard.

The four locations in lines 80–110 relate to required routines. INVEC, the input vector, is almost universally in the same place, although this would have to be changed if you have a monitor that is not compatible with OSI's page–2 usage - for example this may be necessary with WEMON. FILPTR must be two consecutive locations in zero-page that are *not* in the input buffer; again you may have to change these to suit your system monitor. The remaining locations are defined at this point for the convenience of the assembler – they are set up by the startup sector anyway.

This program is written in three sections, the startup section just mentioned links the handler into your system. The other two sections link into the input vector and the output vector of the computer.

Memory location $02BE (A–HOLD) is used to store the accumulator temporarily and $02BF is used as a flag FILFLG. This can contain one of three values, 00 which signals that the indirect file is not in use, $80 if a write is in progress and $40 if a read is in progress. Input is routed through the routine shown in lines 170–340. In line 170, a BIT test loads the status register with bits 6 and 7 of FILFLG and, if bit 6 is clear, the test in line 180 fails and the routine jumps to the normal input handler (line 190). If the test in line 180 succeeds, the Y register is saved and cleared (lines 200–220) and the accumulator is loaded from the current location in the file pointed at by FILPTR. The code generated in line 240 'hides' the next instruction at 250, so in line 260 a call is made to the subroutine INCPTR which increments FILPTR. On return the character in the accumulator is checked for a $5D–<SHIFT M>– which terminates input and output operations. If this is not the case, the Y register is retrieved and the routine returns, whereas if a $5D was detected, FILFLG is cleared to return control to normal.

The output vector is routed through the code starting at line 450, where a BIT test is used to check bit 7 of FILFLG (BIT tests only change the status register). If bit 7 is set the test in line 460 will fail, the value in the accumulator is saved and

printed on screen (line 480), the Y register is saved and cleared (lines 490–520) and the routine jumps to the hidden instruction at line 250, writing to the file etc. Thus lines 250–340 are used by both read and write to the file, courtesy of the BIT opcode infiltrated by line 240. Now, if the test in line 460 had succeeded (no output to file), a check is made for the file control characters <SHIFTK> and <CTRL X> (lines 540–570) and if neither of these is found, the keyboard is checked for a <CTRL S> which, if found, will halt output until some other key is pressed before the output is (finally) put on to the screen.

In the event that a <CTRL X> or a <SHIFT K> (these, as with <SHIFT M>, have been chosen to maintain compatibility with the system used by disks – you can use what you like by changing the arguments of the appropriate CMP instructions) was detected in the output stream, the section SETO.P (lines 680–760) is used to set FILFLG to the appropriate value by setting it to $80 and shifting the flag right (= $40) if a control X was detected. A subroutine call in line 710 sets up FILPTR to the start of the file (lines 780–820). *You may need to change the values shown in line 780 and line 800 to suit your system.*

The subroutine INCPTR (lines 360–410) does no range checking other than to abort the handler if it attempts to go beyond $FFFF. No check is made to see whether memory actually exists for the file, as in the disk version; this is left to the user.

The section from line 890 on merely sets up the system so that the user can reset the computer, do a warm start and go CALL 704 to link in the handler. Reset, M, 02C0G would work just as well. This code was mainly used for convenience in testing, and its ommission would save a certain amount of room for other machine code routines. If this is done, the labels INPUT and OUTPUT would have to be correctly defined for your system, and you would have to POKE INVEC and OUTVEC to point to the appropriate points in the routine – $0240 and $0270 - to setup the system at any time you pressed <BREAK>. Should the handler lose its closing marker ($5D) - say you have written beyond the end of existing memory - it may be necessary to POKE FILFLG to 0 in order to regain control of the handler on restart. In this case the contents of the indirect file (as far as it got), can be recovered by using the monitor, before trying to download.

One of the problems with a small machine is to define a sensible starting point for the indirect file. This is set by the subroutine SETPTR (lines 780–820). The code suggested in the comment field (LDA $7B, etc.) would make the file start immediately after the BASIC source file, while (LDA $85...LDA $86) would make the file start in a reversed area determined on cold start by memory size. All of these options could be useful depending on the circumstances. The locations given in the main code are the same as the default locations on disk systems: this would allow the user to transform programs between systems. It should be noted that there is no reason why (if you are downloading into BASIC using this system) the BASIC source should not overwrite the start of the indirect file – it will never catch up! Thus, for some applications (say the use of high speed cassette interfaces) it could be sensible to start the file at $0340.

A general discussion on the use of indirect files is the subject of a later section in this newsletter.

```
10                      ;Indirect file handler
20                      ;for ROM BASIC systems
30                      ;  OSI/UK USER GROUP
40                      ;       (c)1981
50                      ;
60                      ;
```

```
 70                             ;
 80 0213=                       INVEC  = $0213  ;$F098 For Cegmon D
 90 00FE=                       FILPTR = $00FE
100 FF9B=                       OUTPUT = $FF9B  ;Dummy locations
110 FB46=                       INPUT  = $FB46  ;  "       "
120                             ;
130 0240                        * = $0240       ;Adjust to suit system
140                             ;
150                             ;Sector linked to i/p
160                             ;
170 0240 2CBF02   START  BIT FILFLG
180 0243 7003            BVS RDFILE
190 0245 4C46FB          JMP INPUT
200 0248 98       RDFILE TYA
210 0249 48              PHA
220 024A A000            LDY £$00
230 024C B1FE            LDA (FILPTR),Y
240 024E 2C              .BYTE $2C       ;hides next instruction
250 024F 91FE     WRITFL STA (FILPTR),Y
260 0251 206402          JSR INCPTR
270 0254 C95D            CMP £$5D        ;SHIFT M CLOSES FILE
280 0256 D003            BNE NOTEND
290 0258 8CBF02          STY FILFLG
300 025B 8DBE02   NOTEND STA A.HOLD
310 025E 68              PLA
320 025F A8              TAY
330 0260 ADBE02          LDA A.HOLD
340 0263 60              RTS
350                             ;
360 0264 E6FE     INCPTR INC FILPTR
370 0266 D007            BNE ENDINC
380 0268 E6FF            INC FILPTR+1
390 026A D003            BNE ENDINC
400 026C 8CBF02          STY FILFLG
410 026F 60       ENDINC RTS
420                             ;
430                             ;Sector linked to output
440                             ;
450 0270 2CBF02   ST.O.P BIT FILFLG
460 0273 1010            BPL NOWRIT
470 0275 8DBE02          STA A.HOLD
480 0278 209BFF          JSR OUTPUT
490 027B 98              TYA
500 027C 48              PHA
510 027D A000            LDY £$00
520 027F ADBE02          LDA A.HOLD
530 0282 4C4F02          JMP WRITFL
540 0285 C95B     NOWRIT CMP £$5B        ;SHIFT K OPENS FILE
550 0287 F018            BEQ SET0.P
560 0289 C918            CMP £$18       ;CNTRL X READS FILE
570 028B F014            BEQ SET0.P
580 028D 48              PHA
590 028E A909     HALTOP LDA £%00001001  ;£%11110110 For CI/UK101
600 0290 8D00DF          STA $DF00
610 0293 2C00DF          BIT $DF00
620 0296 5005            BVC CONT       ;BVS for CI/UK101
630 0298 1003            BPL CONT       ;BMI for CI/UK101
640 029A 2000FD          JSR $FD00
```



```
650 029D 68       CONT   PLA
660 029E 4C9BFF          JMP OUTPUT
670                      ;
680 02A1 48       SET0.P PHA
690 02A2 A980            LDA £$80
700 02A4 8DBF02          STA FILFLG
710 02A7 20B502          JSR SETPTR
720 02AA 68              PLA
730 02AB C918            CMP £$18
740 02AD D003            BNE NOTI.P
750 02AF 4EBF02          LSR FILFLG
760 02B2 4C9BFF   NOTI.P JMP OUTPUT
770                      ;                      ;SUGGESTED FILE START
780 02B5 A900     SETPTR LDA £$00               ;LDA $7B
790 02B7 85FE            STA FILPTR
800 02B9 A980            LDA £$80              ;LDA $7C
810 02BB 85FF            STA FILPTR+1
820 02BD 60              RTS
830                      ;
840 02BE 00       A.HOLD .BYTE $00
850 02BF 00       FILFLG .BYTE $00
860                      ;
870                      ;START UP SECTOR
880                      ;
890 02C0 AD1802   INIZ   LDA INVEC
900 02C3 8D4602          STA RDFILE-2
910 02C6 AD1902          LDA INVEC+1
920 02C9 8D4702          STA RDFILE-1
930 02CC A940            LDA £$40
940 02CE 8D1302          STA INVEC
950 02D1 AD1A02          LDA INVEC+2
960 02D4 8D9F02          STA SET0.P-2
970 02D7 8DB302          STA SETPTR-2
980 02DA 8D7902          STA ST.O.P+9
990 02DD AD1B02          LDA INVEC+3
1000 02E0 8DA002         STA SET0.P-1
1010 02E3 8DB402         STA SETPTR-1
1020 02E6 8D7A02         STA ST.O.P+10
1030 02E9 A970           LDA £$70
1040 02EB 8D1A02         STA INVEC+2
1050 02EE A902           LDA £$02
1060 02F0 8D1B02         STA INVEC+3
1070 02F3 8D1902         STA INVEC+1
1080 02F6 A900           LDA £$00
1090 02F8 8DBF02         STA FILFLG
1100 02FB 4C0000         JMP 0000
1110                     ;
```

Cursor up

R. Mallett writes: Having used CEGMON for some time now, I have discovered an extra cursor control for moving the cursor upwards; it may work on OSI computers without CEGMON but I am not sure. It works by first returning the cursor to the beginning of the line, then it PRINTs the RUBOUT character (CHR$(95)) to get onto the previous line, and finally PRINTs the carriage-return character again (CHR$(13)).

```
10 UP$ = CHR$(13) + CHR$(95) + CHR$(13)
20 ..... whenever the cursor control is wanted PRINT UP$; .....
```

A HIGH SPEED CASSETTE LOADER
Using the indirect file

As mentioned in the article on high speed tape handling in the June 1981 issue of the newsletter, the problems associated with the time taken to tokenise the BASIC line which limit the effective handling speed can be got over by using a special loader. A *suggestion* as to how this can be accomplished is shown in the program accompanying this article. This program is an extension to the indirect file handler also shown, and is to be used with it. It has been assembled to run on a CEGMON-based *C2* system, but the necessary changes for other systems are given. The code could be assembled to run in place of the initialisation routine shown in the file handler.

The routine assumes that BASIC has been cold started and that the file handler has been initialised. This can be achieved by <BREAK> M02E6G or by CALL 742 from BASIC. The tape to be loaded should be lined up ready and CALL 704 typed to start the loading process. The program on tape *must* be terminated by a <SHIFT M> ($5D) – this can be achieved by typing <SHIFT M> after listing the program when in SAVE mode. When the new program has been loaded, the indirect file handler is automatically turned on in the read mode, and downloads the program into the source code area. A special pointer setup procedure should allow the user to concatenate programs.

The definitions in lines 70–130 relate to the machine and monitor that you are using, those in lines 100–130 link this with the indirect file handler (nickname INDFIL). These will have to be changed if you have re-located INDFIL.

When the routine is called, the file pointer (FILPTR) is set, by the subroutine SETUP in lines 300–340, to the end of any source code (program) already in the computer, after which the main loop (lines 180–240) is entered. The control register of the ACIA is checked for a 'ready to read' flag. In lines 180–200, the Logical Shift Right puts this flag into the carry bit in the status register, after which the character that is present in the ACIA is loaded. A subroutine call to ST.O.P+5 (line 220) takes care of putting the character into memory and incrementing the pointers, after which a check is made to see whether the end of the listing has been reached (denoted by a <SHIFT M>, $5D) and the program then loops until this is so. When the end of the listing has been reached, SETUP is called to reset FILOTR to the start of the file, FILFLG is set to the read mode and the program warm starts BASIC (which then calls the indirect file for input ...).

Very high tape speeds should be possible using this routine (maybe even 9600 baud). If even greater speed is needed, this could be achieved by using the alternative code shown in lines 470 up in place of the main loop, although this would probably be too long to fit into page 2.

```
CMD ERR
.A

   10                    ;High speed cassette loader
   20                    ;(couples with INDFIL)
   30                    ;
   40                    ;   OSI/UK USER GROUP
   50                    ;       (c) 1981
   60                    ;
```



```
   70 FC00=         ACIA   = $FC00          ;$F000 for C1/UK101
   80 0218=         INVEC  = $0218
   90 FF95=         OUTPUT = $FF95
  100 0266=         INCPTR = $0266          ;See INDFIL
  110 00FE=         FILPTR = $00FE          ;      "
  120 02BF=         FILFLG = $02BF          ;      "
  130 0270=         ST.O.P = $0270
  140                      ;
  150 02C0                 * = $02C0
  160                      ;
  170 02C0 20DD02   START  JSR SETUP
  180 02C3 AD00FC   LOOP   LDA ACIA
  190 02C6 4A              LSR A
  200 02C7 90FA            BCC LOOP
  210 02C9 AD01FC          LDA ACIA+1
  220 02CC 207502          JSR ST.O.P+5
  230 02CF C95D            CMP £$5D        ;To close file
  240 02D1 D0F0            BNE LOOP
  250 02D3 20DD02   DNLOAD JSR SETUP       ;Reset pointers
  260 02D6 A940            LDA £$40        ;Set INDFIL to read
  270 02D8 8DBF02          STA FILFLG      ;mimics control X
  280 02DB D018            BNE EXIT        ;Warm start of BASIC
  290                      ;
  300 02DD A57B     SETUP  LDA $7B         ;Loads new file
  310 02DF 85FE            STA FILPTR      ;immediately after
  320 02E1 A57C            LDA $7C         ;existing source code
  330 02E3 85FF            STA FILPTR+1
  340 02E5 60              RTS
  350                      ;
  360                      ;
  370 02E6 A003     INIZ   LDY £$03
  380 02E8 B9F802   LOOP2  LDA TABLE,Y
  390 02EB 990102          STA INVEC,Y
  400 02EE 88              DEY
  410 02EF 10F7            BPL LOOP2
  420 02F1 C8              INY
  430 02F2 8CBF02          STY FILFLG
  440 02F5 4C0000   EXIT   JMP $0000       ;Warm start of BASIC
  450                      ;
  460 02F8 40       TABLE  .BYTE $40,$02,$70,$02
  460 02F9 02
  460 02FA 70
  460 02FB 02

  480                      ;Alternative code to
  490                      ;replace lines 220-230
  495                      ;
  500 02CC 91FE            STA (FILPTR),Y
  510 02CE E6FE            INC FILPTR
  520 02D0 D002            BNE ENDINC
  530 02D2 E6FF            INC FILPTR+1
  540 02D4 C95D     ENDINC CMP £$5D
```

ASCII Files and Memory I/O

by George Chkiantz

A very useful feature of OSI computers running OS–65D or OS–65U is that a section of memory may be used as an input or output device by means of routines built into the system. A similar handler for ROM–based machines is published elsewhere in this issue. With this feature it is possible to merge programs simply, pass programs between incompatible file systems (say between a word processor and the assembler or BASIC), pass programs between different versions of BASIC (ROM and disk, pass variables and strings between various programs, create command files and so on.

This aspect of the disk systems has been in use for a long time and seems to have been the main way of controlling the early versions of 65D. Disk users may like to know that it is the method used to make the computer run the program BEXEC* automatically on boot up and at no other time.

It should be noted at the outset that neither the memory output system nor its subset, the indirect file handler, have any error–trapping. Specifically, the handler does not check to see if memory actually exists at the points in question, nor will it stop writing to, or reading from, memory until actually told to do so. Thus it is essential to take this into account when using the handlers, especially as the disk control port comes straight after the user RAM in a 48K system, and writing into it accidently could do nasty things!

The first step in using the handler is to decide on the way in which memory is to be partitioned for the particular application. If the handler is to be used by a program, the programs working memory must be restricted to stop it from overwriting the indirect file and subsequently causing chaos. This is easily achieved by POKEing the following locations with the required values:

	ROM BASIC	65D	65U	ASM
Lo byte	133 ($85)	132 ($84)	132 ($84)	$12CB
Hi byte	134 ($86)	133 ($85)	133 ($85)	$12CC

With the disk version of the assembler, use the 'himem' command *Hnnnn* to set the upper memory limit to the hex address *nnnn*. The next job is to set up the handler to use the designated area of memory. In the disk systems this is somewhat complicated by the fact that there are two ways of accessing the memory handler. The one offering the most complete control is that using Device 5 in 65D (Device 4 in 65U), while more convenient for immediate commands is called the indirect file, which is a subset of the above systems with a preset start point of $8000. Change the locations shown below to suit your memory configuration.

Memory pointers for indirect file

	ROM Handler	65D (Kernel)	65U
Input Lo	694 ($02B6)	9373 ($249D)	11666 ($2D92)
Input Hi	698 ($02BA)	9368 ($2498)	11667 ($2D93)
Output Lo	694 ($02B6)	9559 ($2557)	11666 ($2D92
Output Hi	698 ($02BA)	9554 ($2552)	11667 ($2D93)

Note that both 65U and the ROM handler suggested in this issue have the same address for input and output, while the manuals only quote the Hi address for 65D (it is much easier to start the file off at a page boundary). The changes to the pointers may be poked from BASIC or can be permanently installed on the disk for convenience – even by POKEs in BEXEC*. As the indirect file handler is part of the DOS Kernel in 65D, it will work with any transient utility program, eg the assembler, Exmon or certain word processors.

The location and values of the control codes for the handler are given in the following table. These could be changed to suit your system – for example <CTRL X> is convenient with the UK101, and if accidently pressed will probably cause you to lose control of your system. Those for the ROM handler are not given here as the assembler file can be changed to suit your machine.

Indirect file control codes

System	Location	Enable	Disable	Function
65D	9550 ($254E)	91 ($5B)	255 ($FF)	Write O/P to file
	9571 ($2563)	93 ($5D)	255 ($FF)	Close file
	9594 ($2579)	24 ($18	255 ($FF)	Read file
65U	14646 ($3936)	91 ($5B)	255 ($FF)	Write O/P to file
	14677 ($3954)	93 ($5D)	255 ($FF)	Close file
	14721 ($3981)	24 ($18)	255 ($FF)	Read file

These control codes are accessed from the keyboard by typing (with shift lock down) <SHIFT K> ([), <SHIFT M> (]) and <CTRL X> respectively. The codes are trapped on input by the system, which makes the screen output a little unpredictable. The ROM handler published in this issue traps its control codes on the output, to allow it to be used easily from BASIC and so the syntax is slightly different.

When a $5B is detected on input, the indirect file handler is turned on and all output from the computer is written into memory from the defined location until a $5D is detected from the input device, which turns off the handler. Despite what the manual says, the $5B is not echoed on the screen, although the $5D is echoed twice to make up for it. When a $18 is detected, input will take place from the defined start of the file and will only cease when a $5D is detected. In effect, memory is being used as if it was the cassette recorder in a ROM–based system.

Some simple examples should clarify the use of this feature. We will assume that the file handler has been set up to partition the memory equitably between BASIC and the file. All instructions shown thus <SHIFT K> <RET> mean that the user should press the shift key with letter K, followed by the return key.

1. To delete lines 100 – 199 in a BASIC program type, in the immediate mode:
FOR I = 100 TO 199: PRINT I: NEXT: <SHIFT K> <RET>
Numbers from 100 to 199 will appear in the left hand side of the screen. Then press <SHIFT M> <RET>.
]] and (probably)
SN ERROR
OK will appear – don't worry!
<CTRL X>

The previous numbers will appear; BASIC thinks you are typing them in, and deletes any existing line with a corresponding number. An OK,] and SN ERROR will probably crop up again (BASIC is trying to obey the OK and the graphic as a direct command). However, when you list your program, you will find that the lines in question have all been deleted.

2. To merge PROG 1 and lines 100 to 374 of PROG 2 on disk systems:

```
DISK!"LO PROG 2 <RET> .....(65U: LOAD"PROG 2 <RET>)
LIST 100–374 <SHIFT K> <RET> required lines will list on screen.
<SHIFT M> <RET>
]]
SN ERROR
OK    will appear on screen
DISK! "LO PROG 1 <RET> .....(65U: LOAD"PROG 1 <RET>)
<CTRL X> <RET>
```

Required lines will be listed on screen, followed by:

```
OK
SN ERROR
]
SN ERROR
OK
```

or something like that!

LIST should verify that the two programs have been merged.

Users of 65D who have written their random access file program but have forgotton to define the buffer first can easily LIST their program to the indirect file, run the change utility to define the buffer, and then download the program so that it is re–located above the buffer, then save the total on disk.

3. To transfer a ROM BASIC program, stored on disk in 65D, into the ROM BASIC operating area, make sure that both handlers address the same start point. Then, in the disk system, load the program and list it into the indirect file. Exit from BASIC and, when in the Kernel, CALL the ROM handler into position. BREAK and cold start ROM BASIC, presetting memory size so as not to wipe memory. Initialise the handler and type <CTRL X>. The program should now be ready to run. Programs could be uploaded ready to store on disk by the same procedure.

These examples should suffice to show that a lot can be done with this very flexible system, and readers are urged to experiment. Use of memory in/out via the device handlers will now be considered. This part of the discussion pertains to the disk handlers only, but the function can easily be emulated by the ROM–based system as the control codes for the file are trapped on output, so printing the appropriate character will re–direct subsequent input or output as required.

In OS–65D, Device 5 is the memory input and output system. The Kernel command *'MEM nnnn, mmmm'* will direct any output to device 5 to start from *mmmm,* while input will start from *nnnn,* both locations being hexadecimal in form (from BASIC this would be *DISK!"MEM nnnn, mmmm").* In OS–65U the memory input start pointer is at locations 11657, 11658 ($2D89, $2D8A) Lo/Hi byte; the output pointer is similar and is located at 11661, 11662 ($2D8D, $2D8E). Once these pointers have been set up, variables may be printed to the memory block by PRINT#DV, data, or absorbed from it by INPUT#DV, variable where DV = 5 for 65D and 4 for 65U. Once again, an example might make this clear.

4. As part of a suite of programs, it is desired to transfer the values of the variables N and A$(I) between PROG 1 and PROG 2. We will assume that space has been allocated at the top of memory, starting at $7000 for this purpose. Then, the end of PROG 1 could be:

```
1000 DISK!"MEM 7000, 7000": REM SETUP DEVICE 5
1010 PRINT#5, N
1020 FOR I = 1 TO N: PRINT#5, A$ (I): NEXT
1030 RUN" PROG 2"
```

Now, the start of PROG 2 could read as follows:

```
10 DISK!" MEM 7000, 7000"
20 INPUT#5, N
30 FOR I = 1 TO N: INPUT#5, A$ (I): NEXT
```

It should be obvious to readers that this will transfer the values required between the two programs. In this manner, programs can be split up into sections so that one is no longer limited by the amount of RAM on the system. This ability is one of the reasons why disk–based computers can be so much more powerful than cassette–based ones ever could be.

5. The next example is almost trivial but will serve to illustrate the concept of a command file. In this case, the computer uses commands in the immediate mode to change languages. The example to be discussed shows how to get BEXEC* to select the assembler. The unwary may think that a mere 'DISK!" ASM"' would suffice, but unfortunately this is not so. Regretfully the havoc that such an attempt can cause may only show up when attempting to save your file to disk. If we assume that the menu procedure will call line 1000 when the assembler is to be selected, and that the memory is set up as for the above example, the following will generate a simple command file and subsequently execute it.

```
1000 DISK!" MEM 7000, 7000"
1010 PRINT#5, "EXIT": PRINT#5, "ASM": PRINT#5, "!IO 02, 02"
1020 DISK!" IO 10, 02": END
```

It should be clear by now that the indirect file handler is not difficult to use, and can be used to make the system do many interesting things. The following points may help if things go wrong. First, input via the <CTRL X> function will only terminate if a $5D (])is detected in the input stream. Thus, if the file has written beyond existing memory, this character may not be present. If this has happened, the monitor may be used to place this code in the last byte of real memory to attempt to salvage the situation. When merging programs, deleting lines or using other 'programming aids', it may not be necessary to limit BASICs memory as it is not using it. This makes the indirect file really easy to use – your risk! Obviously, memory pointers should only be changed at the beginning of a program, otherwise string data may well be lost or get muddled up with the indirect file data. Conflicting usage of memory or attempting to write beyond the limits of actual memory are the only likely sources of problems with this system. And, finally, remember that the disk PIA comes right after the end of the user RAM, for all those fortunate enough to have 48K machines. Writing to these is an easy accident, but can prove expensive (and how do you think I know, eh?)

It is difficult to decide on a good set of defaults for a given memory configuration, although if this is divided equally between the indirect file and the language few problems should occur. It is worth bearing in mind that the indirect file is in ASCII, includes carriage returns and line feeds, and is thus often a lot longer than it might appear at first sight.

The Modem: key to computer communication

by Richard Elen

In the previous issue of the Newsletter, we saw how a Modem may be used to couple your computer to a telephone line, thus enabling it (and you!) to communicate with the outside world, via such media as the Computer Bulletin Board Systems (CBBS) which exist in a number of places. Although, of course, CBBS originated in the US, we are starting to see them spring up in Great Britain as well (see the article elsewhere in this Newsletter).

But, like many computer-related subjects, we owe a great deal to work done in the United States, and in the field of computer communication, we need to pay special attention to the American standards for modems. These standards, referred to generally as 'the Bell standard' in the previous issue, fall into several main classes, of which the most important to us here are the Bell 103 and 113 standards. Generally speaking, the 103 and 113 types are used for communication up to 300 baud (600 maximum), while faster types (such as the Bell 202, utilised for systems which require higher speeds, commonly 1200 baud), run at rates up to 9600 baud! The 113 type is a single-mode ('originate' or 'answer' - these terms will be described later) 103-type compatible version. Certain high-speed systems, using synchronous transmission, can operate at 2400 baud and above; the majority of systems, however, are asynchronous, like the 103 type which is of greatest interest to us here.

Of course, telephones were designed for voice transmission, and not for data transfer, so the primary purpose of a modem (MODulator-DEModulator) is to convert the digital signals which a computer can produce into analogue audio signals which can be transmitted down a 'phone line, and reconvert them into digital information at the other end. The bandwidth of a telephone line is only about 3kHz, so the signals produced must be capable of being transmitted satisfactorily within this band.

As noted in the previous article, there are a number of communications modes which are possible: *Simplex,* in which only one-way information passing is possible (as in receiving Teletext transmissions on TV); *Half-duplex,* where 2-way communication is possible, but with only one operator 'talking' at one time while the other is 'listening' (CB radio is an example of this); and *Full-duplex,* where both parties can transmit and receive at once (a normal telephone conversation is full-duplex). Generally, when computers operate in full-duplex mode down a telephone line, both machines use the same baud rate; however, two speeds are sometimes used to increase data transfer if the majority of traffic is in one direction. In this case, the lower speed (called the 'reverse channel') is generally used to transmit confirmations and error-checks. However, this mode does not concern us here. Additionally, the term 'Half-duplex' is often used to denote that outgoing data is printed directly at the sending terminal, while 'Full-duplex' refers to a mode in which the distant computer 'echoes' the outputted characters without them being printed locally. This confirms reception by the distant

machine of the correct characters. An OSI serial system uses 'Full-duplex' in this sense between the terminal and the computer, and the terminal routine in the last Newsletter is capable of 'Full-' and 'Half-duplex' modes in this sense. However, this use of the terms does not necessarily refer to the two-way capability of the transmission link in use at the time.

An important aspect of telephone data links is the level at which signals are sent and received. It is important that the signal level is not too high so as to introduce distortion, yet not too low for noise to cause errors. The actual level sent down the line should generally not exceed 0dBm (0.775VRMS or about 2V peak-to-peak), and should not drop below about −12dBm (0.5V p-p). The best compromise is a level between −6 and −9dBm (1V to 0.75V p-p). On reception, of course, the level may well have dropped considerably. On a local call full strength may be received, but very often it may be considerably less. Below −50dBm is generally too low for reasonable data transfer, and many modems refuse to deal with signals below about −40dBm (0.02V p-p). Normal voice communication down a 'phone line would be well-nigh impossible at 40dBm or less.

The modem itself converts the digital data into audio tones, and the exact tones produced vary between the different types of modem. Bell 103 type modems are the most usual, and 300 baud is the most usual speed, higher rates being possible but more subject to error, as users of the standard OSI cassette interface will know well (the cassette interface also converts digital serial signals into tones). The exact tones produced for a logical 'zero' and logical 'one' depend on which mode the modem is operating in. In 'originate' mode, the mode in which a system sending a call will be in, the tones are lower than they are in 'answer' mode, the mode into which the receiving modem will be placed. The Bell 113 modems only operate in one of these modes, a 113A modem operating in 'originate' mode, and a 113B modem operating only in 'answer'. Apart from this, the 103 and 113 standards are compatible. For talking to a CBBS, the 'originate-only' type is sufficient (113A), and many surplus modems are of this type, most of them having come from terminal network-type systems where the terminal is required to 'call up' the central computer. Several hobbyist modems, however, are 'answer-only', and these are of little use. A group of friends who wish to communicate with each other will obviously need full 'originate-answer' types of modems, or communication will only be possible in one direction, or not at all!

Table 1 shows the frequencies used by Bell 103/113 types.

Mode	TX'1'	TX'0'	RX'1'	RX'0'
Originate	1270 Hz	1070 Hz	2225 Hz	2025 Hz
Answer	2225 Hz	2025 Hz	1270 Hz	1070 Hz

To operate successfully at 300 baud, the modem should be designed to cope with at least 400 baud: don't expect the unit to run too close to its operational limit! It should be noticed that the different frequencies used are quite close together: this means that the unit must be capable of switching frequencies quickly and accurately; it also limits the speed. For greater speed, the Bell 202 system is utilised, expanding the difference between '0' and '1' frequencies to 1kHz. This makes full duplex impractical, as there is insufficient bandwidth to carry two channels this far apart. On normal telephone lines, 1200 baud is practical with half-duplex on this system, higher rates being theoretically possible (up to 2000 baud). However, while the modem itself can handle this rate, the telephone lines generally cannot!

Other types of modems exist which can offer even higher speeds, but neither the 202 types nor the faster 201 (2400 baud), 208 (4800 baud) and 209 (9600 baud) systems concern us here, as they are not used by CBBS, and require complex changeover signalling to indicate when the system must change over from send to receive.

A standard Bell 103 modem utilises the normal RS232 pinout for interconnection, using the standard 25-pin 'D-type' connector. The pinout is given in Table 2. However, as always, readers should check the modem they acquire to ensure that it actually works like this.

Table 2 standard pinout for RS232

Pin 1: Chassis.
Pin 2: Data to transmit.
Pin 3: Data to be received.
Pin 4: Request To Send (RTS). May be used for mode switching.
Pin 5: Clear to Send (CTS).
Pin 6: Data Set Ready (DSR). Always high when modem is on.
Pin 7: Logic ground.
Pin 8: Data Carrier Detect (DCD). Either always high, or high when link is established.
Pin 20: Data Terminal Ready (DTR). May usually be ignored.
Pin 21: Signal Quality Detect (SQD). Often not provided. If available, indicated that a bad line is present, and that errors may occur.
Pin 22: Ring Indicator (RI). Generally not provided. If available, is used to switch modem into answer mode when a call is received.
All other pins are unconnected.

Modems are available from a number of sources, mainly American. In all cases, it is worth checking that the unit has the correct modes available, and is of the right type. Remember that the majority of CBBS use Bell 103 type modems at 300 baud. For this application, your unit should at least have 'originate' (Bell 113A) capability, if not both 'originate and 'answer'. Check the small print in advertisements, and avoid units which offer 'answer only' capability, as these will be difficult to use (for 'difficult' read 'impossible'). Check that the unit will operate at *at least* 300 baud. Unless you have some personal high-speed experiments you wish to carry out, avoid other than 103/113 type modems.

If you want to communicate with friends as well as with CBBS, you need a full 103 originate-answer unit. Dave Graham of Mutek informs us that he will shortly be marketing such a device at a very reasonable price (£50 or less), so it is hardly worth building your own. However, if construction takes your fancy, there are a number of modem chips available which will handle the tone conversion for you and will tack on to either the ACIA or direct on to the bus. Once again, check the facilities and modes available as well as the standards. If desired, it should be possible to 'double up' the cassette serial port with a modem system so that you can select which output you require. Do remember, however, if you build your own unit, that only PO-approved modems should be attached to UK phone lines: you may cause damage to the lines, and/or worse still, damage to your relationship with your local Telephone Manager. If in doubt, use acoustic coupling. Although this degrades the signal a little, it's safer that way. And at 300 baud, very little more can go wrong than can go wrong with your standard cassette interface.

Forum-80 computer bulletin board

Frederick Brown

Forum-80 (the first in the UK) is a computerised bulletin board which is open for public use.

Messages and notices can be posted and retrieved over the telephone using a personal computer (or a mainframe if that's all you've got) and a modem.

The system is oriented to the needs and interests of computer hobbyists. Typical messages include: for sale, wanted, personal, hints, plus a library of programs for you to download to use on your own system.

Computer bulletin boards

One of the fastest-growing uses of microcomputers these days is for exchanging information between computers over the phone.

Private companies — for a fee — will let you access their computers to run programs, games and the like, or find a wealth of information, ranging from the latest stock-market prices to the latest news from the *New York Times*.

There is an alternative to such national enterprises, and one is up and running in Hull with another to start in the Midlands soon.

Computer bulletin boards — also known as electronic message systems — supported by local computer groups, retailers and individual hobbyists are simple systems permitting the users to enter and retrieve messages or information.

There are several software packages available for running bulletin boards in the States. One of the most useful is Forum-80, written by Bill Abney who has organised a network of operators and standardised the sytems. The latest improvement to Forum-80 is the downloading of programs, both BASIC and machine-code, providing the caller has a suitable program for receiving downloads.

Although every bulletin board has its own personality, as you will find out, there is total standardisation of serial word length, parity, stop bits and baud rates. This means that you can program your RS232 port and communicate with nearly every bulletin board.

What do you need?

To access a bulletin-board system you need a personal computer or terminal with RS232 installed, a modem and a software program. The programs must organise the communication between the keyboard, CRT and the serial port.

This program can be written in BASIC, although it is far more efficient to write a machine-language program to do the job. The concept is simple, and you should be able to write a similar program for your system.

The idea is to constantly scan the serial port and the keyboard for an input from

the outside world. If a byte is received by the serial port, then it is displayed on the CRT or printed.

Once you have written or purchased a terminal program, you are ready to call a bulletin board. With the serial port set up as described in the following table, pick up the telephone and call the nearest bulletin-board system. When the other end answers, you should hear a tone on the line.

baud = 300
word = 7
parity = even
stop bit = 1

Set your modem on 'originate' as described in its instruction manual. If all goes well, some sort of message will appear on your CRT or printer. If not, press your carriage-return key several times until the distant system responds.

The bulletin-board system may ask you some mysterious questions like DO YOU NEED LINEFEED? and DO YOU NEED NULLS? (0 TO 50). Answer N to the linefeed question.

If you are using a printer slower than 300 baud, answer the NULLS question with an appropriate number between 1 and 50. If the number you enter is too small, the first characters in each line will not print because the head of your printer is in the process of a carriage return while those characters are being sent.

By entering a large enough number to the NULLS question, you force the bulletin-board system to send null bytes ($00) while your printer is executing a return.

Using the bulletin board

The bulletin board now prints a greeting and asks you your name. Use your given name; it's important. Many of the bulletin boards scan the name inputted and compare it to the TO field of messages in the system. The board indicates to the person signing on that there is a message for him awaiting retrieval. Signing on as KLEMCADIDDLEHOPPER defeats the whole purpose of bulletin-board systems. Once you tell the system who you are, you can access all its various functions.

If after calling the system you get lost, press your RETURN key several times to get back to the FUNCTION mode. Then enter an H for Help!' This function explains how to use the system. If you are totally lost, enter a T to exit (while in the FUNCTION mode), and the bulletin-board system will disconnect.

Do not hang up on a bulletin-board system! Although the software is, in many cases, written to handle the user who just hangs up, hanging up could cause a system crash, putting it out of service for many other users.

Conclusion

I would like to thank Bill Abney for all the information and help in starting Forum-80 in the UK.

During the last few weeks all the Forum-80 bulletin boards have been updated and many new features have been added. If anybody is interested in running a Forum-80 bulletin board or would like more information on using a bulletin board, please contact me:

Frederick Brown
421 Endike Lane
Hull HU6 8AG tel (0482) 859169

The bulletin board is available on the above number on Tuesday and Thursday, 7.00pm to 10.00pm; Saturday and Sunday 12 noon to 10.00pm.

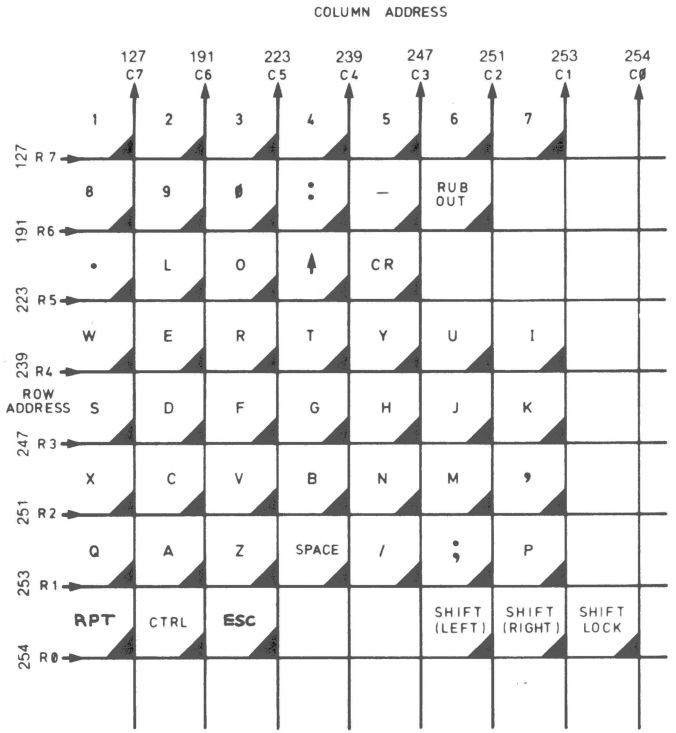
HARDWARE UPDATE

Extra Keys for the UK101

We have already pointed out the fact that 'single-key' BASIC commands can be entered with CEGMON by use of the REPEAT key. This key allows access to a large number of 'higher bit set' graphics characters, some of which are interpreted as BASIC tokens by the interpreter. Unfortunately, The UK101 does not have a REPEAT key, so this function is normally unavailable to UK101 CEGMON users, even though the routines are there in the ROM.

In addition, the UK101 does not have an ESC key. On OSI systems, CEGMON uses the ESC key to copy characters into the new line under the Editor; in the UK101 version Control Q is used instead. However, some printers can be controlled with ESC sequences (ie ESC followed by another key), for example to switch the printer on and off-line.

Both these keys are quite easy to add. The normal position for them on the keyboard is with ESC to the left of the Q key, and REPEAT replacing the left-hand RESET key (the Superboard, for example, has only one RESET key, while the 101 uses two in series to prevent accidental reset). However, you may place them wherever is convenient. Having purchased suitable key switches and keytop, the



REPEAT key is added by attaching one contact to the line which connects the Q and W keys, and the other to the line which connects the two Shift keys together. The ESC key is added by connecting one contact to the line which connects V to F or Z, and the other to the line between the Shift keys as above. The diagram shows where these keys fit into the keyboard matrix.
UK 101 CEGMON users will know that the 'up arrow key becomes Linefeed, 'up arrow' being accessed by Shift-N, as on other machines. Rather than buying an 'LF' keytop, turning the 'up arrow' keytop upside down (producing 'down-arrow') does the job admirably!

BASIC Notes

Many users, like myself, may have been frustrated by the lack, in OSI's BASICs, of the IF ... THEN ... ELSE construct which can help one to write more structured programs. An interesting way of synthesising this type of statement uses Boolean variables.

```
10 X= (A>B)
```

If the above statement is run, X will adopt a value of 0 if A<=B, and –1 if A>B. Using this, an IF ... THEN ... ELSE construct is easily made up as follows:

```
10 X=(A>B): REM ..... Condition
20 ON X+2 GOTO 50, 30
30 REM ..... "ELSE" mode – condition failed
40 REM ..... "ELSE" mode – conditioned failed
50 REM ..... "THEN" mode – A was greater than B
```

GOSUBs could, of course be used instead of GOTOs. If the "ELSE" procedure is placed directly after the ON...GOTO statement, line 20 can become

```
20 ON X+2 GOTO50
```

with no problems as ON...GOTO will "fall through" under these conditions. Another advantage of using this method is that the conditional operations are not restricted to the remainder of a BASIC line, itself limited to 72 characters in length.

A similar system could be used to simulate other structured programming techniques such as the WHILE *(condition)* DO *(procedure)* or REPEAT *(procedure)* UNTIL *(condition)*.

The difficulty in implementing these using FOR...NEXT loops is that a FOR...NEXT loop is always executed at least once. However, the following will never be executed if the condition is not met

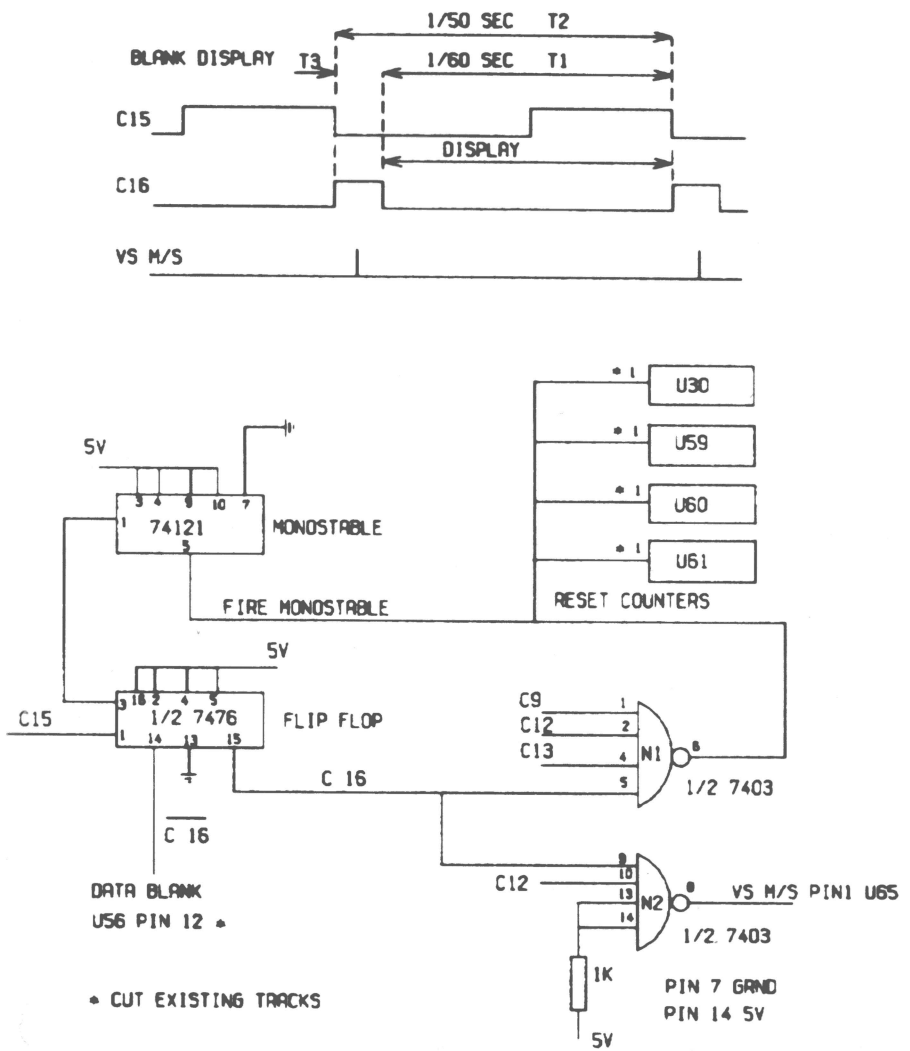
```
10 X=(A>B)
20 ON X+2 GOSUB 100, 120 :REM WHILE (A>B) DO (sub 100)
30 REM ... Condition failed, continue program
.
.
.
100 :     REM ... (sub 100) BEGIN
110 :     PRINT "A>B"
120 :     RETURN: REM ... END
```

This illustrates the use of ON...GOSUB. Note that the fail condition also calls a subroutine, so it must be pointed at a RETURN instruction at the minimum (any handy one will do!). You may also note the technique used to indent BASIC lines.

50Hz Superboard display

J.C. Harris

This circuit was originally lashed up on a breadboard, and when it was working properly I built it on to the prototype area of the computer. On looking at it again it looks as if it might be possible to omit the 74121 and achieve the same result. But at least as it is shown I can say that it gives a steady display of over 28 lines



without any circuit layout problems. Processor operation was not affected by resetting the counters.

Flip-flop divides C15 to "C16". N1 gives a reset pulse to counters U30, U59, U60, U61 and C16 after time T3, thereby extending VS to 1/50 second (actually 1/50.2 second). The display is blanked in T3. N2 gives a pulse to fire U65 (VS monostable) at a time such that the display is in the centre of the scan.

Some modifications to the Superboard are required. Isolation of pin 1 of the counters is not straightforward, and to cut the track to pin 12 of U56 I had to unsolder the chip. (This could possibly be omitted since the display in T3 would probably be in the TV overscan area.)

REVIEW

Magnum EPROM board

reviewed by J.M. Leach

Magnum Electronics, at 3 New Inns Lane, Rubery, Birmingham, have introduced a series of boards for the OSI Superboard/UK101 which plug into the 40-pin extension socket. Their range includes a 6-socket motherboard, an 8K RAM card, link selected from $2000 to $7FFF, and an 8K EPROM board, with the Extended Monitor on an EPROM thrown in as a package at a bargain price of £50.

We decided that this board would be a useful extra for the UK101, so after a telephone call to find out about the address decoding, we posted off a cheque. The board kit arrived by return of post (an almost unique occurrence!) and after a couple of hours with the soldering iron and the clear and explicit instructions I plugged in the board, connected by a generous two feet of 40-way ribbon cable, and found that it worked first time. So now we have the Extended Monitor permanently on tap, with a further 6K EPROM available for extra goodies, such as our auto line-number routine, the *AY-3-8910* music chip driver etc.

Magnum have made some changes to the Extended Monitor, which now starts at $9800. The memory-change command (was @) is now U, with / to increment and – to decrement memory, ie a one-finger key input. / also steps the disassembler, which is a bit of a nuisance when printing a list as the / appears at the end of each 13 lines. A <CONTROL-J> (linefeed on the UK101) would have been useful as an alternative. In the U mode ';' gives the ASCII equivalent of the current byte. Otherwise the commands are the same as the original version, apart from J and @. The J command gets ready for a return to BASIC, followed by a warmstart, while @ goes direct to the resident machine-code monitor at 0000. The <RUBOUT> key deletes the characters on the screen, but does not appear to affect the input buffer. If a mistake is made it is wise to abort with a <CR> and start again!

It is really helpful to have this facility permanently available, and not sitting right in the middle of the UK101's 8K memory. Now, on with a new EPROM with all the useful routines that are too tedious to load up each time.

Save your Keyboard with a Cheap Joystick

John Partridge

I have found that the Two-Axis Joystick sold by Maplin (HQ50E at £2.95) is suitable for connecting to the keyboards of C1's and C2's for games playing.

Although the joystick uses two potentiometers, these are short travel, and have a low enough resistance at both ends to act as switches.

I use the R0 line of the keyboard connected to the wiper of the pots, and the other pot terminals connected to C1 to C4. A fire button connected from R0 to C5 completes the arrangement.

The position of the joystick can be found with a PEEK (57100), which will return 3, 5, 9, or 17 with 33 from the fire button.

A suitable case is the R.S. Components *508-914,* which has enough internal depth to take the joystick, but it is a convenient size for holding in the hand.

EPROM MEMORY BOARD AND PROGRAMMER

As advertised in the last but one issue of OSI User Group Newsletter, the MCS/A2 EPROM Memory Board and Programmer is available now in kit form from M.C.S. Electronics, 9 Willowfields, Hilton, Derbys. DE6 5GU.

This kit offers you a High quality, fully buffered memory board with 8 × 24 pin EPROM sockets, user defined addressing and also an EPROM PROGRAMMER for 2K, 4K, 8K and 16K single supply rail EPROMs, all on one PCB. Kit has been designed to plug directly into your Superboard or Compukit UK 101 via 40 way ribbon cable.

Kit comprises PCB sockets and all components including programmer power supply, control software on tape and comprehensive manual. Price, excluding EPROMs, £59.95p.

Optional extras are as follows:- 12" 40 way ribbon cable (with plugs) £4.95p, Textool ZIF socket £6.90p, 16 pin DIL switch (to select EPROM type) £2.00p. Don't forget we also supply Exmon in EPROM £6.50, and Assembler in 3 EPROMs £20.00p. Please include your original tapes with order. There is no VAT to pay.

UK101–SUPERBOARD EXPANSION

A full range of integrated enhancements now available

ROMDOS

ROMDOS has been commissioned and written specifically for the PREMIER UK101/OHIO Disk System. It is principally aimed at the user with a small capacity RAM machine, but is also extremely useful for the larger RAM machine user since it allows BASIC programs to run with disk with little or no alteration. ROMDOS links the standard BASIC-in-ROM with a disk controller program so no RAM memory is used for the BASIC interpreter and under 4K for ROMDOS, giving an 8K saving in memory over the normal OS-65D system. The BASIC-in-ROM continues to work at its normal high speed and is enhanced by a wide range of disk commands. The system is compatible with all standard Premier EPROM upgrades such as BASIC 4, BASIC 5, and TOOLKIT 2. ROMDOS comes as a two disk set with complete documentation. PRICE £17.95

FLOPPY DISK CARD

UK101/OHIO. The Premier F.D.C features:- Integral Data Separator or link-selectable for on-drive separator if required. Supports 4 × single-sided 5.25 or 8in drives or 2 × double-sided 5.25 or 8in drives. 1 or 2MHz operation (DOS permitting). Interrupt linkable if required. Padding for future options. Shugart Bus as supplied; linkable to other Bus requirements providing signal compatibility is maintained. OSI system compatible (software and hardware). Drives available early March.
PRICE Delivery - March.

SPECIAL OFFERS

TOOLKIT 2 + MINI EPROM BOARD	£29.95
BASIC 5 + MINI EPROM BOARD	£29.95
CODEKIT + MINI EPROM BOARD	£29.95
WORD WIZARD + MINI EPROM BOARD	£29.95
SOUND/VIA – Base, Sound and Via kits	£43.95

CEGMON

CEGMON is Premier's standard monitor for the UK101/OHIO range. Expansion to disk becomes a reality once CEGMON is installed.

* DISK BOOTSTRAP !
* full BASIC editor: delete, copy, concatenate. Auto repeat on all functions.
* powerful, sophisticated screen management system.
* user-defined windows, non-scrolling areas
machine code monitor, tabular display, memory move, M/C SAVE
* keyboard gives true rubout, typewriter response
* BASIC & Assembler vectors in RAM
* comprehensive manual includes entry points
* Extensive software available
* PRICE £25.87 State machine when ordering

SCREEN ENHANCEMENT KIT

This kit offers 20 software selectable screen formats for the UK101/OHIO, including a true 32 × 64 format. It plugs directly into the main board (OHIOs need sockets inserting) and provides almost every available screen format for ultimate software compatiblity.
PRICES KIT £55.95 BUILT £69.95. (+£2.00 P&P).
Fitting sevice available.

SOUND/V.I.A BOARD

The TES II VIA/SOUND kit gives you up to 56 Input/Output lines and programmable sound generation. In order to allow total flexibility, we are offering the kit in low-cost packs. The Base Kit consists of PCB, connector, address decoding and buffering, plus IC sockets. The Sound Pack consists of AY-3-8910 sound chip, amplifier and components. The VIA Pack consists of VIA and support.
BASE KIT £24.95 SOUND £11.95 VIA £9.95

BASICS for UK101 and OHIO

adds 17 new BASIC words to your interpreter which can be used in program lines and gives machine-code response speed. HLIN, VLIN, SCR, BLK, SET and TEST allow high speed generation and manipulation of graphics. PRINTUSING, PRINTAT, INAT allow total control of screen input/output. GET (key), RD (Read DATA), GS and GT (GOSUB and GO TO a variable), GO and GO$ (GOTO a machine code routine), allow total program flexibility. WI and CWI allows CEGMON screen manipulation. BASIC 5 is available for CEGMON, MON02 and SYNMON/MON01 only. State precisely your computer and monitor when ordering. Comes complete with comprehensive manual.
Available on DISK or in EPROM (9000hex) £19.95

BASIC 4 cassette file handling system

This new EPROM for the UK101/OHIO provides a comprehensive file-handling system, capable of working at up to 4800 baud.

* named programs to cassette
* verify tape contents facility
* reliable high speed SAVE/LOAD
* selectable auto/run of loaded BASIC program
* superb crash recovery command (OLD)
* original SAVE/LOAD commands unaltered
* reduces LOAD/SAVE times by up to 40%
* seven new SAVE/LOAD commands
* non-destructive memory test
* initialises BASIC 5 automatically if resident BASIC 4 is a plug-in replacement for your existing BASIC 4 ROM. Not suitable for MON01 or Synmon monitors. PRICE £11.95

TOOLKIT 2 for UK101/OHIO

The most powerful TOOLKIT on the market, TOOLKIT 2 gives the following facilities in only ONE EPROM.
REPL exceptionally powerful Global Search and Replace of BASIC listings. DUPL copy a line into a new line. LIST/ controlled listing of program. FIND anything in a BASIC listing. RENUM renumber from any start in any increment – full error messages, totally reliable. AUTO any start, any increment. DELETE high speed block line delete. VIEW cassette dump verification. TRACE superb trace feature – screen transparent. MC enter the monitor quickly! TOOLKIT 2 also lists on error and cures the warm start 'OM ERROR' bug. Available in EPROM only (80000hex), for CEGMON. MON01 & 2, and SYNMON monitors (DISK soon).
PRICE £19.95. State machine & monitor.

INVADERS

Quite simply the best machine code game ever written for the UK101/OHIO. PREMIER have succeeded where others have failed. Our INVADERS is faster than any version we have yet seen, including Arcade machines. INVADERS has all the features you expect, plus superb graphics and two-player option. AVAILABLE for CIE and UK101. PRICE £7.95.
Also now available for the 32×48 CEGMON based UK101 (with new BASIC 1) is KAMIKAZE INVADERS – a new slant on this popular game. £5.95.

POSTAGE & PACKING Software 70p per order, EPROMS/DISKS 90p per order, HARDWARE £1.50 per item. Maximum £3.00. **ALL PRICES INCLUDE V.A.T.**

BARCLAYCARD VISA

Premier Publications

208 Croydon Rd, Anerley, London SE 20 7YX. Tel: 01-659 7131

BACK PAGE PROGRAM

```
10 REM: 'Midnight Hacker' by S A Smith, 1981
100 VI=53248+37+64:CU=53608:T$(1)="?SN ERROR"
110 T$(2)="?OM ERROR" : T$(3)="NO CHANCE!"
120 PRINTCHR$(26)"To the Midnight Hackers :"
130 FOR L=1 TO 14
140 :  FOR C=1 TO 18
150 :     READ N : IF N=0 THEN N=32
160 :     IF N=1 THEN N=187
170 :     POKE VI+L*64+C,N
180 :  NEXT C
190 NEXT L : POKE CU,95
200 FOR TEXT=1 TO 3
210 :  T$="RUN":GOSUB 280:FORDE=1 TO 2000:NEXT
220 :  T$=T$(TEXT):GOSUB 280
230 :  T$="OK":GOSUB280:FOR DE=1 TO 1000:NEXT
240 :  CH=161:GOSUB 360:CH=32:GOSUB 360
250 :  FOR L=53419 TO 53426:POKE L,148:NEXT
260 NEXT TEXT
270 GOTO 200
280 FOR CH=1 TO LEN(T$)
290 :  POKE CU+CH-1,ASC(MID$(T$,CH,1))
300 NEXT CH
310 FOR C=0 TO 9
320 :  A=PEEK(CU+C-64):POKE CU+C-128,A
330 :  A=PEEK(CU+C)   :POKE CU+C-64 ,A
340 :  POKE CU+C,32
350 NEXT C : POKE CU,95 : RETURN
360 FOR R=0 TO 5
370 :  FOR O=0 TO 1 STEP 0.2
380 :     POKE 53614-R*64-R*O,CH
390 :     POKE 53615-R*64+R*O,CH
400 :  NEXT O
410 NEXT R : RETURN
420 DATA ,221,148,148,148,148,148,148,148
430 DATA 148,148,148,148,217,148,148,222,,
440 DATA 140,,,,,,,,,,,149,173,,139,,,140,,
450 DATA ,,,,,,,,149,173,,139,,,140,,,,,,,,
460 DATA ,,,149,173,,139,,,140,,,,,221,195
470 DATA 135,135,197,222,,149,173,,139,,,140
480 DATA ,,,,202,,,,,199,,149,,,139,,,140,,,
490 DATA 224,136,,,,,143,225,149,227,228,139
500 DATA ,,220,148,158,148,148,200,,,,,201
510 DATA 148,215,158,148,223,,210,135,135
520 DATA 135,135,135,1,1,1,1,1,1,135,135,135
530 DATA 135,135,207,209,128,128,176,161,1,1
540 DATA 1,1,1,1,1,1,161,178,128,128,208,136
550 DATA ,,1,1,1,1,1,1,1,1,1,1,1,1,,,143,209
560 DATA 154,161,1,1,1,1,1,1,1,1,1,1,1,1,161
570 DATA 154,208,209,128,128,1,185,1,1,1,1,1
580 DATA 1,1,1,186,1,128,128,208,,,,1,185,1
590 DATA 1,1,1,1,1,1,1,186,1,,,
OK
```