OSI/UK User Group Newsletter

Vol.2 No.5 August/September 1981

Communications

Modem communications for your system

The BASICs of machine code

Superboard 32×64 display: the full story

Get your system on the BUS...

...the IEEE 488 to be exact!

Our **680 expansion board** connects directly to all Ohio Scientific based computers (including the UK101) and provides the following facilities:

Centronics compatible output port
IEEE compatible I/O port
2K of battery backed non-volatile RAM

Software supplied with the board provides drivers for the two output ports. When used with the OSI Superboard the 680 board provides a very powerful IEEE controller offering approximately *5 times higher execution speed*, more built-in I/O capability and a lower price than an equivalent minimum system from Commodore or Apple Corporation.

Price: **£165** complete with documentation, guarantee and all cables and connections.

MUPROMPT

EPROM/EEPROM programmer

Can your programmer load data from a 2716, a 2532 and a RS232 link and then program the edited results into a 2764? If not then you need **MUPROMPT.** It turns your Superboard, UK101 or PET into a real development tool. The system is idealy suited for use by Universities development labs and all those developing EPROM software. **MUPROMPT** requires a 32K system for operation and will read/verify/write to 2758 (both types) 2516, 2716, 2532, 2732, 2764, 27128 (conforming to Intel standard) EPROMs and 2816 EEPROM.

Price: **£385** complete with firmware, cables and full guarantee.

MUPROMPT is also suitable for small scale production of firmware. For larger requirements, the **MUCOPY** is a stand-alone insrument designed to complement **MUPROMPT** and capable of programming 8 or 16 devices simultaneously from a master. Please call for further details on both units.

MUMON

— a new monitor for OSI serial systems

How many times have you wanted a warm-start on your OSI serial disk system? or machine code facilities? or even a built-in memory test? **MUMON** offers all of these facilities and many others in a single replacement EPROM taking less than 10 minutes to fit. The firmware package includes the EPROM, fitting instructions and a comprehensive manual defining its operation and use.

Price: **£27** ex-stock. Overseas customers please add £3 for Airmail.
All prices quoted exclude VAT.

MUTEK

Quarry Hill
Box Wilts
Tel: Bath [illegible]

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.

Under new management

As you will no doubt know, we have had some difficulty getting the last couple of Newsletters into the postal system, this being the second issue which has been substantially late. Tom Graves gave several of the reasons in the last issue: Tom has had to get his business together and George and I have had to put earning money first as well. Quite simply, under the present economic state of this country, we all have less free time ('free' in more senses than one) to play with. Hopefully, our centralising the User Group in London (apart from production aspects of the Newsletter) will enable Tom to concentrate more on his business, and us to use our limited time more efficiently in dealing with your queries. It is easy to underestimate the time it takes to deal with complex technical problems, particularly by telephone, and while there is just the three of us, it is bound to be difficult. As a result, we would be grateful if technical queries by phone could be kept to a minimum for the time being, unless they *really are* urgent. Letters, of course, should be accompanied by an SAE.

Which brings us to another point. By rights, we should be charging rather more for membership than we do, as our primary function (producing the Newsletter) costs a great deal to perform. In addition, postage and telephone costs are always rising along with everything else in these inflating times. To offset a potentially crippling financial position, we are making more pages of the Newsletter available for advertising, as you will see from this issue. The resulting change of style will, we hope, enable us to avoid increasing subs by such a large amount at the end of the year. We are also examining other means of increasing cost-effectiveness, to help us give you the best service at the least possible cost.

Of course, one way in which we can help you better is if *you* can help *us*! Even for a small bimonthly magazine (and we do more than just produce that) we are heavily understaffed, and we would very much like to hear from members who feel they can help out: people who know a fair amount about certain aspects of OSI equipment and can answer the odd letter, or those who can help a little with administration. We are presently badly overworked! Any offers?

Finally, the existence of this Newsletter (and of the Group, in many ways) relies on the contributions members make to the Newsletter. If you have an interesting comment, routine or discovery, please tell us about it. It is quite disheartening to see little snippets from members appearing in the monthly magazines, with never a word from them to us! If you *do* write the odd item for the commercial magazines, please at least send us a photocopy! The items which appear in the monthlies often need adjustment for other machines, or can be linked with other contributions to produce a better result than a mere note amongst a pageful of items in the monthlies from people who have just discovered POKE 15! We may not be the most accurate (we're not the worst, either!) but we do try, and we're *not* a profit-making group. We exist to provide a service, and the quality of our service doesn't just depend on what *we* do — it depends on all of us. We are not the Group: (we are merely its co-ordinators. *You* and we are the Group, and how good we are depends on how much we are *all* willing to put into it. Without that effort we cease to exist as a User Group. And that would be a pity.

Richard Elen

BASIC NOTES

The SPC, TAB and POS commands in OSI BASIC

Bob Bonser

The SPC command is used in print statements in the following way:

```
PRINT SPC(X)"EXAMPLE"
```

where X may range in value from 0 to 255. It may be thought of as a shorthand way of writing:

A$=" *255 blanks* ": PRINT LEFT$(A$,X); "EXAMPLE"

If X=0 then 256 spaces are printed; otherwise it behaves as expected. If a comma is inserted between the SPC(X) and "EXAMPLE" then "EXAMPLE" will be printed at the start of the next comma spaced field i.e. if X=5 "EXAMPLE" would be printed starting in column 14.

The TAB command is used again in print statements to format the output; for example:

```
PRINT TAB(X)"EXAMPLE"
```

X can take values from 0 to 255. When using this command some thought must be given to the output required as the following examples will show:

```
PRINT TAB(4)"THIS"TAB(6)"THAT"
```

gives as output

```
THISTHAT
```

because the cursor is already in column 8 after printing "THIS" and cannot step backwards.

```
PRINT TAB(12)"FIRST" CHR$(13) "AND THIS SECOND"
```

gives

```
AND THIS SECOND
```

as the output; but if you run the program again after having first typed SAVE then you will see "FIRST" printed and then overwritten again.

The operation of TAB can be altered by using POKE 14,0 which has the following effect. Using the statement PRINT TAB(A)"THIS";: POKE 14,0: PRINT TAB(B)"THAT" —

If A=0 and B=0
```
    THISTHAT
```
If A=4 and B=0
```
THISTHAT
```
If A=4 and B=1
```
THIS THAT
```
If A=0 and B=10
```
THIS          THAT
```

Note the effect of the POKE 14,0 command.

A little used function OSI BASIC is the POS command. It is used in the following way:

```
PRINT "HELLO";: X=POS(0)
```

This will result in X taking the value of 5. If the semicolon was replaced by a comma then X would have the value of 14 and if the semicolon was omitted then X would equal 0. The explanation of this odd result is as follows.

The value of X is the column number where any further printing would start. A use of this command is to right justify output.

```
100 INPUT "3 values ";A,B,C
110 PRINT"EXAMPLE ";A;
120 X=POS(0)-1
130 PRINT CHR$(10);CHR$(13);
140 PRINT TAB(X-LEN(STR$(B)));B;
150 PRINT CHR$(10);CHR$(13);
160 PRINT TAB(X-LEN(STR$(C)));C;
```

Also this command can be used to simulate a move cursor left command:

```
100 INPUT "How many";Q
110 A$="A TEST OF THE POS COMMAND"
120 PRINT A$;
130 X=POS(0): PRINT CHR$(13);
140 FOR Z=1 TO X-Q: PRINT CHR$(11);: NEXT
150 PRINT "*"
```

Hopefully, the above has removed a little of the mystery that has surrounded these commands.

Hidden aspects of BASIC

Jack Pike writes: In case you are under the impression that there is little left in Microsoft BASIC in ROM that you are unaware of, here is a demo program for you. The most important feature of the program is the 'LOAD: INPUT A$' combination for preventing program exit on hitting RETURN in answer to an INPUT call. Among the other things demonstrated is that CLEAR clears the stack as well as all the variables with implications on its use in subroutines, and the comment line 5, which prevents accidental entry to the subroutine. This program can be exitted without using the BREAK key because it inputs some numerical variables as well as strings. An input sequence such as <SPACE, Z, Z, RETURN, RETURN> will exit the program.

```
0 REM DEMO PROG - Jack Pike
1 ::::::::::::::
2 :
3 GOSUB 6 (to input routine)
4 :
5 SUB * test INPUT *
6 LOAD: INPUT"Hit space bar & input A"; A$, A, A(-A*(A<))
7 PRINT "#" A$ " " A CHR$(A) A(.) "#
8 CLEAR: REM clears GOSUB return address
9 GOTO : (start)
```

While on input, I have only recently realised that the input delimiters *comma* and *colon* are different. Colon terminates the input to variables on the line (it also seems to have that effect when it is the first character on the line in immediate mode). This property of : could be useful in data files when comments could be added to the file following the variable value, e.g. 9: Stock item no.

Extra Mathematical Functions for BASIC

The mathematical functions provided in BASIC may seem to have been chosen in a rather arbitrary manner (ATN's main function according to OSI seems to be to get wiped out whenever they want room for a new keyword!). Here, for those who would like to use some of the functions not included but who cannot remember the derivations, are the remainder of the trig and hyperbolic functions. These could be declared in a program by using DEF.

The list below has been adapted from *24 Tested Ready-to-Run Games Programs in Basic* by Ken Tracton (TAB Books). The following functions which are not typical of standard BASIC library functions may be easily implemented by the following formulae:

```
ARCSIN(X)=ATN(X/SQR(X*X+1))
ARCCOS(X)=ATN(X/SQR(X*X+1))+1.5708
ARC SEC(X)=ATN(SQR(X*X−1))+(SGN(X)−1)*1.5708
ARCCSC(X)=ATN(1/SQR(X*X)−1))+(SGN(X)−1)*1.5708
ARC COT(X)=ATN(X)+1.5708
ARC SINH(X)=LOG(X+SQR(X*X+1))
ARC COSH(X)=LOG(X+SQR(X*X−1))
ARC TANH(X)=LOG((1+X)/(1−X))/2
ARC SECH(X)=LOG((SQR(X*X+1)+1)/X)
ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
ARCCOTH(X)=LOG((X+1/(X−1))/2
COT(X)=1/TAN(X)
CSC(X)=1/SIN(X)
SEC(X)=1/COS(X)
COSH(X)=(EXP(X)+EXP(−X))/2
COTH(X)=EXP(−X)/(EXP(X)−EXP(−X))*2+1
CSCH(X)=2/(EXP(X)−EXP(−X))
SECH(X)=2/(EXP(X)+EXP(−X))
SINH(X)=(EXP(X)−EXP(−X))/2
TANH(X)=EXP(−X)/(EXP(X)+EXP(−X))*2+1
```

CALL conversion routine

If you have the new BASIC 1 chip with CALL, or have other need for a program to check the hex equivalent of a value, this subroutine from *Dave Woolcock* does the trick simply. Enter the routine after setting X to the value to be converted; the hex equivalent is returned in X$.

```
15000 LO=X AND 255: HI=INT(X/256): X$="$": X=HI: GOSUB 15010: X=LO
15010 Y=(X AND 240)/16: GOSUB 15020: Y=X AND 15
15020 X$=X$+CHR$(Y+48–(Y>9)*7): RETURN
```

Implementation of BASIC programs in EPROM

Ian Dyson

The standard board microcomputer provides an acceptable range of facilities which is available at switch on. The majority of computer enthusiasts accept the five minute delay which may be required to load a program from cassette, but I suspect that many would be discouraged if they had to load the BASIC interpreter from tape before they could use the machine. Similarly, when the computer is being used in machine code, the enthusiast will probably store and load programs using tape. (Implementation of ExMon on EPROM, though, has proved invaluable).

There is a substantial potential for using board computers for industrial and laboratory applications. With the addition of a VIA, the system can become a cost-effective dedicated controller or data logger; BASIC can be used for number crunching or report writing. For these applications we may be unable to rely on an enthusiastic operator; an industrial controller must have the machine code in ROM with auto-start and as few external switches as possible. If BASIC is to be used where operator interaction is required (Report Writer, Results Sorter etc.), there are advantages if the BASIC program is in EPROM: no skill is required to load the program, there are no load errors and no accidental corruption of the program can occur.

The use of EPROM is well established for machine-code applications; the technique is equally valid for BASIC programs.

The BASIC interpreter of the UK101 and Superboard requires that the addresses which bound the memory reserved for storage of the program and for variable storage when the program is running are defined in such a way that the interpreter can find the succession of instructions and can store variables at real memory locations without writing over the program.

Normally, the program is packed in from the bottom of the user memory, a record of the top of program memory being kept; the memory above the program is used for variable storage when the program is running or when the machine is used in the direct mode.

For our requirements it is fortunate that these two areas of memory can be treated independently by the interpreter. The program storage can be at addresses above that used for variable storage provided the relevant pointers are adjusted.

The relevant pointers are in the zero page of memory:

- **1st line of BASIC pointer ($0079, $007A)**, indicates the address at which the first line of the program starts. Each encoded statement includes the address of the next. As each statement is interpreted, a note of the memory location of the next statement is updated. The system can follow the program without reference to the next pointer which is incremented as the program is written or loaded in order that it may be used to prevent the overwriting of program by variables.
- **start of variables pointer ($007B, $007C)**, indicates the memory address above which can be used for storage of variables without overwriting the program.
- **top of memory pointer ($0085, $0086)** restricts the top of memory address which is available for BASIC and is usually set up at cold start.

The pointers are set up by cold start and NEW as follows:

$0079	01	points to $0301, first line of BASIC
$007A	03	
$007B	04	points to $0304, start of variable space when no program stored
$007C	03	
$0085	*xx*	
$0086	*xx*	points to top of BASIC user memory

$0300	00	start of program null
$0301	00	end of program nulls
$0302	00	

When a program is loaded, the start of variable pointer is changed to an address above that used for program storage.

Using EPROM

To store a program in EPROM, it is convenient to use an address block above the user RAM. It is necessary to change the contents of the first line of BASIC pointer to the new start address; the location below the start address must contain a null. The top of memory pointer has to be changed for writing or loading a program into abnormal locations (otherwise an OM ERROR will occur). The latter change should not be made when RUNning the program.

To write the program, it is necessary to temporarily select a block of RAM at the intended EPROM address. *The program should have been written, tested and saved in advance.* The method of implementation is best indicated by the following example of putting a program into EPROM at address $3800 onwards.

Before starting it is a good idea to fill the block of RAM with FF's in order that unused locations are not programmed when the EPROM is burnt.

The following machine code is entered at the beginning of the block. It changes the first line pointers to $3811, puts the null at $3810 (when setting up the RAM) and then jumps to BASIC warm start.

```
$3800    LDA   #$11      A9  11
$3802    STA   $79       85  79
$3804    LDA   #$38      A9  38
$3806    STA   $7A       85  7A
$3808    LDA   #$00      A9  00
$380A    STA   $3810     8D  10  38
$380D    JMP   $0000     4C  00  00
```

Perform cold start, reset, and enter the monitor. Manually set the top address of the temporary RAM in the pointer $0085 and $0086 (00 and 40 in our example). Set the monitor address to .3800 when pressing G will warm start the system.

After typing NEW RETURN, the program can be loaded normally except that the code is stored at $3811 instead of $0301. The contents of the memory block may be saved as machine code, MOVEd to a more convenient area of RAM and saved as machine code or used directly for ROM burning. After a cold start the computer can be used in BASIC without corrupting the special program. (If the user RAM extends to the special area, the user memory must be restricted in the usual way at cold start).

To test the program:
1. Cold start to restore pointers
2. Reset
3. Start at $3800 in the monitor mode. This changes the first line of BASIC pointer and performs a warm start.
4. On RUN, the program, in its abnormal location, will be followed; the whole of the user RAM above $0304 is available for variable storage.

The program can be burnt into EPROM directly from the temporary RAM or after MOVEing and saving. After a cold start, the computer can be used in BASIC for EPROM burning.

Running the program in EPROM follows a BASIC cold-start and entry through the Monitor as described for testing. The program may also follow a warm start and Monitor entry. It may be interrupted, continued and re-RUN as a normal program but cannot be edited. The EPROM program may be co-resident with a RAM program and the two can be RUN alternately by resetting the pointers at $79/$7A to 01/03 to warm start in RAM or reset and warm start which is followed by monitor entry for the EPROM program. Similarly, several EPROM programs may be implemented and activated by their own machine code entry.

MACHINE - CODE NOTES

CEGMON screen fill

CEGMON, unlike the original monitor, uses a subroutine to clear the screen, which can be called into action very easily. It resides at $FE59–$FE6F and is largely relocatable. If it is moved to start at $0240, for example, (MFE59,FE6F>0240), it may be called from BASIC and used as a screen fill. The character to be used is POKEd into 587, and the routine is called by POKE 11,64: POKE 12,2: X=USR(X). If you have the new BASIC 1 chip, CALL 576 is sufficient to instantly fill the screen. (This and a couple of other items were kindly supplied by the UK101 User Group, 9 Moss Lane, Romford, Essex).

Monitor mods

Some of the new monitors for our machines are less compatible than others. Sometimes changes are minor; sometimes less so. Here are a couple of useful conversions:

Key press routine: SYNMON and Superboard Series 2 CEGMON is POKE 11,0: POKE 12,253: X=USR(X): X=PEEK(531). For other CEGMONs, use PEEK(533) instead of 531. Under WEMON, the routine is POKE 11,52: POKE 12,248: X=USR(X): X=PEEK(531)

CompShop's *Super Space Invaders* needs these mods for CEGMON use: Load as a machine-code program (BREAK M L); after machine code and BASIC sections have loaded, BREAK and Warm Start; in lines 984, 986 and 1450 change 255 to 251; in immediate mode, enter POKE 660,76: POKE 661,232: POKE 662,28; then RUN.

We would be grateful to hear of other conversions to common routines and programs so that they run under the various available monitors.

Extending CLEAR

Jack Pike

After the discussion of CLEAR in earlier Newsletters I had a go at defining an improved CLEAR. I wrote a modification to the BASIC $BC routine to add an optional parameter to CLEAR, such that CLEAR(*N*) would release *N* bytes from the top of BASIC. Breaking into the $BC routine indeed provides a route for an extended BASIC. Something more than 8K should be set aside as the recommended location for languages. A 16K slot is probably needed. I was pleased to see that you are thinking about the problem of 'where to put things'.

The syntax for the extended command is CLEAR(*N*), where *N* is an expression giving the number of bytes to be 'freed' at the top of RAM (–32768 ≤ *N* ≤ 32768).

```
D0 02        BNE  +2            ; Displaced BASIC operating code
E6 C4        INC  $C4           ; from $BE to $C1
8A           TXA                ; store X
48           PHA                ; on stack
A2 00        LDX  #$00
A1 C3        LDA  ($C3, $C4)    ; get current BASIC character
```
Test previous and current BASIC character for CLEAR token and '('.
```
A6 E2        LDX  $E2           ; previous character
85 E2        STA  $E2           ; current character
E0 9A        CPX  #$9A          ; is previous character CLEAR?
D0 1B        BNE  +27           ; no, then return
C9 28        CMP  #$28          ; is current character '('?
D0 17        BNE  +23           ; no, then return
```
Get variable value following CLEAR(as 15-bit signed integer in $AE (hi) and $AF (lo).
```
20 F5 AB     JSR  $ABF5         ; get value of expression
20 05 AE     JSR  $AE05         ; put value in $AE (hi), $AF (lo)
```
Change pointer to top of BASIC RAM at ($85, $86) and string pointer at ($81, $82).
```
38           SEC
A5 85        LDA  $85           ; current top of BASIC (lo)
E5 AF        SBC  $AF           ; new top of BASIC
85 85        STA  $85           ; amend $85 and
85 81        STA  $81           ; bottom of string space
A5 86        LDA  $86           ; repeat for pointer (hi)
E5 AE        SBC  $AE
85 86        STA  $86
85 82        STA  $82
```
Tidy up and return.
```
68           PLA                ; replace X
AA           TAX                ; from stack
60           RTS                ; and return
```
This routine is fully relocatable. Locations $BE to $C1 in the $BC routine should be replaced with *20 xx xx EA*, where the missing address points to the beginning of the routine above.

GETKEY routine for the UK 101

John Leach

The UK 101 lacks a GET command, as found in the PET. It is possible to overcome this problem in BASIC by a clumsy series of POKEs to the keyboard memory location, but decoding for any possible key is quite a problem, and takes so long that it is easy to miss key entry by nimble fingers; a serious disadvantage is that CTRL-C has to be disabled.

This short Machine Code routine, written for the New Monitor (now the standard monitor) allows the user to have complete control, CEGMONers and WEMONers will have to find their own solutions.

```
10 REM GETKEY Machine Code routine
20 FOR I=592 TO 619: READ Z: POKE 1, Z: NEXT I
30 DATA 173,79,2,240,5,32,231,249,208,4,141,19,2,96,32
40 DATA 0,253,169,0,141,79,2,169,1,141,20,2,96
50 REM LOAD, RUN and type NEW (protected from Cold Start once loaded)
60 REM
70 REM Demonstration program
100 POKE 11,80: POKE 12,2: POKE 591,1: REM Startup
110 X=USR(X): Z=PEEK(531): IF Z<>0 GOTO 130
120 PRINT "KEY NOT PRESSED": GOTO 110
130 PRINT CHR$(Z): POKE 591,1: GOTO 110
```

The flag at 591 allows user control of the result of key pressing. If 591=1, the USR routine returns 0 at 531 if no key is pressed, but if a key is pressed, 531 contains the ASCII value of the key, and flag 591 is set to 0 before the routine returns to BASIC. If 591=0, the USR routine returns 0 in 531, whether a key is pressed or not.

The use of the 591 flag allows the programmer to do something with the character entered, and after that the keyboard is dead until a POKE 591,1 is encountered. This prevents keybounce, and multiple entry of keys, unless this is wanted. Note that there is no need to disable CTRL-C. Normal INPUT is not affected.

The Machine Code routine uses New Monitor subroutines as shown.

```
0250   AD4F02   LDA   $024F    ; Test 591 flag
0253   F005     BEQ   $025A    ; If zero, bypass Keyboard entry
0255   20E7F9   JSR   $F9E7    ; Test for Key presses (Monitor)
0258   D004     BNE   $025E    ; If pressed, go and decode it
025A   8D1302   STA   $0213    ; Set 531 to zero (0 in accumulator)
025D   60       RTS            ; and return
025E   2000FD   JSR   $FD00    ; Keyboard input routine (Monitor)
0261   A900     LDA   #$00
0263   8D4F02   STA   $024F    ; Zero 591 flag
0266   A901     LDA   #$01
0268   8D1402   STA   $0214    ; Spoil $FD00 comparison with $0213
026B   60       RTS            ; on next call (otherwise $FD00 waits for entry)
```

DISK NOTES

14 Character names for OS65D.
A very useful hint from OS65D users in America points out that it is possible (and quite easy) to alter the DOS so that 14 character names can be used. To do this, make the following changes to the DOS.

Address	find	change to
$2DE1	$07	$0F
$2DE3	$06	$OE
$2DF1	$F8	$F0
$2DF4	$08	$10

[illegible]
PUT DIR ON TRACK 9 4 secs 2DC4 = Ø9 2E07:04

Apart from these modifications, the various utilities will have to be rewritten to cope with the new directory format and a BEXEC* program should be ready to put on the altered disk. Obviously it will not be possible to have as many entries in the directory without more extensive changes to the DOS. It should be possible to develop this idea further so as to have a code letter giving the type of file (Assembler, Basic, Text etc.) and maybe even the date. If anyone follows this through we would like to hear about it for a follow up article, especially if it included details of a similar enhancement to 65U

CURRENT VERSIONS 65D

Most mini-floppy users have been supplied with 65D version 3.1. This version of the DOS will only work successfully at 1MHz and is not at all easy to convert. Version 3.2, the NMHz version, is available, but buyers should be aware that N means 1, 2 or 3.3; 1.5MHz operation is not catered for, although it may be possible to alter 3.2 successfully. However, all speed freaks should take note of the fact that the 610 board disk interface derives its clock frequency for the ACIAs from the processor clock, and thus alterations will have to be made here to achieve standard data rates onto the disk. Does anybody know where 3.2 sets up the NMHz variable at $267B?

Cassette and printer problem with a disk-based C1E under OS-65D having fitted CEGMON
One reader writes: We operate a 32K C1E with single disk drive for school office use, but fitted CEGMON as the machine is also used to teach programming to young children (10–14yrs), CEGMON's editing facilities being invaluable here.

However, if you fit CEGMON, you then lose the use of the cassette and printer port when under OS-65D, as the fitting of a C1E version of CEGMON relocates this port to $F000, whereas OS-65D still thinks it is at $FC00, because that is where the C2 monitor in SYNMON (used in the C1E) places it, and a C1E uses C2-style disks.

There are two possible solutions:

- Fit a four-pole, 2-way switch to re-address the ACIA correctly when in OS-65D (i.e. to reverse the modification necessary when fitting CEGMON to the C1E).
- Try to modify the OS65D operating system to relocate the cassette port from $FC00 to $F000.

It was this latter solution I chose to adopt, and I eventually came up with the discovery that five memory locations, loaded from track-0, have to have their contents changed from $FC to $F0. They are: $24D0, $24D8, $24DC, $24f8 and $2501.

Alteration of these locations requires that we:

- Read track-0 into memory, using a convenient start location: e.g. $4200 (it normally loads to $2200)
- Using the extended monitor (or indeed the CEGMON monitor) change the contents of the appropriate five locations from $FC to $F0
- Write the modified system back onto track-0 (preferably onto a new disk in case of errors!)

The following procedure carries out the necessary modifications:

1) Boot disk to be modified
2) Enter kernel (type EXIT from BASIC)
3) EM (loads Extended Monitor. RETURN needs to be hit after every line unless stated)
4) !CA 0200=13,1
5) !GO 0200 (this gives menu for copier or track-0 R/W)
6) 2 (select track-0 R/W)
7) R 4200 (load track-0 to $4200, i.e. with an offset of $2000)
8) E
9) RE EM
10) @44D0
11) 44D0/FC F0
12) @44D8
13) 44D8/FC F0
14) @44DC
15) 44DC/FC F0
16) @44F8
17) 44F8/FC F0
18) @4501
19) 4501/FC F0
20) !GO 0200 (Menu for copier or track-0 R/W)
21) 2 (select track-0 R/W)
22) W4200/2200,8 (writes system back to track-0)
23) E

And the job is done!

The BASICs of machine-code *Part 6:* *Tom Graves*

Before we start, we ought to tidy up a few errors that have crept in during this series. The "Binary Beans" routine in Part 4 seems to have suffered the most; our keyboard op didn't make quite so much of a mess of it as of the renumberer in that issue, but some errors came through. These are listed in the "Glitches" section of this issue.

In the last section, some people were confused by my using both S (sign) and N (negative) for the sign bit, bit 7; the former is Leventhal's term, the latter is Zaks', and I'll stick to the 'N' version from now on. Also in the last part, two more mistakes, one a keyboard error, the other a blunder on my part, I'm sorry to say. The keyboard mistake was in lines 40 and 50 of the subroutine series — it should have read ... C=1 (new line) 50 GOTO 100 ... not C=150 GOTO 100! The blunder was that I have the carry value the wrong way up in the subtraction operation in line 70: C should be 1 to start, not 0; so the line should read:

```
70 C=1: IF RES<0 THEN C=0
```

The subtraction routine is complicated in that it uses the *inverse* of the carry rather than the carry itself.

In any case, now that we have the main processor flags under control, we can look at the opcodes themselves, and simulate them by BASIC statements. As with last issue, we will leave the problem of the variety of addressing modes aside for the moment, and assume that — as happens with the 6502 itself — the addressing is sorted out separately, leaving the contents of the 'effective address' in a variable called MEM. The disentangling of that effective address will be dealt with in the next part of this series.

We made a start last time by defining the entry point to the 'flags' subroutine for most of the opcodes that affect them. We also need to make sure that all values stay within the limits of an eight-bit word, from 0 to 255; OSI's BASIC should be capable of maintaining all values as integers, but if problems arise this will also need to be checked with an INT statement.

```
  5 REM — V flag test
 10 V=0: IF (RES AND 64) THEN V=1
 20 RETURN
 25 REM — V and C+ update
 30 GOSUB 10
 35 REM — C+ only update
 40 C=0: IF RES>255 THEN C+1
 50 GOTO 100
 55 REM — V and C– update for subtract
 60 GOSUB 10
 65 REM — C– only update for compares
 70 C=1: IF RES<0 THEN C=0
 80 GOTO 100
 85 REM — C/ update for LSR, ROR divides
 90 C=0: IF (RES–INT(RES))<>0 THEN C=1
 95 REM — general N and Z update for most operations
100 Z=1: IF RES<>0 THEN Z=0
110 N=0: IF (RES AND 128)<>0 THEN N=1
115 REM — limit RES to eight bits wide
120 IF RES=>256 THEN RES=RES–256
130 IF RES<0THEN RES=RES+256
140 RETURN
```

We can now expand this as an almost complete look-up table for the 6502 opcodes. (Almost, because some of the logic operations are simple in machine code but extremely complex in BASIC, and because the interrupt, stack and decimal modes have no easy equivalents in BASIC at all — so unless you really *like* getting confused ...). Remember that we only have three active variables: A, X and Y. The flags C, V, N and Z are only single bits in a register, and can thus only contain 0 or 1; and MEM and RES (result) variables represent temporary stores within the processor itself. The table below shows all of the 6502 opcodes with the exception of BRK, CLD, CLI, EOR, PHA, PHP, PLA, PLP, RTI, SED, SEI, TSX and TXS.

```
1000 IF OP$="ADC" THEN RES=A+MEM+C: GOSUB 30: A=RES
1010 IF OP$="AND" THEN RES=(A AND MEM): GOSUB 100: A=RES
1020 IF OP$="ASL" THEN RES=MEM*2: GOSUB 40: MEM=RES
1030 IF OP$="BCC" THEN IF c=0 THEN GOTO ...
1040 IF OP$="BCS" THEN IF C<>0 THEN GOTO ...
1050 IF OP$="BEQ" THEN IF Z<>0 THEN GOTO ...
1060 IF OP$="BIT" THEN RES=(A AND MEM): GOSUB 100: RES=MEM: GOSUB
10: GOSUB 110
1070 IF OP$="BMI" THEN IF N<>0 THEN GOTO ...
1080 IF OP$="BNE" THEN IF Z=0 THEN GOTO ...
1090 IF OP$="BPL" THEN IF N=0 THEN GOTO ...
1100 IF OP$="BVC" THEN IF V=0 THEN GOTO ...
1110 IF OP$="BVS" THEN IF V<>0 THEN GOTO ...
1120 IF OP$="CLC" THEN C=0
1130 IF OP$="CLV" THEN V=0
1140 IF OP$="CMP" THEN RES=A–MEM: GOSUB 70
1150 IF OP$="CPX" THEN RES=X–MEM: GOSUB 70
1160 IF OP$="CPY" THEN RES=Y–MEM: GOSUB 70
1170 IF OP$="DEC" THEN RES=MEM–1: GOSUB 100: MEM=RES
1180 IF OP$="DEX" THEN RES=X–1: GOSUB 100: X=RES
1190 IF OP$="DEY" THEN RES=Y–1: GOSUB 100: Y=RES
1200 IF OP$="INC" THEN RES=MEM+1: GOSUB 100: MEM=RES
1210 IF OP$="INX" THEN RES=X+1: GOSUB 100: X=RES
1220 IF OP$="INY" THEN RES=Y+1: GOSUB 100: Y=RES
1230 IF OP$="JMP" THEN GOTO ...
1240 IF OP$="JSR" THEN GOSUB ...
1250 IF OP$="LDA" THEN RES=MEM: GOSUB 100: A=RES
1260 IF OP$="LDX" THEN RES=MEM: GOSUB 100: X=RES
1270 IF OP$="LDY" THEN RES=MEM: GOSUB 100: Y=RES
1280 IF OP$="LSR" THEN RES=MEM/2: GOSUB 90: MEM=RES
1290 IF OP$="NOP" THEN REM
1300 IF OP$="ORA" THEN RES=(A OR MEM): GOSUB 100: A=RES
1310 IF OP$="ROL" THEN RES=MEM*2+C: GOSUB 40: MEM=RES
1320 IF OP$="ROR" THEN RES=MEM/2+(C*128): GOSUB 90: MEM=RES
1330 IF OP$="RTS" THEN RETURN
1340 IF OP$="SBC" THEN RES=A–(MEM+(1–C)): GOSUB 60: A=RES
1350 IF OP$="SEC" THEN C=1
1360 IF OP$="STA" THEN MEM=A
1370 IF OP$="STX" THEN MEM=X
1380 IF OP$="STY" THEN MEM=Y
1400 IF OP$="TAX" THEN RES=A: GOSUB 100: X=RES
1410 IF OP$="TAY" THEN RES=A: GOSUB 100: Y=RES
1420 IF OP$="TXA" THEN RES=X: GOSUB 100: A=RES
1430 IF OP$="TYA" THEN RES=Y: GOSUB 100: A=RES
```

Most of these are simple enough to follow, even if the BASIC code which represents them is somewhat tortuous. (it also gives you some idea of how much work the processor does even on the 'simple' instructions!). There are, of course, a few confusions:

ADC — remember that the 6502 has no simple addition — it *always* does an 'add with carry', hence the importance to clear the carry before most additions.

BIT — probably the most confusing of all the 6502 opcodes. It first does a logical 'and' between A and MEM, setting the Z flag accordingly; it then *copies* bits 7 and 6 of MEM into the N and V flags respectively (overwriting the setting of N from the previous GOSUB 100); and throws the rest of the result of these operations away, saving only the settings of these three flag bits as a result of the operation. The instruction is used mostly as a flag test, particularly with I/O devices like the 6821 PIA and 6850 ACIA that are used variously in the OSI systems; it also allows you to keep up to three flag-bits in the same byte (if you can keep track of them!).

CMP, CPX, CPY — note that, like BIT, these only set flags; they throw away the actual result of the operation, leaving the registers' contents intact. Because to this, the compare and BIT opcodes are sometimes used to 'conceal' another instruction, as was described in the discussion on the input routine in the first part of this series.

ROL, ROR — remember that these are *rotates*, not simple shifts, with the 'ninth' bit held in the carry. In ROL the carry is rotated into bit-0 (hence the simple addition), while in ROR it is rotated into bit-7 — hence the addition of C*128, C*2 ↑ 7.

SBC — note that, as with ADC, the 6502 cannot do a simple subtraction; it can only do one with carry, or rather 'borrow'. The *inverse* of the carry is used ((1–C) in the statement) — hence the carry must be *set* with SEC before a simple subtraction.

The remaining instructions not in the table above can cause even more confusion — hence the reason for leaving them out. The machine-code 'exclusive-OR' EOR instruction can't be described by a simple one-line statement — as you'll see if you look up what an exclusive-OR actually involves. Each bit of MEM has to be checked separately against the respective bit in A, calling for a fairly complex FOR ... NEXT loop. The other 'missing' instructions fall into three groups: the interrupt handlers BRK, CLI, RTI and SEI; the decimal-mode pair CLD and SED; and the stack group PHA, PHP, PLA, PLP, TSX and TXS.

BRK performs much the same action as STOP in BASIC, except that its method of working needs specific set-up routines before it will work (as opposed to hanging-up the system); in fact it acts exactly like a software-controlled version of an interrupt, and is only distinguishable from the IRQ hardware interrupt by setting the B flag-bit (bit-4) in the status register. There is no easy way to simulate the other interrupt instructions, except perhaps with the WAIT statement in BASIC.

The 6502's decimal-mode addition and subtraction is not something to be recommended without some experience, since out of the variety of arithmetical instructions available on the 6502, only ADC and SBC are affected by it — the rotates, shifts and logical operations are not. When the Decimal-mode flag (bit-3 of the status register) is set by a SED instruction, all addition and subtraction is done in BCD or binary-coded decimal form: a result greater than 9 in any nibble generates a further +6 addition, to correct the result to binary format (e.g.

$B+$6=$11, the correct BCD representation of decimal 11), and the reverse applies to values less than 0, where a further -6 subtraction takes place. As you can imagine, this produces decidedly scrambled results if the decimal-mode is applied in the wrong place, such as working with ASCII-coded text ... beware!

The last group, the stack operations, can be simulated to some extent in BASIC; but not really adequately, as the 6502 uses its single hardware stack to perform two separate functions. One is the temporary storing of registers' contents and the like, to protect them from being changed during some subroutine; the other is saving subroutine return addresses, something we cannot simulate at all in OSI's BASIC. We can simulate the stack itself quite simply: define an array called STACK(S), with a pointer variable called S. In the 6502, the stack pointer goes *downward* with increasing 'pushes'; so to start we set S at a fairly high level — say 40 locations — with X=40: S=X (the BASIC equivalent of LDX #$28, TXS, the standard SYNMON/CEGMON set-up value), and do an S=S-1 before each 'push', or S=S+1 *after* each 'pull' — making sure that S never exceeds 0 at the bottom end or the dimension of the STACK(S) array at the top:

```
1440 IF OP$="PLA" THEN RES=STACK(S): GOSUB 100: S=S+1: A=RES
1450 IF OP$="PHA" THEN S=S-1: STACK(S)=A
1460 IF OP$="TSX" THEN RES=S: GOSUB 100: X=RES
1470 IF OP$="TXS" THEN S=X
```

PHP and PLP, which save and restore the processor status register respectively, would need to be simulated by a FOR ... NEXT loop converting the contents of the registers into a single combined value, and restoring them in the same order.

The difficulty with this simple-looking approach is that it conceals a trap: 'pushing' A within a subroutine, and trying to 'pull' it *after* returning from the subroutine will result in chaos. The processor would try to 'return' to an address made up of the saved contents of A as the low-order address, the former low-order byte as the high-order part of the address, and the former high-order byte (in the unlikely event of recovery) masquerading as the former contents of A. To get round this, and to illustrate the problem, we need to change our earlier and simpler definitions of JSR and RTS:

```
1250 IF OP$="JSR" THEN S=S-2: GOSUB ...
1340 IF OP$="RTS" THEN S=S+2: RETURN
```

BASIC will tell us very quickly, in the form of a '?BS ERROR', if we exceed the limits of our software stack.

This roughly sums up the functions of the opcodes of the 6502. At first sight, it seems very limited, and a long way from BASIC — but that should illustrate partly the flexibility of low-level languages, and also the amount of work that went into writing your BASIC! By comparison even with other processors like the Z-80, for example, the opcodes may well seem very restricted. But the advantage of these simple opcodes is that they *are* simple; and the real power of a processor lies not just in its instruction set, but the way in which it can use it to work on memory — its addressing modes. By comparison with the Z-80, and particularly the 6800, the 6502 has a much richer set of memory modes. They are also extremely fast — the indirect-indexed mode, one of the most powerful of the set, will load the A register in a mere 2 clock cycles, compared to *23* for the nearest equivalent on the Z-80, while the 6800 has nothing like it at all. But the addressing modes — the means by which we arrive at loading the variable we've called MEM in the list above — can be very tortuous indeed: and that's what we will deal with in the next part of this series.

Make your computer communicate

Richard Elen

The idea of the Computer Bulletin Board Service (CBBS) is finally catching on in the UK, and about time too. The idea is simple: you equip yourself with a terminal (or make a computer pretend to be one), a modem, a telephone, and you can dial up a CBBS and access teletext-type data, software, articles, information and the like. The telephone we can take for granted: the other parts of the chain take a bit of thinking about.

The modem

A modem is a device which enables your computer/terminal to be connected to a telephone so as to be able to send or receive data. Although it is theoretically (and practically) possible to hook up your cassette interface to the phone line via an acoustic coupler (a device with a microphone and loudspeaker which squirts the cassette data tones into your telephone handset), this system has severe limitations. The main problem is that while you could talk to other OSI machines and UK101s, you would not be able to talk to PETs, Apples and the like: this makes your communication a little limited, although it might be useful for starters. More useful is to have a device which produces standard tones which are recognised all over the world, and at a standard baud rate which most people can utilise. Such a standard exists in the United States: the Bell standard, and many modems available there (and no doubt imported into the UK) use this standard. The usual baud rate is 300 baud, which suits the vast majority of systems. The usual form of a modem is a device which attaches to the computer via an RS232C serial link (or sometimes a parallel port), taking the data signals and converting them into a series of tones to the Bell standard at 300 baud.

The tones are then sent down the line either via an acoustic coupler (usually part of the modem unit) or a transformer straight on to the phone line. It is important to note here that British Telecom do not like you attaching strange devices to the telephone line, and all such devices require type approval. The penalty for not observing this requirement is twofold: you risk damaging BT's equipment (for which they will detest you forever); alternatively, they might visit you one day and rip out your modem just like they do illegal extension 'phones. However, there are devices which have BT type approval, and you will see them advertised from time to time. Many modems have a large range of facilities, the most expensive offering such things as automatic dialling. Such expensive complexities are fine if you have the time and the money to find them. But whatever type you obtain, this solves your first problem: attaching your system to the 'phone line in such a way as to be able to talk to other machines in an agreed standard fashion.

The terminal

The second problem is exactly what you attach to the modem, and what it has to do. In its simplest form, all you need is a terminal which can operate at 300 baud into your modem. Most of us, however, want a little more: namely, some form of intelligence, and storage capability. It is a little tedious to copy some kind of program off the screen of a terminal and into your machine, when your machine could do it by itself just as well! So what we need is some way of making our computers into 'intelligent terminals'. At least one US manufacturer, Micro-Interface, offers software to do this for the Superboard: its *ROMTERM* software (in two versions, for Series II Superboard and for disk-based systems) is being marketed as *StarLink* by Mutek. This software makes your system largely compatible with many of the CBBS in the States, and no doubt British systems (including our own, about which more later) will be organised along similar lines, with similar protocols. The other alternative is to write it yourself.

Terminal software
Before designing software of our own, we should define what we expect it to do. At the basic level, the system should be able to allow typing at the keyboard to be sent down the line, either displaying the text on the screen as it is typed (half-duplex) or displaying the text as it is 'echoed' back by the machine at the other end (full duplex). As different systems and modems use one or the other, our software should be able to handle both; this is simply a matter of printing the character as it is sent for half-duplex, or not for full duplex operation. It may be useful to be able to alter the baud rate: although 300 is normal, you may encounter other speeds. The 6850 ACIA in OSI gear enables the baud rate to be software-determined to a fair extent.

The system must also be able to receive data from the CBBS computer, *even if it responds while you are typing* (if you are running full duplex it will certainly have to do this!). There are two basic approaches to this: either your software must look at the ACIA 'between' characters from the keyboard to see if something is coming in, as part of the main routine (this can be sluggish, as most of the time nothing will be happening; although in machine-code it is not too great a problem); or you can make use of the fact that the ACIA has an interrupt handler which can be hooked up to divert the computer into a 'receive' routine when necessary. We will consider this approach here, as it is quite easy and gives you a chance to try out those rusty interrupt vectors!

As well as talking and listening, the system should be able to download a program from the CBBS machine, store it in memory or on disk/cassette, and recover it later. This is more complex than simply what we might call 'CB Mode' and we will not consider it in this article. For disk users, a good place to start is by using the Indirect File system to get the downloaded program 'out of the way' as an ASCII file in memory so that you can call it into the workspace and save it later at your leisure. *ROMTERM* also does this for non-disk-based C1 users and the concept is quite straightforward.

Another feature of our software could be to allow the machine to send your own programs to the CBBS machine for other users. If it can't do this, you will have to rely on software supplied by the CBBS itself, thus neglecting one aspect of the system which is most fascinating: that of exchanging programs. But, for now, we will hold a discussion of this over for a later article.

A starting point
We will now consider a minimal system which we can use as the basis of a more complex CBBS communicator. Hopefully some readers will be able to expand on this in later issues (hint!). The basis of this approach is taken from a useful article in the excellent American computer magazine, *Microcomputing*. In their May 1981 issue (p.208), James C. Daly describes the basis of a terminal routine for the C1, and his article (and the magazine in general) can be heartily recommended.

Interrupts
The central part of this approach revolves about the use of the IRQ (Interrupt ReQuest) facility of the 6502, and the fact that the 6850 ACIA has an interrupt handling facility which can be used to force the computer to 'listen' when data is trying to come in. There are in fact two interrupts on the 6502: the IRQ, which can be screened off and ignored by the processor if desired, and the NMI (Non Maskable Interrupt) which can't. The latter is generally used in large systems to

call up an emergency routine in the event of a power failure. A sensor is used to detect a nasty on the mains (eg it is disappearing or fluctuating in disturbing ways). When this sensor operates, it holds a line low, which is connected to the NMI pin on the processor. The NMI cannot be ignored, and tells the processor to stop what it is doing and vanish off to some predetermined location (the NMI Vector tells it where to go to: in our machines it is usually sent to $0130) where it finds a special interrupt routine or program which generally saves the important aspects of the main program in, say, memory which has battery backup, thus enabling you to continue from more or less where you left off when the power has returned.

The IRQ is similar in concept, in that when activated it forces a jump (usually to $01C0 in our machines) at which location it is told what to do. The difference is that one of the processor flags (the Interrupt Mask flag I, conveniently enough) tells the processor whether it is to go off and service the interrupt or ignore it altogether. When this flag is *cleared* (with the CLI instruction), the processor will service an Interrupt ReQuest when it sees one. When it is *set* (with the SEI instruction), the processor ignores any IRQs that might be presented to its hot little pin. You will notice that this flag works in perhaps the opposite way to that which you'd expect: this is because it is really an interrupt *mask* flag: when the flag is set, an interrupt is masked, ie ignored. For this reason, the NMI is a Non Maskable Interrupt: in other words, the processor cannot ignore it, however boring and tedious the 6502's silicon intelligence might find the interrupt routine (if you'll pardon my anthopomorphism). Here endeth the tutorial on interrupts.

A little hardware mod
Before going any further, we need to perform a little hardware modification. But before you run away in disgust, I should tell you that all that is required at the minimum is a single piece of wire. You don't even need a soldering iron unless you feel like it.

When data is received by the ACIA, it sends its interrupt request pin (pin 7) down to ground potential. Normally this is meaningless to OSI machines because pin 7 doesn't go anywhere. Take a suitable piece of wire and connect pin 7 of the ACIA to the IRQ pin (pin 4) of the 6502. You can do this either by stuffing the ends of the wire into the IC socket next to the correct pin, or you can solder it via a switch to the underside of the board. Note that however you do it you should be able to remove the wire or disable the interrupt line (with the switch) when you don't need the facility. Otherwise strange things may start to happen when you are trying to load programs from cassette. For Superboard or C1 users, you may find it useful to push the wire from pin 7 of the ACIA (U14) into pin 1 of J1, the expansion connector. This pin carries the IRQ line to the processor with no unpleasant bending.

Apart from implementing the RS232 port on your machine if it isn't already there, this is all the hardware modification that you need to do for this application. It is worth noting here that while the RS232C standard states that the serial data should swing between about +5v (logic '1') and −9v (logic '0'), some modems only require the logic '0' voltage to drop to zero, and not to −9v. If this is the case with your modem, you will not need to implement the negative supply rail on the RS232 interface: it can be grounded instead. This may save you extra bits of power supply.

The program
This basic program is in machine code (no pun intended: I mean 'basic' in the sense of 'ready to be expanded upon') as the interrupt mask is difficult (otherwise known as 'impossible') to handle from BASIC, although you could use a BASIC

driver program to get you into it, either using a USR(X) to call the program from BASIC, or by executing a CALL to the start of the main routine if you have the new BASIC 1 chip. It is simplest, of course, simply to load it and execute it with the 'G' command from the monitor (or 'GO' from the DOS kernel). It is written in assembly language to give you practice with that funny tape you bought the other week called the Assembler, or that weird DOS command called ASM. More importantly, assembling it yourself not only gives you practice (it is small, and can even be hand-assembled) but also means that you can stuff it into memory anywhere you like.

The program has two parts. One sets up the ACIA, gets a character from the keyboard, and throws it out of the ACIA, while the other is a program called by the interrupt, which gets a character from the ACIA and displays it, checking to see that there are none left before returning to the input routine. You will note that the latter program section has an RTI instruction at the end: this tells the computer simply to ReTurn from Interrupt servicing — it's rather like a special kind of RTS, which returns from a special kind of subroutine which has a hardware-forced interrupt instead of a software-commanded JSR.

The two programs may be described like this:

'Send' routine

1. Clear interrupt mask on processor
2. Reset ACIA
3. Set up ACIA for desired baud rate, define word length and parity, and enable ACIA interrupt
4. Get character from keyboard
5. Save character on the stack
6. Read ACIA status register: wait until transmit data register is empty
7. Pull character off the stack
8. Send character to ACIA
9. For half-duplex, display character on screen (omit this step for full duplex)
10. GOTO step 3

'Receive' (interrupt) routine

1. Disable processor interrupt (set the flag)
2. Save registers on the stack (so the status of the 'send' program isn't lost)
3. Get character from ACIA
4. Display character on screen
5. IF there are more characters to come, then GOTO step 3. ELSE continue
6. Pull registers off the stack (ready to return to 'send' routine)
7. Enable processor interrupt (clear the flag)
8. Return to 'send' routine

A point should be made about step 3 of the 'send' routine. The 6850 ACIA can support a number of different baud rates, via a programmable on-chip divider, and can also send and receive data with different numbers of bits/word, stop bits, and types of parity. The standard for OSI is 8 bits word length, no parity and two stop bits, and 300 baud is achieved by dividing the clock by 16. This specification is achieved by stuffing $91 into the ACIA's Control Register. To give a 300 baud output and interrupt enable, this and other combinations are given below:

Contents (hex)	*Word length (bits)*	*Stop bits*	*Parity*
81	7	2	even
85	7	2	odd
89	7	1	even
8D	7	1	odd
91	8	2	none
95	8	1	none
99	8	1	even
9D	8	1	odd

The 6850 has two memory addresses: the lower of the two is the address to which the Control Register is *written*, and where the processor can *read* the Status Register, which tells it whether there are other characters to send or receive; while the higher of the two addresses is the data register which may be written to (for sending) or read from (for receiving). The ACIA is located at $F000/$F001 (standard C1 and UK101) or $FC00/$FC01 (C2, C4 etc); non-standard format monitors may have them placed elsewhere. These addresses should be inserted into the routines as appropriate.

Send routine

```
10               *=$(start location)
20               ACIA=$F000     ; $FC00 for C2 etc (ACIA address)
30  START        CLI            ; Clear interrupt mask
40               LDA #$03
50               STA ACIA       ; Reset ACIA
60  NEWCHR       LDA #$91       ; Set up for 300 baud, OSI standard (change if required)
70               STA ACIA       ; and enable interrupt
80  GETCHR       JSR $FFEB      ; Get character from monitor keyboard routine
90               PHA            ; Save character on stack
100 CHECK        LDA ACIA       ; Get ACIA status
110              LSR A          ; Check bit 1 of status register,
120              LSR A          ; if clear,
130              BCC CHECK      ; try again (check last char has been sent)
140              PLA            ; Get character from stack
150              STA ACIA+1     ; Send character to ACIA
160              JSR $FFEE      ; Display char (half-duplex only) via output vector
170              JMP NEWCHR     ; Do it all again
```

Receive (interrupt) routine

```
180 RECV         SEI            ; Set interrupt mask
190              PHA
200              TXA            ; Save processor registers on stack
210              PHA
220              TYA
230              PHA
240 NEXT         LDA ACIA+1     ; Get character from ACIA
250              JSR $FFEE      ; Display character via output vector
260              LDA ACIA       ; Read ACIA status
270              LSR A          ; If status bit zero=1, (i.e. more to come)
280              BCS NEXT       ; then get another character
290              PLA            ; Otherwise, restore registers
300              TAY            ; by pulling
310              PLA            ; off the stack
320              TAX            ; and transferring
330              PLA            ; where necessary
340              CLI            ; Enable processor interrupt again
350              RTI            ; Return to 'send' routine
360                             ;
370              *=$01C0        ; System interrupt vector
380 IRQVEC       JMP RECV       ; Set up IRQ jump to 'Receive' routine
```

Note that this program utilises two monitor routines: the keyboard routine (accessed via $FFEB) and the display driver (accessed via the output vector at $FFEE). The use of an interrupt routine for received data means that the normal keyboard routine ('wait until a key is pressed') will not cause loss of data as an interrupt can bring in received data even while the keyboard routine is waiting for input!

What you can do now

Implementation of the above routines (added to if necessary) plus the required hardware can bring you all the benefits of CBBS access, if you can find one to use, or a direct line to other users. As it makes your machine pretend to be a terminal, you should even be able to access a CBBS designed for other machines with this program — the programs on such a system may be of little use to you however! To remedy the problem, we are considering starting up a User Group CBBS, using a multi-user C3 and a handful of telephone lines. We'll keep you informed as the system develops. There is still a good deal to be done to get such a system 'on the air'. The database of interesting information is one problem, but a more immediate one is to define protocols which will suit the majority of users. The system will need to be able to ask what machine is in use and set itself up accordingly to send 'pages' of the right screen size, with the right formatting commands to suit C1s of both types, UK101s, serial systems and C4s, and will have to send out, for example, the right screen-clear commands to suit different screen handlers (eg CEGMON). This will take a great deal of time and effort, and anything readers have to suggest on all aspects of the system will be welcomed. At the moment we envisage some kind of password and identification 'logging in' system, which will recognise User Group members and give preferential rates for access (obviously the thing will have to pay for itself), but it is early days yet. Quite obviously, CBBS are the systems of the future, and we expect the system to develop as time goes by. In these early days, however, we need as much help with ideas as possible, to provide another service to User Group members.

REVIEWS

Expansion board from Elcomp

New from Elcomp is an expansion board with a difference: it plugs into the 40-way expansion socket on a Superboard or UK101 and provides on S-44 card slot and, wait for it ... four Apple-type 50-pin slots. While you can't use Apple cards which rely on dynamic RAM refresh, the Apple monitor, or the language card system, there are still a large number of boards which *can* be used. Elcomp also offer a range of Apple-style cards for prototyping, I/O, EPROM programming, EPROMs, 12-bit A/D, sound generation, and other purposes. As is typical of Elcomp, the prices are not incredibly cheap. The Elcomp-1 expansion board (order number 606) costs $49.00 for the double-sided PCB and instructions only — you still have to buy the components. Elcomp can be contacted in Europe at PO Box 75437, D-8000 Munchen 75, West Germany.

Word Wizard, Codekit and BASIC 5

These three programs are some of the many utilities currently marketed by Premier Publications. Word Wizard is a machine code word processor, Codekit is a single line assembler/disassembler, and BASIC 5 is an extension to BASIC providing 17 new commands. Premier Publications' software is designed principally for the UK101, but versions for the Superboard and other OSI machines BASICs are available from them.

Word Wizard

Many of our members aspire to have a good word processor to run on their machines. Almost certainly they neither want nor need the business type, mostly used to write the canned letters that the AA and other organisations send you. They (our members that is) also have to contend with a limited amount of memory, so programs with unnecessary features simply reduce the amount of text space left for you to use. The Word Wizard, written by N. Davies, neatly fills this gap.

To use this program you load your text from tape or type it in, only using the return key to terminate a line when this is necessary for the text format, for example at the end of a paragraph. The program uses its own keyboard routine, so the vices of SYNMON (if you are still running it) do not have to be contended with. At any time the cursor may be moved back non-destructively to any point, and text may be inserted or deleted at that point. The cursor may then be moved forward or moved directly to the start of text using one key stroke. A very good feature of this word processor is that the screen can scroll both ways, either on a line by line or on a page by page basis, so that the screen acts as a true window on memory. To help format your text, right margin justification of a line, and centering of a line are achieved by typing the appropriate control code in front of the line concerned, while Tab, Space and end of page functions are also provided.

Blocks of text can be moved or copied from one place to another, and a global search and replace function allows you to search for a string of up to 48 characters and replace it, if desired, with another. This allows you to use shorthand codes for frequently occurring phrases or words, aids you to correct spelling errors and the like. I have also found it very useful in writing BASIC programs, as the program can be heavily compressed, relieving one of much of the typing, and helping to ensure correct syntax, getting the PRINT statements to line up etc.

At any time, the text can be stored on cassette, and when required output to a printer complete or from the cursor position. The processor will format the output on printing, and even justifies the right hand margin of the text to the desired width by adding extra spaces unobtrusively in the line. These functions make no demands on the printer used, but a facility is provided to pass control codes to the printer if required.

Codekit

This is a very useful program to help those of you interested in machine code. The program, which is entirely relocatable within the memory map, includes a single line assembler which allows you to write machine code in mnemonic form instead of having to convert to hex, although as it only works on one line, labels cannot be supported in the same way as a 'standard' assembler. Operands can be stated in either hexadecimal, decimal, binary or ASCII form, and branches require either the absolute destination address or its displacement from the branch as an operand.

A disassembler is also provided, which either decodes seven lines of a code at a gulp for screen viewing, or will disassemble a complete block for output to the RS232 port for printout. A facility exists for moving blocks of code by 127 bytes in either direction, so that you can make room for those extra bytes that proved necessary. Although the assembler has limitations for extensive machine code work, Codekit is a useful way of creating relatively small machine code routines linking into BASIC via the USR command. As Codekit is compatible with BASIC, you can move from one to the other without having to resort to cold starts and possible program loss. As Codekit is only 2Kbytes long, enough workspace is left to make this a practical proposition on an 8K machine.

BASIC 5

This program, written by P. Rihan, is a 2 Kbyte extension to OSI's versions of Microsoft BASIC and is available to work with either ROM or disk BASICs. 17 extra commands are provided in addition to all the usual ones, and the extension is done so that BASIC is not substantially slowed down and the stack's integrity is preserved. This means that there are no limitations as to the use of the new commands (certain methods of extending BASIC can run into trouble if the new commands are placed in a FOR ... NEXT loop or a GOSUB) and the appropriate BASIC error message is generated should an error occur within routines using the new functions. All the new commands are prefixed with an '&' and are divided into three categories.

General commands

GETkey — this much needed command is non-halting and allows you to get characters from the keyboard, returning a null if no key is pressed.

GO *xxxx* jumps to a machine code subroutine at *xxxx* which address may be expressed in decimal or in hexadecimal (at last!).

GT and GS are commands which allow you to GOTO or GOSUB a variable, very useful in menu selections and the like.

RD*xx*<VAR will read down *xx* DATA statements and put the next item into the variable VAR.

INAT and PUTAT allow the input and printing of a string of given length (the string is truncated if an overflow occurs), from any position on the screen that input statements can be executed without messing up graphics.

Formatting

A PRINT USING command is provided with its associated image command. This allows extensive formatting of text fields to both screen and printer, it will also truncate numbers or text, align decimal points etc.

WI and CWI are for CEGMON users and will allow them to set up and change screen windows anywhere in memory. Decimal or hex arguments are permitted in this most useful feature.

Graphics commands

SCR will fill the screen with any character — fast.

BLK will draw a block of any character on screen — also fast.

VLIN and HLIN will draw horizontal or vertical lines of any character on the screen.

SET and TEST allow you to put a character on the screen or to find out what is there.

All the screen based commands use row and column arguments with the origin set at the bottom left hand corner to set the screen locations and are fully error trapped so that you can only write in the area defined by the current screen window.

The above brief description of the functions of BASIC 5 will, I hope, give the reader some impression of the dramatic extent to which the problems relating to formatting input and output in BASIC have been resolved.

All the above programs are sold by Premier Publications on cassette, disk or in EPROM. In EPROM they are available on switch on, of course, and form part of Premier's TES system. The instructions as to their use are adequate, and give examples of each command, and Premier have learned a reputation for being helpful to any client who has difficulty in using any of their products.

Premier Publications' BASIC 4

Reviewed by Richard Elen

No sooner have Premier released P. Rihan's BASIC 5 to the world than they come up with yet another goodie! Premier are fast establishing themselves as the best source of utility software and firmware for our machines, and BASIC 4 represents another string to their bow.

Primarily by throwing away ROM BASIC's announcement messages and a few bits of redundant code, BASIC 4 is able to offer a full set of commands for SAVEing and LOADing programs at a greater speed than normal, and with file names if desired. The speed enhancement is obtained by using a method of saving the bytes directly, rather than LISTing the program in ASCII form onto cassette. Time savings of up to 33% can be obtained by this approach, which is similar to that used by the PET and other machines. The original SAVE and LOAD commands are retained, so as to provide compatibility with existing tapes and other, less fortunate users; the only disadvantage of the byte-load format being that 'automatic merging' of programs is no longer possible. To do this you still have to have your programs recorded in the normal way. BASIC 4 additionally incorporates an indispensible "crash-recovery' command, OLD, which allows you to cold-start and recover program pointers if you've inadvertently POKEd vital bits of BASIC's page-zero locations.

Installation

It is very easy to fit BASIC 4 into your machine: normally you will have a ROM in the appropriate hole in your board, but if you've already replaced BASIC 1 and 3 with unmasked versions with CALL and a 'fixed' garbage-collector respectively, the procedure will be well known to you. You simply bend out pins 18, 20 and 21 of the new BASIC 4 EPROM, fit it into the BASIC 4 socket on your board, and connect up the pins, pin 18 going to ground, pin 21 to +5v, and (on a Superboard) pin 20 to IC 17 pin 4. The whole process takes a few minutes at the most.

Powering up

On powering up and cold-starting, the familiar 'Memory Size?' prompt appears. It is at this point that you type OLD if you are trying to recover a program. Answering in the normal way drops you straight into BASIC: 'Terminal Width' and the BASIC start-up message have been absorbed to provide room for the code. Of course, POKE 15 still changes the terminal width, which is probably how you did it anyway, and of course, you leave it alone if you have CEGMON (the operating instructions kindly point this out). At this point you are now in for a surprise: BASIC 4 supports Rihan's BASIC 5, and if you have it it resident, any key pressed after cold-start will produce the 'Ready' prompt instead of the original 'OK', telling you that BASIC 5 has been initialised.

Once in action, BASIC 4 primarily adds a new set of SAVE and LOAD commands. These are as follows:

SAVE is simply the normal command for saving to tape: there is no change in any respect to the normal. *SAVE "filename"* saves a program in byte-format with the specified filename. LIST is not used, and the program is not displayed on screen. The BASIC prompt announces the completion of the operation. *SAVE"* (no filename and no closing quotes) saves a program in byte-format without a file name: in other words you can't get it back without the corresponding command to load a byte-format tape without a filename *(LOAD")*. Premier describe this as a useful way of preventing illegal use of a program, but I suspect that if you had BASIC4 you would also know about *LOAD"* as well!

The simple command *LOAD,* once again, loads a program as normal: no change. *LOAD"filename"* loads *and auto-runs* the program named. If other filenames are encountered on the tape, their names will be displayed but they will not be loaded. A loading error is indicated by the word BREAK and the BASIC prompt. This command clears the workspace before loading. *LOAD"filename* (note lack of closing quotes) clears the workspace and loads the named program as above, but doesn't auto-run it. Similar syntax applies to the *LOAD""* command: it loads the first program encountered and runs it. Missing out the second quotes character loads, but doesn't auto-run the first program found. Useful, both of them, if you've forgotten the filename.

In fact, as the system loads and saves quite happily at any speed up to 4800 baud (the highest I tried), you could end up with a tape full of programs and dozens of filenames (each up to 32 characters) to be forgotten: a tape directory is a good idea. I found a neat way of doing this, by typing *LOAD"filename"* where "filename" is something non-existent. Running the tape then lists all the filenames on the tape. If you note down counter readings at the same time, you can then use CEGMON's screen editor to call the names off the screen into a program whose line numbers are the corresponding tape-counter readings. For example:

```
1 ?"Directory for tape no. 2, Sept 04 1981"
7 ?"FILE 1
11 ?"FILE 2
... etc.
```

SAVE this program under the name 'DIR' (e.g. leave room for a big one at the front of the tape), and then type *LOAD""* before trying to find the program and up comes the directory. This is very fast!

Another command which is most useful is *LOAD?"*. This is a 'verify' command which compares a tape copy with the resident BASIC program, forcing a BREAK and BASIC prompt to be printed in the event of a bad comparison.

The final, and very useful, command in BASIC 4 is *OLD*. This, typed in response to 'Memory Size', resets BASIC's pointers to look at a program which has been accidentally lost by inveterate POKEing into page zero, the stack, or page 2. It will *not* help you if you've overwritten the program in the workspace, of course! It is worth noting, however, that occasionally a program will crash again when you try to run it after recovering with reset, cold-start and *OLD*, so it is worth SAVEing it first. Often this is the result of having a POKEd variable assuming an unexpected value, so, of course, it will crash itself once again the next time you try. SAVEing the program first allows you to take the extreme measure of a complete cold start if something really odd has happened.

Conclusions
All in all, BASIC 4 is a worthwhile addition to a cassette-based machine. At a cost of only £ , it will hardly break the bank. The added SAVE and LOAD commands are most useful and fast, especially at 4800 baud, and although the inevitable loss of the ability to append programs in this mode is a pity, it matters not because of the continued existence of normal LOAD and SAVE. Indeed, I wonder if it might be possible to write an append routine to patch it in: without looking into the code more deeply I wouldn't know. The routines in BASIC 4 are sensibly and economically written, and use a severely limited space most effectively, providing facilities which many users have been after for some time. I believe I also noticed some clever diversions in the code to detect illegal copies, and there also appears to be a unique chip ID in case anyone was selfish enough to try. The single A5 double-sided instruction sheet is concise and clearly understood, and the installation instructions are quite sufficient. A useful product which fulfils a long-unsatisfied need.
Premier Publications, 12 Kingscote Road, Addiscombe, Croydon, Surrey. Tel: 01-656 6156

One important point to note about BASIC 4 — and this isn't in the manual — is that it requires CEGMON to function. It does not include the old screen handler at $BF2D (this, too, has been removed to make room for the new code), so it requires CEGMON's screen handler to operate at all. It should also be noted that, as a result, machine code programs which start below $0235 must be relocated or otherwise dealt with, as the OLDSCR location in CEGMON will not be able to call $BF2D when the output vector is pointed at it, thus removing the capability of running such routines unchanged under CEGMON. This fact is a nuisance and a great pity, but understandable in view of the room needed for the new functions in BASIC 4.

32×64 Display for Superboard II

J. R. Fornalski

The following article describes a method to obtain guard bands for the Superboard II and is an addition to the modification published in Vol.1 No.2. However, we feel that it will be possible to implement it on computers with other video modifications, including the new Series 2, with some extra circuitry. This circuit is a distant relative to the one suggested by Dr. Abbott in Vol.1 No.4.

Note that throughout the article we are using IC*xx* to refer to a chip on the new board, while those on the computer are referred to by U*xx*, as in the circuit diagrams.

How it works

This modification provides a continuous clock for the processor at the selected frequency via U29, and stops the clock driving the counter chain during the period of line blanking. Using a 12MHz crystal allows adequate guard bands for the television, while providing a line frequency of 15,525.644 Hz. The clock to the ACIA, which is derived from the counter chain, has a very small jitter which is averaged by the ACIA's divider, and the rate is now closer to 300 than it was originally.

If the data blanking is high IC2 is disabled by its R0 line (See Fig 2). A negative going edge from the line reference to U65/9 (C8) triggers IC4 whose output is gated with inverted load pulses to set the latch fromed by part of IC5. This pulse corresponds to the "65th." character. The latch sets, making data blanking go low and stops the clock to the counter chain while enabling IC2. IC2 produces a travelling logic 0 at the outputs of IC3, which is used to *a)* delay the line sync pulse and *b)* generate the delay while overscan takes place. When this pulse reaches the "line trigger" (fig.3, wire E) U65 generates the line sync pulse. The "travelling 0" continues until the output selected by "line frequency adjust" (fig.4, wire D), resets the latch (IC5). This resets data blanking to a high and restarts the counter chain. The process then repeats.

Parts list

1 × 7492
2 × 7493
1 × 74154
2 × 7400
1 × 74LS121
1 × 12MHz Crystal
2n2 cap; 12KΩresistor; matrix board, Vero pins etc.

LS versions of the above chips may be used, but check the pin-outs, especially for the counters.

Build the circuit shown in fig.2. A suggested layout for a matrix board using Vero-wire techniques (as used in several conversions) is shown in fig.3. It is suggested that vero-pins be used to facilitate access to the input/output points on the board.

Initial modifications

Refer to fig.1. On the underside of the board, cut track 'A' near PTH A isolating the processor clock line (to U8/37). Join track A to U30/14. This will run the processor at 2MHz. Turn on the computer and check that all functions still work. If they do not it is likely that either the RAM or EPROMs, rarely ROMs, have access time problems. If this is the case, either weed out the offending devices and exchange them, or settle for a clock speed of 1.5MHz or 1MHz at which few problems should occur.

Replace the crystal X1 for the 12MHz one. Remove the link from track A to U30/14, and connect track A to U29 as follows:

speed	*join*	*to*	*comment*
2MHz	track A	U29/8	use 7492 as U29
1.5MHz	track A	U29/11	join U29/2,3,10; use 7493 as U29
1MHz	track A	U29/13	join U29/8 to U29/14; use 7492 as U29

Check that the computer still works. The screen display will not make sense, but it should be possible to see the screen clear when BREAK is pressed.

Connecting the new board

Refer to fig.4 and locate W9. Cut the link between points B and C. Note that point C leads to the counter chain, at U30/2. Remove U65 (74LS123) from its socket, bend out pin 9 and replace it.

join	*to (new board)*	*comment*
U65/9 (track)	IC4/3	line reference (C8)
fig.1, pt.A	IC1/14	Processor clock (∅0)
U65/9 (pin)	IC3/4	Line trigger (2MHz)
U65/9 (pin)	IC3/3	Line trigger (1.5MHz)
U65/9 (pin)	IC3/2	Line trigger (1MHz)
fig.4, pt.B	IC6/1	12MHz input
fig.4, pt.C	IC6/3	Gated 12MHz output
U42/1	IC5/4+5	Load pulses

All leads should be reasonably short in view of the frequencies involved. Check the computer at this stage. A picture with 64 characters/line should be displayed and the computer should operate normally. However, as blanking has not yet been applied, certain characters may cause the picture to streak.

Blanking

Locate U59 (7420). It is necessary to cut the track leading to pin 12, which, as luck would have it, is only accessible under the chip. The easiest solution is to cut pin 12 just above the surface of the board, bend it out and connect it to IC51/8 on the new board. This completes the modification to the Superboard II.

Adjustments

The timing on monostable IC4 seems fairly flexible. For maximum symmetry (T3) may need alteration. This is a selected output of U43.

The picture position may be shifted slightly by the "line trigger" wire E; the line frequency is affected by wire D.

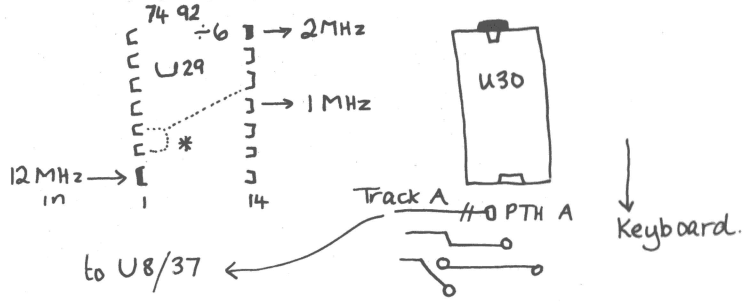
It should be easy to adapt this modification to work with other video conversions to the Superboard with little difficulty, although care will be needed in the region of U29. It is important that the cassette interface is driven from the interrupted clock line, otherwise its rate will be incorrect. Owners of the Series 2 Superboards should be able to achieve 32×32 or 16(!)×64 by adapting it. The majority of components on their machines are labelled as in the older versions, so that most of the instructions will apply. Here are some points which will have to be dealt with (these seem obvious but have not been tried out). If a 12MHz crystal is used, the frequency doubler formed by U79/4, 5, 6, 8, 9, 10 must be disabled. W9 (fig. 1 points B and C) might be emulated by removing U79, bending out pin 8, and re-inserting. U79/5 would then act as a source of the 12MHz clock and U79/8 socket would provide access to the top of the counter chain corresponding to point C.

A divider would have to be added to provide the processor with uninterrupted clock pulses.

The line trigger referred to is not labelled C8 but is still accessible at U65/9.

As U56 does not have spare inputs on this machine, spare gates on IC6 could be used to gate (T3) and DB, the inverted output being fed to U56/13.

Fig 1.
Underside of Board.



* IF U29 is 7493 Link plus 2,3,10.

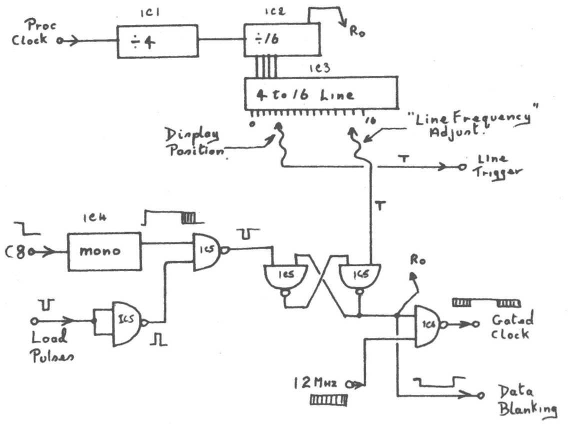


Fig 2. Basic Block Diagram.

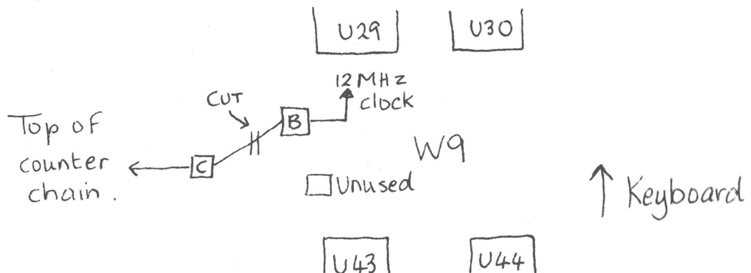
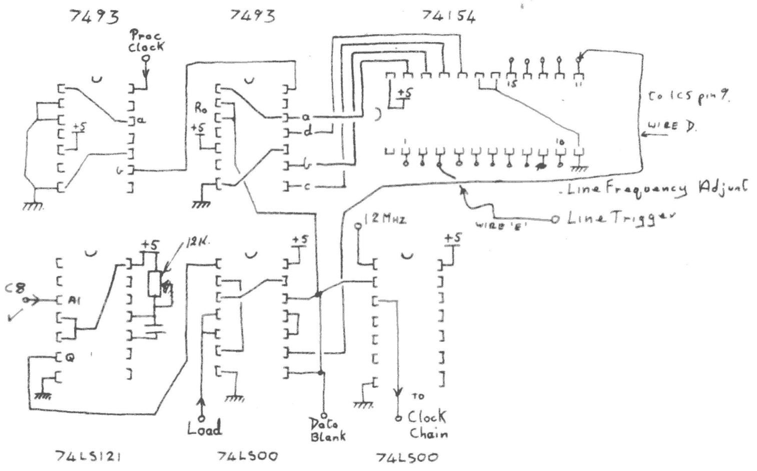


Fig. 4.

fig. 3



Proc Clock - from 65∅2 pin 37
C8 - From track of U65 (pin 9) NOTE 1
Load - from U42 pin 1
Data Blank - to U56 pin 12 NOTE 2
Clock Chain - to W9 NOTE 3
12 MHz - from W9
Line Frequency Adjust for 16·625 Kc/n NOTE
Line Trigger - Adjust for central Display positio

Subsequent trials on modifying a Series 2 Superboard indicate that the modification is not quite as straightforward as was first thought. A promising line of research indicated that both screen formats should be possible if a retriggerable monostable (74123) is used instead of the 74121 on the new board. Would any member successfully modifying a Series 2 please supply us with details.

Readers should note that 1.5MHz operation will prove difficult with disks — the data ACIA's clock is derived from ∅2 and NMHz software only supports operation at 1, 2 or 3.3MHz — Ed.

Dola Software

117 BLENHEIM ROAD, DEAL, KENT

New Programs from DOLA SOFTWARE

Front Panel Program for UK 101

On meeting a breakpoint in your Machine Code program, our new Front Panel program shows all the registers in Hex, Decimal, Binary and Character (as for a POKE to the screen). Any of these can be changed by moving the cursor around. In addition any part of the memory can be shown in the same format, and altered at will. The program occupies 1.25K starting at 1B00, but a free BASIC program, used in conjunction with the Extended Monitor, allows relocation. The program is written for the New Monitor, but a CEGMON version is in preparation. Price £8.00 on cassette.

We also have a Text writer program for the UK 101 for both the New Monitor and CEGMON. This does not pretend to be a word processor, but allows you to write text, add and delete lines, print the text, save it on cassette and load it back for modification. The program also solves the problem, which not everyone knows about, of dropped characters when reading text from tape. This is due to the garbage collector doing its thing while reading your data!
Price £3.00 for documented listing; add £2.00 for cassette.

The ***Dola Software*** library contains stand alone programs in BASIC and many routines, often in Machine Code, that you build into your own programs. These include graphics routines, PIA based programs including an accurate Frequency Meter, AY-3-8910 music chip subroutines and programs and a very fast large digit (7×5 pixel) screen display. There are also some original games.

The programs are written for the UK101 using the New Monitor, but for the Superboard and CEGMON, only minor changes will be needed.

Send an SAE (large) for the catalogue.

For Sale.
C2-4P. 20K RAM. Cegmon. 2MHz. room for 12K more RAM on board. Manuals and numerous tapes, chess, utility etc. **£335.**
Ring 070-682-6188 after 6pm (Lancashire)

NEW IMPROVED CHARACTER SET in EPROM for your UK 101/Superboard (please state which). Includes a full set of Pixel graphics characters, maths, electronic, gaming and other symbols.
Price **£8.00** (+50p P & P) to User Group Members. Reprogramming service available. Send SAE for more details or phone Harrogate 503276. J.O. Linton, 110 Ducy Road, Harrogate, HG1 2HB.

PROGRAMMABLE SOUND GENERATOR — Velvet Software, bought as kit for £50, put in case, stereo DIN socket, 3 channel select switches, 4 software-controlled relays, 2 I/O parts, control signals available on 38-way edge connector. Connects via 20-way cable and D-connector. FAULTY and wiring needs to be done on C1/SUPERBOARD/UK101. Cost £65. Only £40.
N.A. Cannon. Tel. (0737) 65863.

BASIC5 for UK101 and OHIO

The most devastating enhancement yet, adding 17 new BASIC words to your interpreter which can be used in program lines and give machine-code respose speed to graphics and formatting

HLIN, VLIN, SCR, BLK, SET and TEST allow generation and manipulation of graphics at speeds which are unobtainable in BASIC
PRINTUSING, PRINTAT, INAT allow total control of screen input/output
GET (key), RD (Read DATA), GS and GT (GOSUB and GOTO a variable), GO and GO$ (GOTO a machine-code routine) allow total program flexibility
WI and CWI allow CEGMON users to manipulate their screen under variable control, using one command, in Hex or Decimal
BASIC 5 is available for CEGMON, SYNMON, and MONO1/2 only. State precisely your computer and monitor when ordering. Comes complete with comprehensive manual.
Available on DISK or in EPROM (9000hex) £19.95

BASIC4 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
- crash recovery command (OLD)
- original SAVE/LOAD commands unaltered
- reduces LOAD/SAVE times
- seven new SAVE/LOAD commands
- non-destructive memory test
- initialises BASIC5 automatically if resident

BASIC4 is a plug-in replacement for your existing BASIC4 ROM, PRICE £11.95

CEGMON – ONLY £25.87 inc

TOOLKIT 2 for UK101/OHIO

The most powerful TOOLKIT on the market, TOOLKIT 2 gives you all 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 generate new line numbers automatically, any start, any increment.
DELETE high-speed block line delete.
VIEW examine cassette contents without loading to memory.
TRACE superb trace feature – screen transparent. Can be turned on and off within a program
MC enter the monitor quickly!
TOOLKIT 2 also lists the relevant line of BASIC where any error occurs and cures the warm-start 'OM ERROR' bug.
Available in EPROM only (8000hex), for CEGMON, MONO 1/2, and SYNMON monitors (DISK soon). Price £19.95. State machine and monitor when ordering.

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. A firm favourite with all our customers. NOW AVAILABLE for CIE/CIU in addition to UK101.
PRICE £7.95

Also now available for 32 × 48 CEGMON-based UK101 BASIC 1 or 5 machines is KAMIKAZE INVADERS – a new slant on this popular game. £5.95

**All the above products (except CEGMON) are available only from Premier.
Phone or write today for our extensive UK101/OHIO catalogue.
12 Kingscote Road, Addiscombe, Croydon, Surrey 01-656 6156**

SCOPYM a single disk copier

SCOPYM provides a fast, foolproof method of creating a new, useable disk from a Master. It will copy the first fourteen tracks of a disk in around 1.25 minutes. All copying is automatic. SCOPYM provides a safe, simple and extremely fast and efficient way of creating a new disk. It is supplied complete with comprehensive notes. For 5.25" OS65D users only.

SOUND/V.I.A. BOARD

The TES II VIA/SOUND kit gives you up to 56 Input/Output lines and progammable sound generation. Inorder to allow you total flexibility in designing your system, 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.

SPECIAL OFFERS

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

SPECIAL OFFERS

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

CHRISTMAS GAMES PACK

Available for the UK101, Superboard, TRS80, Video Genie, SHARP and Microtan, PREMIER's Christmas Games Pack will entertain youngsters and challenge adults. Super Santa, Reindeer Roundup and Toboggan Run. Three-game pack for only £7.95

COMPACT

This useful machine-code program provides UK101/OHIO users with a utility that they have been waiting for – a BASIC line compactor. COMPACT looks at the resident BASIC program and adds lines together wherever possible, thus aiding running speed and saving memory space. It is an extremely reliable way of compacting your program. COMPACT lives at the top end of your memory and will run with any monitor – please state memory size when ordering. PRICE £7.95

TES II HARDWARE RANGE

product	*kit*	*built*
8K RAM BOARD	£29.95	£39.95
8K EPROM BOARD	£29.95	£39.95
8 Slot MOTHERBOARD + PSU	£29.95	£39.95
JI BUFFER BOARD	£19.95	£29.95
MINI EPROM/ROM BOARD	£14.95	£20.95
SCREEN ENHANCEMENT KIT	£55.95	£69.95

PRINTERS from £175.00 upwards

Phone PREMIER for our best price on ANY currently available printer.
KSR PRINTERS ONLY £175 inc VAT – a simple two-wire connection to any UK101/OHIO. RING FOR AVAILABILITY.

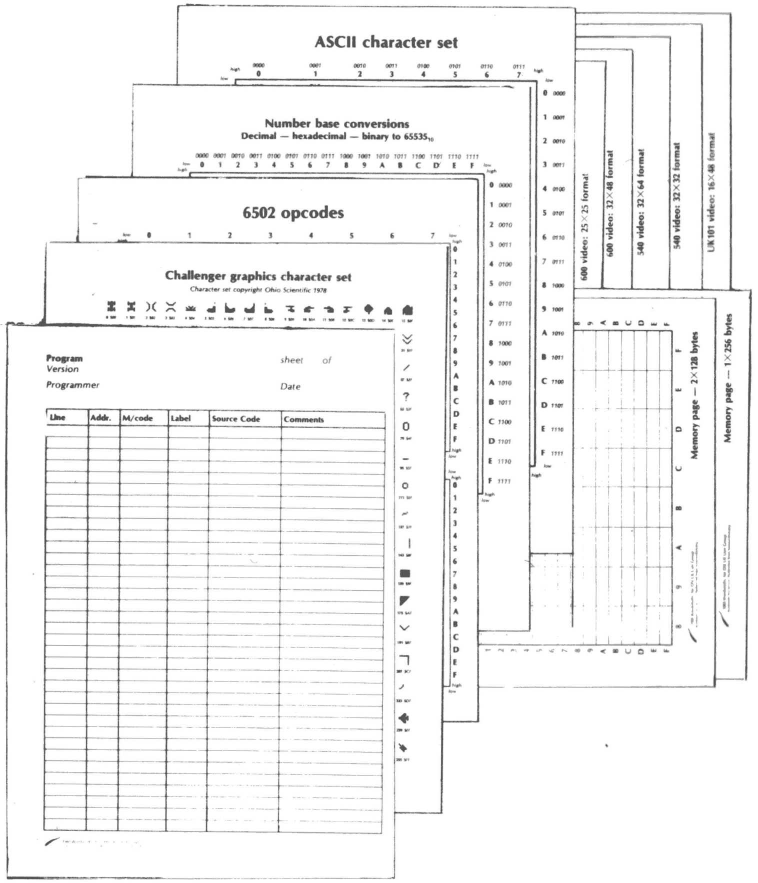
POSTAGE AND PACKING

**Software 60p per order, EPROMS/DISKS 80p per order, HARDWARE £1.50 per item. Maximum £3.00.
ALL PRICES INCLUDE V.A.T.**

from Premier Publications

Planning Cards and Pads

A complete range of planning and programming aids from Wordsmiths. Laminated A4 cards and 100-sheet pads, ready punched for a standard 2-ring binder, at a price that's less than photocopying your own. Bring some order into your programming notes and planning!



Video *(cards and pads)* **Superboard**: 25×25, 32×48, 32×64. **UK101**: 16×48, 32×64. **C2/C4**: 32×32, 32×64.

Programming *(cards and pads)* BASIC, Machine-code/Assembly language, Variables list, Labels list, Memory planner 1 (256 bytes), Memory planner 2 (2 × 128 bytes).

Reference *(cards only)* Hex/decimal/binary conversions to 65535_{10}, ASCII character set, Ohio graphics character set, 6502 opcodes (mnemonics and values).

Cards**65p** each; pads **225p** each, including VAT and postage (minimum order £3.00 please).

Wordsmiths West End, Street, Somerset BA16 0LQ

The Back Page Program

```
10 REM **BACK PAGE PROGRAM**
20 REM
30 REM **TWENTY QUESTIONS**
40 REM
50 REM Adapted from
60 REM "Tales of the Marvelous Machine",
70 REM by Taylor and Green,
80 REM Creative Computing Press.
90 REM
100 PRINT CHR$(26):REM Clear screen
110 PRINT"Welcome to the game"
120 PRINT"of Twenty Questions."
130 PRINT:PRINT"By asking questions"
140 PRINT"which have YES or NO answers,"
150 PRINT"try to guess the object
160 PRINT"I have in mind."
170 PRINT"Be sure to end each question"
180 PRINT"with a question mark."
190 PRINT:PRINT"A simole '?' ends the program":PRINT
200 A$="AEIOUY":CO=Ø
210 CO=CO+1
220 PRINT"Your question number";CO
230 INPUT Q$
235 IF Q$="?" THEN 38Ø
240 IF RIGHT$(Q$,1)="?" THEN 28Ø
250 PRINT"Sorry, that isn't a question."
260 PRINT"Please try again!"
270 GOTO 22Ø
280 ANSWER=Ø
290 FOR I=1 TO 6
300 IF MID$(Q$,LEN(Q$)-1,1)=MID$(A$,I,1) THEN ANSWER=1
310 NEXT I
320 FOR WA=1 TO 1ØØØ*RND(1):NEXT WA
330 IF ANSWER THEN PRINT"YES":GOTO 35Ø
340 PRINT"NO"
350 PRINT
360 IF CO<2Ø THEN 21Ø
370 PRINT"That's the end of this round:"
380 INPUT"Would you like to try again ";Q$
390 IF ASC(Q$)<>ASC("Y") THEN END
400 RUN
```