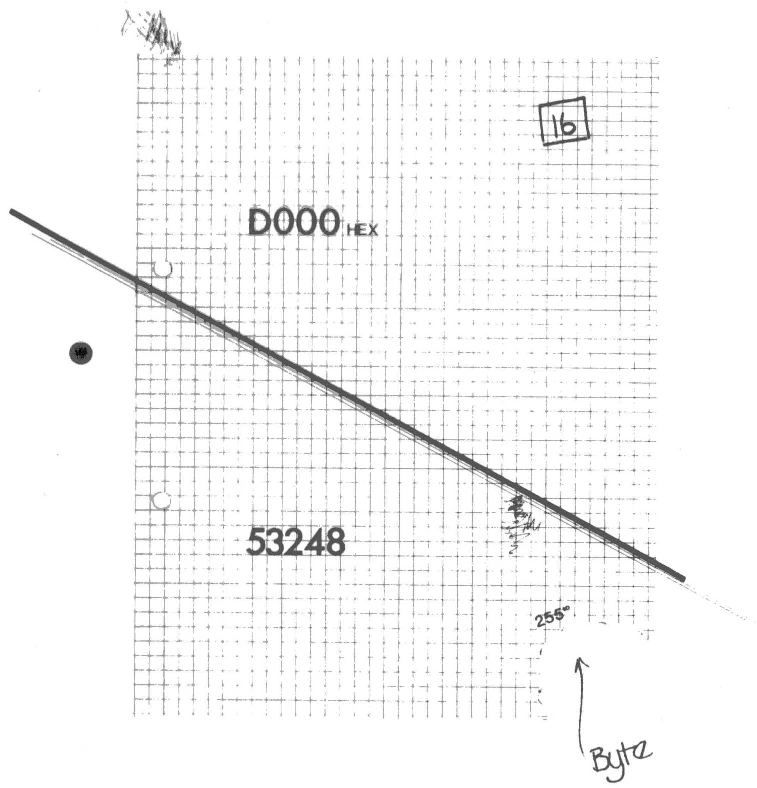
OSI/UK User Group Newsletter

Vol.2 No.1 December 1980



A BASIC guide to machine-code

Notes from members on many matters of hardware and software

Editorial

It's a year later. We've done a lot, too. The worst bugs in the BASIC-in-ROM documentation have been resolved; a variety of 'fixes' for the Superboard's limited screen display have gone out; the 'garbage collector' now clears up string garbage instead of creating it; and the new monitor is out and doing its job. We're delighted, too, with the response to the 'new deal' for the Newsletter — a higher sub but more issues — in that we've had around a 70% resubscription so far, with more every day, and we're still almost a month from publication date.

We know it's a lot — but we give a lot, too. Remember, too, that the monthly mags like PC and PCW would cost something like three times as much as they do if they didn't have that mass of advertising . . . and their circulation is fifty times the size of ours!

But we haven't time to be complacent: there's a vast amount still to be done. For the disc users — we're coming round to your needs at last, especially as Richard and George split a C8-DF upgrade kit between them, to give them each a single 8" disc system. For the hardware hacks — interfaces, internal features and tricks, and of course colour (on which a major feature next issue). Not forgetting the relative beginners — hence the BASIC-to-machine-code series starting this issue. And everyone else, of course.

For which reason we tend to be somewhat slow in answering queries! The Group is *supposed* to be a spare-time occupation . . . and since we're becoming involved in a local (Somerset) set up of Dave Tebbutt's *ComputerTown* computer-literacy scheme, there's still less of that spare time. We do try — but miracle workers we ain't!

Most of our time goes into producing this Newsletter, and into producing it on time. . . So, for those who tend to be worried by dates and things, the next issue is due out on 22nd February 1981, to be followed in April, June, August, October, and round to December again. Mark that on your calendars — and we'll see if we *can* actually keep to those dates!

In the meantime between the issues, we do try to answer your queries as fast as we can; but it does help us if you direct the query to the right person. There is no point in asking me (Tom Graves) questions on hardware — I do actually know one end of a soldering iron from another, but that's about my lot! Documentation and software are more my line → particularly systems software. George, on the other hand, is happier with applications software, and on hardware can probably recite the functions of the 74LS TTL series by heart. So:

Documentation/software:
Tom Graves, 19a West End, Street, Somerset, BA16 0LQ.
Phone (evenings only, please): Street (0458) 45359.
Hardware:
George Chkiantz and *Richard Elen,* 12 Bennerley Road, London SW11 6DS.

Documentation Corner

Speeding up GOSUBs

A comment from *Peek (65)* magazine: GOSUBs are found in the program by scanning every line number from the beginning, each time the GOSUB is called. The nearer they are to the beginning, the faster they will run. *Peek (65)* gave the following example:

```
  1  GOTO 60000
  2  K=K+1: RETURN
  60000  FOR I=1 TO 1500: GOSUB 2: NEXT I: END
```

takes about 10 seconds to run. But the more usual way of doing it:

```
  1  FOR I=1 TO 1500: GOSUB 60000: NEXT I: END
  60000  K=K+1: RETURN
```

takes about 30 seconds to run! As they suggest, "any subroutine called frequently or called from a loop should be located near the start of the program. Put that monster of an initialisation routine at the end next time".

Quotes in text INPUT

A couple of machine-code utilities from *Steve Graham,* written for a C1 under CEGMON. The first allows carriage-return on its own to be treated as a null input, so that the program does not halt. Strings input an empty string, and numerical values get 0. It is activated by pointing the input vector at it, i.e. POKE 536, 64: POKE 537,2 if the routine is located at *$0240.*

```
2046FB   NULL      JSR  $FB46       ; Input – $FFBA under SYNMON
C90D               CMP  #$0D        ; =(CR)?
D009               BNE  RETURN      ; if not, then return
E000               CPX  #$00        ; first char?
D005               BNE  RETURN      ; if not, then return
A020               LDY  #$20        ; put space char
8413               STY  $13         ;    into buffer
E8                 INX              ; increment pointer
60       RETURN    RTS
```

A couple of routines on similar principles, this time to allow commas and colons to be included in string input [resolving *Jack Pike's* problem of Newsletter 2 — Ed.]. It is done by replacing each comma by a right-hand square bracket on input, and re-replacing the bracket by a comma on output (and similarly with colons and left-hand brackets). The result is that, as far as the user is concerned, he types a comma and sees a comma displayed on the screen, but the computer thinks it has got a bracket instead. (Of course, if you are perverse enough to want to *use* square brackets, you will have to change your commas into something else).

```
2046FB   NEWIN     JSR  $FB46       ; input
C92C               CMP  #',         ; if comma
D002               BNE  NCOMMA      ;
A95D               LDA  #$5D        ; replace by ]
C93A     NCOMMA    CMP  #':         ; if colon
D002               BNE  NCOLON      ;
A95B               LDA  #$5B        ; replace by [
60       NCOLON    RTS
```

This time it is necessary to change both the input and output vectors. If the first routine is located at *$0240* then the second will start at *$0250* so the vectors should be changed thus:
POKE 536,64: POKE 537,2: POKE 538,80: POKE539,2

To prove that there is always more than one way to handle any problem, *S.A. Smith* sent us this rather elegant solution to the same problem, this time on a UK101 under Comp's 'New Monitor'. This works by forcing the first character in the string input to be a ", after which any delimiters can be used except another " (which would have to be done by the substitution method shown above).

```
8A        START    TXA                ; check buffer empty
D0 05              BNE   CONT         ;
A9 22              LDA   #'"          ; forces quotes
85 13              STA   $13          ; store as first in buffer
E8                 INX                ; inc. char counter
20 AC FB  CONT     JSR   $FBAC        ; input routine
60                 RTS                ; exit to BASIC
```

The address of the input routine must be the correct one for your monitor: the one shown is as for Comp's New Monitor; for SYN600 use $FFBA, and for CEGMON use $FB46. The input vector at $0218/9 must be changed to point to the start of this routine. You would also have to use this to reload such inputs if they were stored on tape, of course. (C2/C4 systems will need CEGMON before they can use these routines — under SYNMON they do not have user-defined input/output vectors).

A brief note on a tape-viewing routine, from *George Chkiantz*: The following 'View' routine allows you to view tapes without putting them into memory; it runs happily at 300 baud, but is too slow to run for 4800 baud interfaces.

```
10  ACIA=64000: REM 64512 on C2/C4 and C1E
20  ? CHR$(PEEK(ACIA+1));: WAIT ACIA,1: GOTO 20
```

Disc Notes

Block Delete for 65U

Alan Garrett writes: OSI offer their standard combination of inspired idiocy as a block delete function. This one actually works and doesn't crash your program!

Key in the routine as a program and SAVE it with a relevant name (BLKDEL?). Load it on top of the program to be 'manicured' using the FLAG 13 method.

Then give it the two line numbers between which the main program is to be deleted. The routine will print *a)* amount of memory saved in bytes; *b)* old end address of program; *c)* new end address of program; *d) two* values to be POKEd into BASIC so that the system knows how long the program is. These two values should be POKEd in as one line in immediate-mode, and cannot be POKEd in by the routine, as horrible things happen!

```
60010  INPUT"Line numbers please ";L1,L2: IF L1>L2 GOTO 60010
60040  D=6*4096+1: ST=0: FI=0: DO=0
60045  PTR=PEEK(D)+PEEK(D+1)*256
60050  LN=PEEK(D+2)+PEEK(D+3)*256
60060  IF LN=L1 THEN ST=D: REM find lines
60070  IF LN=L2 THEN FI=D
60080  IF LN<60000 THEN DO=D: D=PTR: GOTO 60045
60090  EN=D: IF ST=0 OR FI=0 THEN PRINT "Error ": GOTO 60010
60100  OS=FI–ST: D=FI: REM amount of memory saved
60105  FOR Z=1 TO 10000: REM dummy FOR/NEXT
60110  PTR=PEEK(D)+PEEK(D+1)*256
60120  X0=PTR–OS: X1=INT(X0/256): X2=X0–(X1*256)
60130  POKE D,X2: POKE D=1,X1
60135  IF D<>DO THEN D=PTR: NEXT Z
60140  FOR I=FI TO EN+1: X=PEEK(I): POKE(I–OS),X: NEXT I
60150  POKE EN+1–OS,0: POKE EN–OS,0
60160  PRINT OS: REM memory saved
60170  X=PEEK(122)+PEEK(123)*256
60175  PRINT X: X=X–OS–588: PRINT X: REM see note below!
60180  X1=INT(X/256): X2=X–X1*256
60185  PRINT"POKE into 122 ";X2;" and into 123 ";X1
60195  CLEAR
```

Beware of the number in 60175 — here 588. This is the length of the delete routine, and will effectively change according to the number of spaces or REMS you leave in the routine's code. It must be adjusted to the actual value for the routine to work correctly (otherwise BASIC will muddle up the end of the program with the beginning of variable-storage space).

65D fast screen clear

The BASIC-in-ROM fast screen clear from *Aardvark* (published in Issue 2), which points string storage at the screen area and then 'stores' strings of blanks into it, will also work on C1s and C2s under 65D. There is one difference: the string pointers are at 128^{10} and 129^{10}, not 129 and 130, so the routine as published will need to be changed slightly to suit.

Moving the 65D directory

Another note from *Aardvark*: "One of the most damnable things that the OSI programmers did was to put the directory track right in the middle of the 5" disc. Instead of getting 39 tracks of file space, you get 10 on one side of the directory and 26 on the other" (Aardvark having moved the BASIC utilities onto just two tracks!).

The disc first has to be set up with its DIRectory, CREATE and DELETE utilities set up to call track 39 (or whatever track you want to use) instead of track 12. (i.e. wherever it says DISK!"CA 12, . . ." change it to DISK!"CA 39 . . ."). Do the same for the SAVE functions. (If you don't do this, you won't be able to use the disc, of course!).

The 65D machine-code then needs a little tweaking to complement the changes. Boot up, EXIT, and call EM. Then:

```
!CA 4A00=01,1
@4DC4 /12 39   type (new track no.) to replace 12
!SA 01,1=4A00/8
!CA 4A00=05,1
@513A /12 39
!SA 05,1=4A00/8
!CA 4200=06,1
@42B8 /12 39
!SA 06,1=4200/1    note — one sector only
!CA 4200=12,1    move directory information to new track
!SA 39,1=4200/1
!CA 4200=12,2
!SA 39,2=4200/1
!CA 4200=12,3    and move GET/PUT overlays
!SA 39,3=4200/1
!CA 4200=12,4
!SA 39,4=4200/1
```

Intermittent bug in 65U PRINT

A note from *Paul Nikiel*: if you want to skip over a couple of fields and then start PRINTing, you would attempt to use something like:

```
    INDEX<1>=1000: INPUT%1,A: INPUT%1,B: PRINT%1,X
```

This does not always work. The PRINT statement may not be executed although no warning message is given and control drops through to the next statement. One way round it is to reset the INDEX between the last INPUT statement and the first PRINT statement: INDEX<1>=INDEX(1).

Keyboard PEEK in 65D

A note from *David Dade*: When you want to use the keyboard routine at $FD00 to 'GET' a key value, PEEKing (531) won't work under OS65DV3, as that location no longer contains the key value by the time the keyboard routine has wandered back to BASIC. But the value required is still at 9059_{10}; so a possible BASIC routine would be:

```
5000  REM get key value
5010  POKE 8955,43: POKE 8956,37: REM must be on same line
5020  X=USR(X)
5030  K=PEEK(9059)-48
5040  IF K<0 OR K>9 THEN 5000: REM in this case only allows values 0-9 digits
5050  RETURN
```

Note: Line 5010 loads the X=USR(X) start locations with the start address of the OS65DV3 keyboard routine (not at $FD00 to start with).

A cassette problem with 65D

A couple of problems with 65D from *Tony Restall*: Prior to getting the disc drive for my Superboard I used a cassette for storage and I now want to put some of these programs onto disc from the tapes. So to the problem: how do you do it? According to the documentation on the disc drive all that is required is to type INPUT#1 and it should input from the ACIA input port. But this does not happen as the error code tells me that I cannot use INPUT in the immediate mode. I tried typing EXIT then INPUT#1, to which I got "ARE YOU SURE"; "Y" I said, and initialised track one of my command disc.

Problem 2: I you load a program, through the keyboard, in machine-code, how do you save it on disc? The only way that I can see is to return to BASIC by hitting the BREAK key and warm-starting, but this alters the machine-code program. Any help on these topics would be appreciated.

[*Ed.* — INPUT#1 is for data, not programs; as with ROM BASIC, you can't enter a new source-code line through INPUT. When writing CEGMON we had to do something like this to pass information from a C2 to a C3, by fiddling the input/output pointers (more on this next issue); but the problem here would best be resolved, I think, by using a small BASIC or machine-code routine to scan the ACIA and pass its information to the 65D BASIC input routine. The second problem would appear to be resolved by the DOS command SAVE, which saves memory direct to a specified track of the disc, but I'm none too sure of this. Suggestions, please?]

Assembler Notes

Here is a note from *Roger Pfister* that should warm the hearts of those who use the cassette version of the assembler. The bad news is that it does not make the assembler produce object code in the normal OSI SYNMON tape format rather than the accursed checksum (volunteers, please!), but it does allow you to merge source code files which have the same line numbers — something that can scarcely be avoided with the luxury Resequencer provided by our friends in Ohio.

The machine-code shown below is placed at the start of the assembler's work-space — a good place to put extra routines like this and/or the Elcomp CONTROL-C routine (which halts an assembly in mid-stream). This is at $1391 and does not allow you to retain the checksum 'bootstrap' loader used by OSI to load the assembler itself. Our views on this are probably well known by now, and we suggest that you save the modified assembler using a straight hex dump. Otherwise you will have to relocate the loader or waste valuable space.

A useful trick with the OSI assembler is that any lines numbered 0 are placed sequentially after the last Printed line in the work-space. The modification works by interrupting the call to $0A84 to check the Load flag. If this is set it forces a null ($00) into $00E7 which fools the assembler into thinking that the line number being loaded is 0. Thus, once the complete source code to be merged has been loaded in, normal operations and line numbers on the combined file can be resumed using Resequence.

```
$1391    48                      PHA              ;
$1392    A9 00                   LDA  #$00        ;
$1394    CD 03 02                CMP  $0203       ; (load flag)
$1397    F0 02                   BEQ  $1392       ; (EXIT)
$1399    85 E7                   STA  $E7         ;
$139B    68           EXIT       PLA              ;
$139C    20 84 0A                JSR  $0A84       ;
$139F    4C 1C 03                JMP  $031C       ;
```

The following locations will have to be changed in the assembler itself:

Change	*From*	*To*	*Code*
$0319	20 84 0A	4C 91 11	JMP $1391
$12C9	91	A2	(new start of work-space)
$12FF	91	A2	(new start of work-space)

This mod can be turned off by using the monitor to change the contents of $1399 from $85 to $A5. In use, Print to the line number you wish to be followed by the new section, then load the required section from tape. Though the tape's line numbers will be output to the screen, they are not in fact in the source file. When the tape has finished loading, turn off Load, and Resequence. Although it is possible to use most of the assembler functions with lines numbered 0, beware — others (like Delete) have a will of their own!

Here are two useful locations relating to the assembler:

```
  $1311    JSR $FFEB    gets a character (monitor INVEC)
  $1333    JSR $FFEE    prints it (monitor OUTVEC)
```

Armed with this information it is easy to add useful routines to halt printout or to implement backspace (users of CEGMON will know that its editor and backspace both work on the assembler as is). To retain compatibility with CEGMON we recommend that you don't use the limited 'free RAM' from $0222-$023F for these routines, but place them at the beginning of the assembler's source file, changing $12C9 and $12FF to point to the new start (as shown above) as required.

Another point that has been pointed out by several members, and has caught out several others, is that the assembler uses the IRQ vector at $01C0 (via the BRK opcode, $00) as a jump to warm-start on error-recovery. This means that if you use this area — in testing a routine with CEGMON's breakpoint handler, for example — you will have to re-install the correct machine-code at $01C0 before restarting the assembler, or nasty things may happen! The same applies to ExMon, by the way.

CEGMON Notes

Judging by the number of notes and comments we've been sent, CEGMON is evidently 'relieving our members firmware headaches' as promised — it's nice to know that our hard work is appreciated! A number of common points and confusions still turn up, though, of which the most important is about 'BASIC unmask'. If you wish to use the CEGMON control characters to clear the screen or whatever in BASIC's immediate-mode, BASIC's 'masking-off' of most control characters has to be bypassed. To do this, you either have to replace the BASIC 1 chip with an EPROM with the masking code removed (it occurs around $A357), or else you have to direct the input vector to a routine that bypasses the masking code. It was the latter solution we used with CEGMON. *But* although BASIC then permits control codes — like CONTROL-Z, to clear the screen — to be input and stored in the input buffer, it does *not* recognise them as valid characters. Typing CONTROL-Z in the immediate-mode with the masking removed will clear the screen; but the next RETURN will cause a SYNTAX ERROR message to be printed. A number of people writing in gave ingenious solutions to this, including one to check for control characters and to avoid them being placed in the buffer. However, we feel that the easiest and simplest solution is to use the backspace — CONTROL-Z RUBOUT — and delete the control code once it has done its job.

Another minor problem that has arisen is that in using the machine-code load routine, it is possible that location $0000 may have its data corrupted. As with SYNMON, this is because CEGMON's load routine can fall into the data mode if a particular sequence of glitch characters occurs; and others (such as commas or line-feeds) could even increment the address. This is inconvenient if you are loading machine-code patches to BASIC and the result is that BASIC won't warm-start. Obviously if this happens, restore the first few page-zero locations ($0000-$0002 should read 4C, 74, A2 respectively) before calling warm-start. However, the chances of this happening will be considerably reduced if you type a ROM address like $A000 as the current address before you Load. Just let it try to corrupt that!

A couple of people have asked why backspace in CEGMON was made to correspond to $5F instead of the true ASCII $7F. The reason is that of compatibility: OSI have based all their major programs round the old Teletype bodge of SHIFT-O as 'underscore-as-delete', which meant that we had to follow suit; RUBOUT was decoded in this way so as to make it easily accessible regardless of the state of shift-lock or shift keys.

One additional compatibility problem has confused a few C1E users. The Mutek C1E conversion effectively converted the C1 into a C2 with a limited-width display; the width limit was set manually by answering '48' to 'TERMINAL WIDTH?'. This meant that programs had to be LOADed and SAVEd with POKE 15,72, or else lines might be truncated by the input routine. With CEGMON, there is no need to answer '48' to TERMINAL WIDTH?' — the screen width is controlled by the new firmware. So leave it alone! And as with the original C1 and C2, answer 'TERMINAL WIDTH?' with a simple RETURN.

In Issue 2 we mentioned a method of loading machine-code directly after a BASIC tape: the procedure is slightly different under CEGMON. As before, type: SAVE RETURN LIST (turn recorder on) RETURN (stop tape when done) ?"POKE 251,1: POKE 11,136: POKE 12, 249: X=USR(X)" (restart recorder) RETURN (stop tape when done). *Notice that the POKEs are different from under SYNMON*: it restarts during the LOAD at the beginning of the command-mode execution loop, without clearing the machine-code "load" flag (251_{10}). Now put the machine-code you want onto the tape. When you LOAD the tape from BASIC, it will load the BASIC program, switch to monitor mode (without clearing the screen) and load the last part of the tape.

The BASICs of machine-code

Over the past year we've had a fair number of comments on the lines of "All this machine-code information you publish is fine: but I don't even know where to start, so it's no use at all". Most people get on reasonably quickly with BASIC — it is, after all, designed to be a beginner's language — but without some basic idea of its concepts and terms, machine-code programming just can't make sense. The standard books on 6502 programming, such as Zaks' *Programming the 6502* and Leventhal's excellent *6502 Assembly Language Programming*, do assume that you know rather a lot about the subject before they start — or else, like Zaks, tend to confuse rather than help! So for the next few issues we'll be running this series on basic principles and practice of machine-code programming, to be based on, and linked with, OSI's BASIC. We'll assume no knowledge of machine-code at all — so bear with us if you do know the basics! — but we will assume that you have a working knowledge of programming in BASIC, for games and the like rather than for complex mathematical juggling.

BASIC is described as a 'high-level language' for programming computers. But it's high-level only in the sense that you don't have to think much about where things are — the language translates things like PRINT statements for you, for example, without you having to know what is being PRINTed to, where it is in the system memory, how it is accessed, or anything of the 'lower' level at all. But precisely *because* it handles these things for you, the language itself limits the flexibility with which you can use the system. It simplifies programming for some kinds of work — in BASIC's case, for basic mathematical functions — while making others, such as text-matching and fast graphics, well-nigh impossible. To do these elegantly and, above all, fast, they need to be done at a lower level — the machine-code level.

BASIC itself, of course, is made up of these lower-level machine-code instructions. In ROM BASIC, these can be seen by PEEKing the values stored in addresses 40960 to 49151 (we'll be dealing next issue with the meanings of the values you'll find there). Each BASIC instruction consists internally of a long trail of routines and subroutines — every POKE has to be checked such that its value doesn't exceed 255, or its address exceed 65535, for example — which is why clearing the screen by POKEing it with blanks is so agonisingly slow. A single POKE takes about 15 milliseconds; the equivalent LDA/STA machine-code instructions take about 15 *micro*seconds — a significant difference!

Working in machine-code, however, can be more than a little daunting — particularly if your only way of working with it is through OSI's next-to-useless monitor in the standard SYNMON, SYN600 and UK101 monitors. As you'll know, pressing 'M' in response to the reset *'D/C/W/M?'* prompt gives you a four digit number (an address) and a two digit number (the value at the address) in the top left of the screen — and that's your lot! OSI do publish a cassette version of their very good Extended Monitor (ExMon) — supplied as standard with UK101s, otherwise about £8 from most dealers. Otherwise we would recommend the CEGMON monitor (not least because we wrote it!) — its machine-code monitor is designed to be compatible with ExMon, if somewhat less flexible, and it does have the advantage of being built-in to the ROM monitor rather than a five-minute load from cassette each time. We will deal with the use of the standard SYNMON monitor, but for simplicity most of our examples will use the ExMon or CEGMON formats.

The difficulty with machine-code programming is that while the numbers in that four-digit section are relatively easy to understand — namely a system address — the two-digit numbers can mean anything, dependent upon context. The number *4C* (and it *is* a number, a number in hexadecimal notation) could be a simple number (in other words 76 in decimal); part of a larger number stored in a number of bytes; an index, or part of a look-up table; an address, or part of one; an instruction (JMP — 'jump' to the address defined by the following two bytes); the letter 'M' in the ASCII code; or all manner of other things. It all depends on its context; it all depends on where it is. Hence a couple of tools we'll describe in more detail next time: the assembler, which allows you to assemble machine-code instructions into a program with some semblance of clarity; and the disassembler, which allows you to disentangle a string of hexadecimal machine-code values into something reasonably decipherable. Both are essential for serious machine-code work; but for the smaller routines we'll be starting off with they aren't all that important. OSI's assembler, for all systems including the UK101, is available from most dealers at about £20; while there's a disassembler already built into ExMon, and we'll also give a listing for a disassembler in BASIC in the next part of this series.

The other catch is that almost all machine-code work is done in hexadecimal — counting in sixteens — rather than the decimal that BASIC uses. 'Hex' takes a little getting used to, but once you do know it, it will make the layout of your system a lot easier to understand. If you've ever wondered why your screen memory, for example, starts at such an odd number — 53248 — it's because *in hex* it is actually a nice round number: D000. Decimal numbers count from 0-9 before adding a 'one' to the next row to the left; hexadecimal counts in sixteens, from 0-9, then A-F (F is thus fifteen in decimal), and only then adds one to the next row.

The largest number a single eight-bit byte can handle is 255_{10}, or one less than 256, 2^8 (2^8 'overflows', leaving an effective 'all-zeroes' in the byte). Describing a full eight-bit byte in decimal is tedious and misleading, especially as the 6502 processor has a separate BCD (binary coded decimal) 'decimal' mode of operation which doesn't quite appear as true decimal on the outside. Counting in whole bytes, in 256's, is hopelessly impractical; but half a byte (nicknamed a 'nibble'!) holds up to one less than 2^4, or up to 15 in decimal (the processor counts from 0 — thus there are actually sixteen possible numbers in a nibble). We can thus represent the contents of any byte by two hexadecimal numbers: $AF=175_{10}$, $4C=76_{10}$. To reduce confusion, particularly when using an assembler, the convention for 6502 or 6800 processors is to prefix a hex value by a 'dollars' sign: *55* is fifty-five, while *$55* is eighty-five (5×16, +5) in decimal. (Z-80 assemblers use a different convention: hex values have an 'H' suffix, thus *55H*). *Don't* use the $ prefix when working with the machine-code monitors, though; they only know about hex, and will only get confused if you try to tell them otherwise!

Writing in machine-code is rather like writing in a very tightly limited BASIC. You have only three variables which you can use directly — the main registers A, X and Y — although you can operate on the rest of the memory with a very restricted range of functions. You have a very limited set of commands and functions — no large-scale functions like * or MID$, only the ability to copy or change the contents of the registers or of single memory locations. So programs have to be worked out in far more detail than they would in BASIC.

However, it's often easier to plan a machine-code program by writing it in a kind of 'dummy-BASIC'. The system's A register, or Accumulator, is used by almost everything — a kind of high-speed messenger. The other two main registers, X and Y, tend to be used as counters or pointers — hence their description as 'index registers'. If we treat these as BASIC variables, their use may become a little more clear:

Machine-code instruction	BASIC
LDA	A=...
LDX	X=...
LDY	Y=...
TAX	X=A
TYA	A=Y
INX	X=X+1
STA	...=A or POKE ...,A

We then get into the subtle game of the 6502's 'addressing modes'. Unlike the Z-80, which has different instructions for everything, the 6502 and 6800 processors use the same instruction in different ways, to increase programming flexibility while (in principle at least) maintaining some semblance of comprehensibility. Thus the LDA instruction has a number of variations:

LDA #$20	'immediate'	A=32
LDA $20	'zero-page'	A=PEEK(32)
LDA $0220	'absolute'	A=PEEK(2*256+32*1)
LDA $20,X	'zero-page indexed'	A=PEEK(32+X)
LDA$2020,Y	'absolute indexed'	A=PEEK(32*256+32*1+Y)
LDA ($20,X)	'indirect indexed'	A=PEEK(PEEK(32+X))
LDA ($20),Y	'indexed indirect'	A=PEEK (PEEK(32)+Y)

The last two may seem unnecessarily complicated, but they are used a great deal to simplify the handling of look-up tables and the like. We'll be using a 'STA indexed indirect' for a fast screen-clear routine at the end of this article.

Most programming uses loops, conditional loops and branches. BASIC does these with FOR:NEXT, IF/THEN, ON/GOTO or ON/GOSUB, GOTO and GOSUB. In machine code there are equivalents, but they sometimes work in a slightly different way. In particular, the IF/THEN tests are done against particular bits or 'flags' in a separate 'status register', referred to as the 'P' register; the much-used BEQ and BNE (branch if equal / not equal) tests check if the top bit in that register is set or clear respectively, for that bit is set if the last LD.. (or some other 'move' instruction like ROL or INC) resulted in a zero value.

JMP $0020	GOTO 32
JSR $0020	GOSUB 32
BNE ...	IF (last value)<>0 THEN GOTO ...
BEQ	IF (last value)=0 THEN GOTO
JMP ($0020)	('jump indirect') GOTO PEEK(32)

The last one is something you can't do in BASIC! It's used by the SYN600 and CEGMON monitors to pass BASIC's input and output 'vectors' through RAM, to allow you to drive your own devices directly from BASIC — but more on that in a later part of this series.

The 'branches' — BEQ, BNE, BPL, BMI, BCS, BCC, BVS, BVC — we'll mostly leave till later, but there is one point about them which is important. A JMP or JSR point to anywhere in memory; a branch like BNE can only jump within a limited range — back 126 to forward 129, to be precise. The branch instruction is followed by a one-byte 'displacement' — which can only have a value up to 255, being a single byte — which used the top bit of that value as a 'sign' flag, top-bit set meaning '—' or back. The maximum range is –128 to +127; but this is from the next byte after the branch op code and the displacement-value byte, so the effective range is –126 to +129 from the branch op code itself. It takes a little getting used to!

For demonstration, we'll show one of the most called-for routines — a fast screen-clear. In OSI's BASIC, you have two 'official' options: scrolling the screen with FOR X=1 TO 32 : PRINT : NEXT — which is messy and inelegant; or 'POKE the screen with blanks' — which is *slow*. For a C2 the shortest version of the latter would be:

```
FOR X=53248 TO 55295 : POKE X,32 : NEXT
```

which , at a standard 1MHz clock speed, takes a good half-minute. The BASIC version of the fast machine-code screen-clear is rather more tangled:

```
10    A=208
20    Y=0
30    HI=255 : POKE HI,A
```

```
 40     LO=254 : POKE LO,Y
 50     X=4 : REM on 2K screen memory (C2/4, C1-E, etc) X=8
 60     A=32
 70     POKE(PEEK(LO)+256*PEEK(HI))+Y,A
 80     Y=Y+1 : IF Y=256 THEN Y=0
 90     IF Y<>0 THEN GOTO 70
100     POKE HI, PEEK(HI)+1
110     X=X—1
120     IF X<>0 THEN GOTO 70
130     RETURN
```

Try running this as a BASIC program — you'll find it will take nearly twice as long as the simple 'POKE the screen with blanks'. The machine code is as follows:

```
A9 D0          LDA #$D0          ;
A0 00          LDY #$00          ;zero index for later
85 FF          STA HI            ;point the LO,HI pair
84 FE          STY LO            ;to the beginning of screen
A2 04          LDX #$04          ;08 for 2K screens
A9 20          LDA #$20          ;ASCII 'space'
91 FE          STA (LO),Y        ;blank the current location
C8             INY               ;bump up current location
D0 FB          BNE -5            ;and loop back until Y 'overflows'
E6 FF          INC HI            ;bump up to point to next 'page'
CA             DEX               ;check last page not done
D0 F6          BNE -10           ;and clear next page if not
60             RTS               ;else return
```

This gives you a subroutine that will clear the screen in about one hundredth of a second — around 500 times faster. The catch is that you have to be able to get at it somehow! Although the routine itself is relocatable — it has no absolute addresses within it — it has to be put *somewhere*. On the assumption that it will be used with BASIC, we can place it in the 'free RAM' area in page 2 (around 576_{10}-760_{10}) — $0240 is probably the safest all round.

Using the standard SYNMON/SYN600 monitor, type . (for address) 0240 / (to input data as hex values) and then each hex pair, separated by 'RETURN' (◁): A9◁D0◁A0◁00◁85 and so on to D0◁F6◁60◁. Each of these values should 'roll' into the two-digit part of the display; the four-digit value (the current address) will go up by one each time you press 'RETURN', and a random value will appear at the same time in the two-digit area, to be replaced by your new value. After you've pressed 'RETURN' after the last byte, the 60, the address counter should read 0257.

With EXMON, type @0240, then each value separated by LINE FEED. With CEGMON, type .0240 then type each value separated by , (comma), LINE-FEED (▽) or RETURN — it's probably simplest to type them in as in the assembly listing, such as:

```
A9,D0▽
A0,00▽
```

Since this is stored in that 'free RAM', it isn't wiped out by BASIC when you cold-start. But it is, of course, lost when you turn the power off. We can presume that you don't want to have to type it in with the monitor each time you start your system! But BASIC can't use hex — it only knows decimal — so we have to go about it another way. One is to convert the whole lot to decimal DATA statements, and POKE them in at the right addresses:

Dealer Notes

Things are moving in OSI...

There's been quite a bit happening on the OSI front these past few months. Some kind of franchise deal has been arranged with CDC, the mainframe giant, to sell OSI equipment — probably as 'CDC's micro' — in CDC's 'retail' outlets in the States. An important side-effect is that CDC are busy converting their *Plato* program suite/database to run on OSI kit from C2-D upwards. This is still mighty expensive — probably well over the £5000 mark — but it's a fraction of the present cost; and when you remember that you can hang over a dozen C1s off the main system, it makes for the cheapest purpose-built schools system ever. Worth watching out for that.

OSI — or rather our old friends American Data — had a stand in Compec at Olympia a month ago. American Data had finally sprouted a UK subsidiary, American Data (UK) Ltd. But since their prices are only minutely lower than before (e.g. £185 for a standard Superboard), we doubt if their market position will much improve. For the same reason, although the dealers were all there — Pete Fawthrop of CTS, Alan Garrett of Mutek and Steve Hanlan of Beaver, to name but three — we rather doubt that they will be rushing back into American Data's 'official' fold.

The much-heralded new C1 Series II was also at Compec. We don't exactly think it will improve OSI's market position either — quite the opposite. It has all the hallmarks of a badly thrown together bodge — a nasty plastic case, a profusion of little add-on and stick-on boards to fix one thing or the other; the serial and parallel ports, colour and sound, while there in principle, are not there in fact, for they aren't actually implemented; the keyboard and garbage collector are as scatterbrained as ever. The colour memory starts at $D400 instead of $E000 (for no apparent reason), which makes a 2K screen memory very much more difficult to implement; and the much-vaunted 12×48 screen format can only run with a software patch, which in any case overwrites the few locations that its still absurdly limited machine-code monitor uses. Documentation is marginally improved, but still diabolical — most of it is for the disc system which wasn't even at the show. The *only* good point was that the BREAK key, although still in the same place, now has a practical safety device — you have to hold it down for a second (presumably for a capacitor) before the BREAK signal goes through. The C1 Series II replaces the existing C1, and costs around a hundred pounds more; the Superboard, at the moment, is unchanged. So the moral of this sad tale is: if you are thinking of getting the old-style C1, get it *now*, before it's too late — because its 'improved' version is certainly *not* worth the wait or the extra cost.

Two further OSI titbits: Barbara Hall has resigned from American Data — no reason given, but I don't blame her with the mess they've made of their operations here. And OSI themselves have been taken over by, a consortium called Macon: we heard this about a couple of days before going to press, so no further details as yet.

...and dealer wrangles at home

The UK dealers themselves have been going through the usual hiccups: *Watford Electronics* parted company from the *Mighty Micro* partnership, followed by Ian Jones, who has set up his own firm, *Mitrecrest* (see below). Assorted affiliations abound — Dave Philips of *Northamber* and *Philbrand* has added a new company to his list, namely *OSI Computers!*

We've always been viewed with great suspicion by Pete Fawthrop of *CTS* — he regarded us as a 'walking advert for Mutek', which we're not (we hope!). We met him at Compec, and we hope we've convinced him otherwise — but we do wish he *would* send us details of what he's doing, so that we could publish them here. We have, however, had repeated comments from members about CTS's very high level of service, even from as far afield as Bristol (CTS are in Lancashire). Keep up the good work — but do keep us informed! Their number is 0706 79332.

On the service front, we've had comments of a different kind about two other dealers. One has an infuriating habit of leaving the phone off the hook; the other, a software house, does have an answering machine, but doesn't seem to play it back very often, and in one case has taken up to two months to send off software that's been paid for. We know that both have had difficulties with expansion of business; but would they please do something about it, or we'll be rather more explicit about who they are next time.

Odds and sods
Various interesting items arrived or on the way. *Beaver Systems* will be importing a programmable character generator early in the New Year: details from Steve Hanlan on 084 421 5020. And the indefatigable Martin Kay of *Zen Computer Services* (061 962 3251) sent us this release:

"48-line compatable 8-in/8-out (PIA) board at £14.00. IBM Selectric interface board for do-it-yourself conversions, requires 8-in and 2 handshake lines from PIA and 50v supply to directly drive solenoids. Uses a software code conversion — suitable for all systems. Board £15.00, software £5.00.

"Coming soon (prototype built and working), a 32K dynamic RAM board (kit). With 16K supplied around £95.00; with full 32K around £120. 8K block select, 1MHz, single 5v supply, 0-6A consumption."

The saga of BASIC 3
BASIC 3 is the chip with OSI's garbage-creating 'garbage collector', discussed in detail last issue. At that time we'd had brief words with *Comp Shop* about doing a piggy-back order along with their current order for masked ROMs for their next batch of UK101s; they seemed amenable to the idea of letting us have them at cost for our members. At Breadboard the other day, however, it seems *a)* the new ROMs won't be here until February, *b)* they haven't ordered any extra, and *c)* they may well have the wrong garbage-collector 'fix' in any case! Ah well, we did try!

But Steve Hanlan of *Beaver* told us that he'd located a manufacturer of the fixed BASIC 3 in the States; and Dave Graham of *Mutek* is keen to produce EPROMs of BASIC 3 and BASIC 1 (for the 'unmask' and perhaps for user-defined keywords). More definite details on this next issue, we hope.

David Richards (Printers and Distributors) Ltd, 61 Hoe Street, London E17 4SA. Tel: 01-521 5231.

Continuous stationery for printout — you know the stuff: drive holes down the side, perforated horizontally. Complete range of standard listing paper from stock, including music-ruled paper and gummed labels. Your own headed notepaper, invoices etc. to order.

Chromasonic Electronics, 48 Junction Road, Archway, London N19 5RD. Tel. 01-263 9493.

We should have had Chromasonic on the list ages ago. Deals mainly in UK101, with a full range of add-ons: 16K Smart-2 RAM kit, printer interface and (shortly) multi-way D-A unit and colour system.

Dola Software, 117 Blenheim Road, Deal, Kent.

We've had several members recommend Dola's software, for C1/UK101, and also Acorn and Research Machines 380-Z. They're also organising a weekend course on micro-computing, particularly for 6502-based machines. It will include plenty of hands-on practical work. The course is on 6-8 February '81, at the Royal Hotel, Deal; cost about £65. Details from Dola's J.M. Leach.

Mitrecrest, 61 New Market Square, Basingstoke, Hants. RG21 1HW.
Phone: 0256 56468 and 56417.

Also known as *Micro Peripherals*, Ian Jones' new company deals mainly in peripherals like printers — currently pushing the new high-speed version of the Base-2 printer, for example.

small-ads

'Breakout' game on tape for Superboard or C1. Bat is controlled by potentiometer, making it very easy to use. 2 simple hardware mods needed. All components ready assembled, tape and full instructions for only £3. J.R. Fletcher, 45 Haydn Ave, Purley, Surrey, CR2 4AG.

TTY ASR28 for sale. Keyboard, printing, perf-tape reader, page printer, 3 handbooks. Working when taken off circuit. £20. Motor-start relay and circuit unit available at £7.50 reinstatement charge. Buyer collects. P. Knight, Sevenoaks (0732) 62227, or 01-242 4433 ext 4287.

EXPANSIONS FOR SUPERBOARD/C1, COMPUKIT. Light pen £12, parallel I/O port. Two versions; simple £18, advanced £35, Eprom programmer £28 (single supply) and more — please sae for further info. Cheques and PO's to B. Minstry, 75 St Margaret's Rd, Bradford, West Yorks. Members of OSI user group *deduct £2* per item. Prices inclusive. All items ready built and tested with good documentation. For kits *add* £2.00 extra. Our primary aim is quality not quantity.

ComputerTown
For those who haven't heard about it, *Computertown* is the UK version of a kind of 'computer-literacy' scheme — a slight misnomer, but never mind. Originally devised in the States by Bob Albrecht and others, it was started up over here by Dave Tebbutt and Peter Rodwell of *Personal Computer World* (see PCW November '80). This seems a very good idea to us; so out in the backwoods of Street, Somerset, we're setting up our own ComputerTown.

The idea is simply to let people play with computers. Not to sell, evangelise, cajole — just play. We happen — crazy fools — to *like* computers, but most people don't. But whether they like them or not, people are going to have to live with them; so it's up to us to make that fact as palatable as possible by showing that computers can be *fun*.

We're fortunate here in that we'll shortly be moving part of our stuff into the shop next door — ideal for a ComputerTown setup. We have at present one C2, one C3, promises of two C1s and of a Sharp MZ-80. We're after more, of course! Anyone with an old Pet they don't want, would you care to contribute it to a good cause?

We hope to be starting up in January at 19a West End, Street, Somerset, initially on Monday evenings only. (The chip shop across the road is shut on Mondays — otherwise have you ever tried scraping congealed fish and chips out of a keyboard?!)

Anyone in the mid-Somerset area interested in helping, please contact Tom Graves at Street (0458) 45359.

User Group Notes

The User Group and UK101s
A lot of people have asked us our 'policy' on UK101s — do we accept 101 users as members? Or are 101s 'enemies'! The obvious answer is, yes, of course we want UK101s with us — more, more!

Our feeling is that the UK101 should not really be regarded as something separate from OSI kit. It uses exactly the same BASIC as the Superboard (early versions even left the author's name unchanged . . .), the circuitry is best described as a 'photocopy' rather than a copy of the Superboard, and the memory map, I/O and very nearly everything else is pretty much the same as that on the C1/Superboard series. It has considerably fewer differences than the C2, for example, let alone the C3 series. And what with the bewildering variety of screen formats in the OSI range — including all the home-brew jobs — there's not that much of a distinction to draw. CompShop may have the cheek to call the UK101 their own design; but as far as we're concerned, it's an OSI machine with a few add-ons. And with that said, we *do* want to know about the use and uses of UK101s as well as the more conventional OSI machines.

In case anyone asks . . .
. . . don't forget to tell them about the User Group! But just to remind anyone who isn't sure, membership/subscription of the Group is now *£10.00* for *six issues*. We're also adjusting everyone's renewal figures so that, next year, all subscriptions will be due for renewal in December, with the beginning of Volume 3 — and yes, we do intend to be around that long!

The restive festive bit
No doubt you're as fed up with the 'run-up to another commercial Christmas'; but this issue happens to coincide with the season, and anyway it may give you something else to think about, other than children's electronic 'computer' toys which forget to warn you about their fragility or their absence of batteries. We hope you'll find this as interesting as the children's annuals! You'll be hearing from us again in February, a month earlier now that the Newsletter has gone bi-monthly.

And best wishes from all of us, of course.

```
FOR X=576 TO 598 : READ Y : POKE X,Y : NEXT
DATA 169, 208, 160, 0, 133, 255, 132, 254
DATA 162, 4, 169, 32, 145, 254, 200, 208, 251
DATA 230, 255, 202, 208, 246, 96
```

A final problem, for this time, is to connect this routine to BASIC in such a way that it can be called direct as a subroutine within BASIC. There is a provision for this, in the USR(X) function. As usual, OSI's description of this in the 'manual' is both garbled and wrong!

USR(X) is used as a means not just of calling a machine-code routine, but of transferring values to it if required. We don't need that part at the moment; the only thing we need to know is that the BASIC statement X=USR(X) jumps to a machine-code subroutine pointed to by two locations. In the BASIC manual we are told that these are 23E and 23F — which appears to be true for disc systems under 65D, but *not* for BASIC in ROM. The right locations are (decimal) 11 and 12. They need a two-byte hex address, in the right order and, of course, converted to decimal — just to make things easier? The right order is the low byte *first*, a peculiarity of the 6502 which apparently allows it to run faster. If the start address of our screen-clear is at 576_{10}, convert it to hex: $0240. The high byte is $02, and goes in 12_{10}; the low byte is $40, or 64_{10}, and goes in 11. Thus POKE 11,64 : POKE 12,2 sets up the USR function to point to our screen-clear at $0240, 576_{10}. Note, though, that although the screen-clear will be untouched by cold-start, these two locations will be cleared then — in fact to point at the FC (function call) error — and will be need to be reset after cold-start.

To set up the screen-clear:

```
10  FOR X=576 TO 598 : READY : POKE X,Y : NEXT
20  DATA 169, 208, 160, 0, 133, 255, 132, 254
30  DATA 162, 4, 169, 32, 145, 254, 200, 208, 251
40  DATA 230, 255, 202, 208, 246, 96
50  POKE 11,64 : POKE 12,2
```

Thereafter, to call the routine:

```
            X=USR(X)
```

Next issue: the disassembler, and a TRACE function for BASIC.

Errata

It seems to be more than about time that we owned up to our mistakes! A number of members have pointed out the following errors which escaped our 'quality control' — thanks!

Issue 1: In the page-zero locations, the pair to be printed in decimal should be $AC, $AD (not $AD, $AE). Also the current cursor location on C1s is at $D300+($0200); the value shown, of $D700+($0200), is correct for C2s. In the remote switching option in the hardware mods, U14 should be U41 — this has been corrected manually in most copies, but some may have slipped through.

Issue 2: On the diagrams relating to the video mod for the Superboard/C1, fig.3, point K should be the plated-through hole just below the track marked. PTH E should be above pins 2 and 3 of the character generator.

Issue 4: Page 3: Ray Fox's first function should end +(X=0), not +(X) as printed. Also Dr Abbott's video mod is essentially correct, an update is in preparation — please wait for it!

Screen 'menus' for the C1

Since I sent you the modified input routine to allow real-time keyboard scanning, I have discovered why it ignores a "RETURN" — this gives a value of 13_{10} (i.e. carriage return without line feed) instead of the expected 96!

With this in mind, and needing a system to input and display array variables in a program to calculate shift duties for a nursing acquaintance, I decided to develop the routine into one capable of supporting a flashing cursor. I have abstracted the lines dealing with the cursor management and screen array input and output, and present them in the hope that they will prove useful to other C1 users. As well as the program I am including some notes on its operation and possible modification.

```
   10  GOSUB 10000
   20  FOR I=1 TO 30: PRINT: NEXT
   30  C=53381: K1=PEEK(C)
   40  X=0: Y=0
   50  POKE C,161
  100  FOR T=1 TO 20
  110  IF T=11 THEN POKE C,K1
  120  X=USR(X): K=PEEK(531): IF K<12 THEN 160
  130  IF K=44 OR K=46 OR K=47 OR K=59 THEN 200
  140  IF K=32 OR (K>47 AND K<58) OR (K>64 AND K>91) THEN 170
  150  IF K=13 THEN 300
  160  NEXT: POKE C,161: GOTO 100
  170  K1=K: K=46
  200  IF K=46 THEN X=X+1: IF X>23 THEN X=0: K=47
  210  IF K=44 THEN X=X–1: IF X<0 THEN X=0
  220  IF K=47 THEN Y=Y+1: IF Y>22 THEN Y=22: GOTO 400
  230  IF K=59 THEN Y=Y–1: IF Y<0 THEN Y=0
  240  GOTO 320
  300  IF X>0 THEN X=0: GOTO 320
  310  Y=0
  320  POKE C,K1: C=53381+X+32*Y: K1=PEEK(C): GOTO 100
  400  END
10000  Y=546
10010  FOR I=64768 TO 64967
10020  X=PEEK(I)
10030  POKE Y,X
10040  Y=Y+1
10050  NEXT
10060  POKE 662,66
10070  POKE 11,34: POKE 12,2
10080  RETURN
```

The base screen address (53381) and maximum values of X and Y (23 and 22 respectively) work on my TV; they may need slight modification for different sets. Limiting Y to 22 leaves the normal print line clear for messages (use *PRINT CHR$(13);"MESSAGE";* to avoid scrolling the screen and confusing the cursor!).

Auto-repeat functions normally on all keys (including the cursor direction keys, although the cursor is not displayed until the key is released), allowing multiple entries in the X-direction and rapid cursor movement in any direction.

Lines 10000 to 10080 copy the monitor input routine into the unused page-2 RAM and modify it (line 10060) to prevent it looping indefinitely until a key is pressed, thus allowing the housekeeping necessary to display a flashing cursor. [CEGMON users: refer to page 16 of the CEGMON handbook for the CEGMON equivalent — Ed.] If RAM space is limited these lines can be entered before the program proper, RUN, and then deleted with NEW, leaving only the page-2 machine-code. Control-C is not disabled as it is when polling the keyboard the OSI way, and will operate normally when the main program is running.

Line 130 and lines 200 to 230 define the cursor control keys (<, >, +, ?) and can be altered if other keys are preferred.

Line 140 defines the valid ASCII values from the keyboard, and lines 170 and 320 poke the selected character to the screen. Modifying K1 between lines 170 and 320 will allow graphics characters to be poked directly from the keyboard to the cursor location on the screen.

X and Y are always valid co-ordinates for the current cursor position and can be used to enter the keyboard ASCII value into an array, A(X,Y).

A phantom cursor using C1=53381+X1+32*Y1 (with X1 and/or Y1 in a FOR:NEXT loop if required) can be used to peek or poke information from or to the screen without disturbing the flashing cursor.

Jumping out from line 170 to a subroutine allows the entered character to be processed (e.g. summed into row and column totals) and the result displayed immediately. *NB* start the subroutine with POKE C,K1 or nothing will appear to happen for a while, and use the phantom cursor to display the results or the flashing cursor will be displaced on returning.

Lines 300 and 310 reset the cursor first in the X direction and then, on a second pressing of the RETURN key, in the Y direction. If preferred, they can be replaced with *300 X=0: Y=0* to do the whole job in one keystroke.

By inserting additional statements (e.g. AND X<>13) into the brackets in line 140 it can be made impossible to enter either figures or letters into specified rows or columns; illegal entries will simply default to the NEXT in line 160. This may be useful in preventing errors when processing the array data. Rows and columns can be reserved in the same way for output rather than input by stopping the cursor from entering them (lines 200 to 230).

By modifying lines 200 to 230 to increment or decrement X and Y by 2 the cursor can be made to jump over a grid of lines previously poked to the screen allowing clear separation of rows and columns at some sacrifice of display space.

Entering a space at the cursor location will delete the character there, and store a 32_{10} in the corresponding array variable. Characters can be over-written simply by positioning the cursor over them and entering the desired new character. These two features make editing and altering the array very easy.

The array corresponding to the screen display can be larger in both directions than the screen itself. Entering a digit corresponding to the desired top left X or Y co-ordinate in a reserved position can easily be made to poke the screen with values from the specified area of the array. The screen thus becomes a window, through which any desired part of the array can be viewed and modified.

Although the cursor controls are specified as <, >, + and ?, it is *not* necessary to use the shift key in conjunction with them to move the cursor, provided the shift-lock key is down as it would be in normal keyboard usage. These keys were chosen for their labels (3 out of 4 are correct!) and spatial layout on the keyboard.

Lines 100 and 110 control the flashing frequency of the cursor, and can be easily modified to alter the flash rate. If modifying this part of the program, don't put too many statements into the T FOR: NEXT loop, or the keyboard scan rate will be slowed down and some keystrokes may be missed whilst the program is busy doing other things. In normal use, the cursor will stop flashing whilst the entered character is stored and processed; keyboard scanning is discontinued until line 100 is reached again, preventing erroneous data entry.

John Attwood

Passing multiple values through USR(X)

As a product of my recent initiation into machine-code, I tackled a problem you referred to in the first issue of the Newsletter, namely a 'fast print routine' requirement. The solution shown here is nearly three times faster than a single POKE to input a complete 'message'. It works by coding the screen address and the message directly following the USR(X) statement in BASIC. For example:

X=USR(X)XY"***

would write *** near the top right hand corner of the screen. For the standard Superboard the origin is at CC, the top left hand corner at CZ, the top right hand corner at ZZ etc. Thus

X=USR(X)GB"TITLE

will, on a standard Superboard, put a title on the non-roll part of the screen at the bottom. For a C2, the alphabetical coding will, presumably, only cover centre of the screen, but the top and right hand edge can be got using [|] ↑ (SHIFT K, L, M, N) and the bottom line and left hand edge with '?'. As the 'message' is entered as part of a BASIC program the graphics characters cannot be included [except by using a BASIC-unmask routine — *Ed.*]. However a slight swindle does allow a few of them to be included, simply be replacing the " (character 3 in the sequence) with a space and allowing the BASIC-token 'coder' to operate on the message. Thus

X=USR(X)CC NOT

will place a bright square — the graphic representation of the BASIC-token value for the NOT operator — at the point C,C.

I make no claim that this is the fastest code that can be devised, as over half the code is needed to reconstruct the screen address from the first two bytes in the sequence. By using less convenient screen mappings or otherwise, I would imagine factors of two or more are still to be had. How about an animation program competition for Christmas?

Get X, Y co-ordinates and transform to screen locations:

```
A0 00          LDY  #$00         ;  Y=character counter
B1 C3          LDA  ($C3),Y      ;  get 1st letter — X co-ordinate
18             CLC               ;
69 BF          ADC  #$BF         ;  convert 'A-Z' letters to 0-25 values
85 E0          STA  $E0          ;  temporary 'X' store
C8             INY               ;  set to get next char — Y co-ord
A9 5F          LDA  #$5F         ;
18             CLC               ;
F1 C3          SBC  ($C3),Y      ;  convert 'Y' letter to Y value
85 E1          STA  $E1          ;  temporary Y-coordinate store
4A             LSR  A            ;
4A             LSR  A            ;
4A             LSR  A            ;  (effective multiply-by-32)
18             CLC               ;     (to add line value to high byte)
69 D0          ADC  #$D0         ;  add $D0 for high of screen address
85 E2          STA  $E2          ;  store screen hi in $E2
A5 E1          LDA  $E1          ;  get 'Y' from $E1
```

```
0A             ASL  A            ;  (multiply by 32)
0A             ASL  A            ;     (for low byte of
0A             ASL  A            ;     (screen address)
0A             ASL  A            ;
0A             ASL  A            ;
05 E0          ORA  $E0          ;  Add 'X' co-ord for final lo address
85 E1          STA  $E1          ;  store screen low in $E1
C8             INY               ;  skip 3rd char (" or space)
```

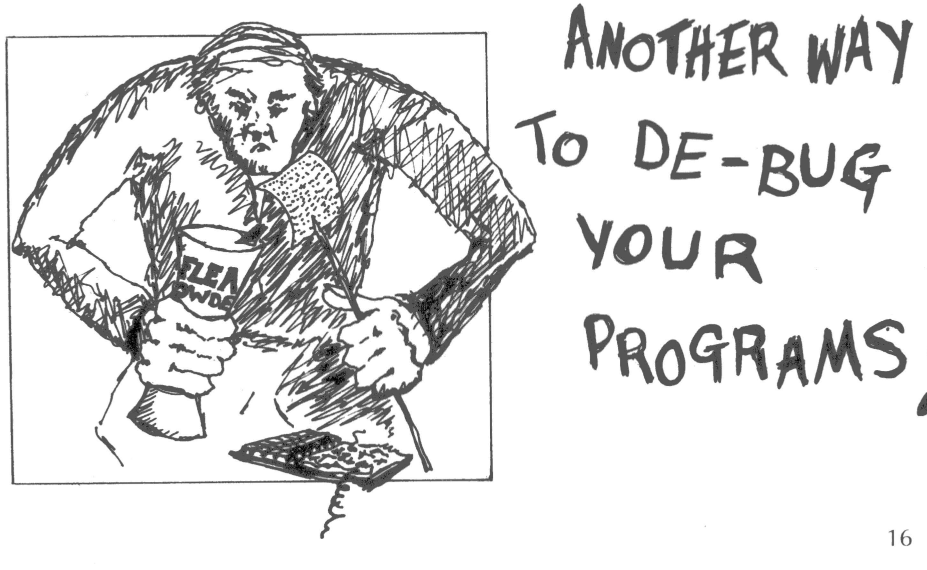
Transfer message to screen locations:

```
C8             INY  NEXT         ;  set for message char(s)
B1 C3          LDA  ($C3),Y      ;  get next char from BASIC line
F0 04          BEQ  EOL          ;  exit on null (end of BASIC line)
91 E1          STA  ($E1),Y      ;  store direct to screen
D0 F7          BNE  NEXT         ;  always jump to get next char
```

'Fix' BASIC pointer to ignore 'message' and co-ordinate characters:

```
98       EOL   TYA               ;  no. of message+co-ord chars
18             CLC               ;
65 C3          ADC  $C3          ;  calculate new pointer-lo
85 C3          STA  $C3          ;  and replace
B0 02          BCS  EXIT         ;  update $C4 pointer-hi
EC C4          INC  $C4          ;     if necessary
60       EXIT  RTS               ;  and return
```

In the first issue also, a number of routine entry-points were given. Although some particular error code routines are listed, more useful is the entry point at $A25F. This routine prints the contents of the accumulator (A-register) followed by ERROR IN (BASIC line no.). Using this it is easy to give machine-code routines their own single letter error warnings.



Self-starting programs

This technique originated simply as a means of appending a RUN command to a SAVEd program but has now developed so that it provides a means of loading machine-code, setting storage limits, setting screen widths and in general performing initial housekeeping for a program without using run-time code space. I keep the code shown in printout 1 on a cassette and when ready to SAVE a debugged program I load it in behind.

To SAVE the program I enter RUN 63900, which causes the following to happen. 63900 picks up the end-of-text address. The loop formed by 63910-20 scans backwards down the program replacing all '%' (CHR$(37)) and the single '!' (CHR$(33)) with Carriage Returns (CHR$(13)), until the '!' has been processed. 63930 then delays a nice long time to allow the lengthy leaders on the salvaged Data General software distribution cassettes I use to pass by, then initiates a normal SAVE procedure; the whole lot, CRs and all, now gets written to the tape up to the EOT address which is unchanged.

When the tape is LOADed everything is normal up to and including line 63950. After this every line number is immediately followed by a CR and is therefore ignored, but the rest of the line (except for the special case of 63998) is treated as an immediate command string and executed. 63960/61 between them load a screen-clear routine into Hex 222 and up, using the subroutine in 63940/50 to convert the Hex characters loaded into X$. The crafty bit here is the two '%' at the end of each text line, which are put as far apart as you can get them to give the cassette reader something to chew on while the conversion subroutine does its stuff. I run my cassette at 600 baud and ten bytes at a time seems to be about as much as it will take; at 300 baud I imagine you could get away with a few more, but ten is a nice round number anyway!

If machine-code is to be loaded into high memory, say at 8000, then you must include a line of the form shown in printout 2 to set the memory limit so that X$ does not get mixed up with the code you are loading. Incidentally the line numbers followed by CRs do have the effect of making the system perform a CLEAR before each text line is loaded so that X$'s data always resides in the highest memory available.

63998 turns into a string of deletes for the lines from 63900 to 63950.

63999 clears the screen, sets the line width to suit the crummy old TV I use, turns off the LOAD switch and finally kicks off the program.

If anyone has any ideas to improve this technique please communicate!

Not the least of the virtues of this method of loading machine-code is that it is entered in nice intelligible Hex instead of strings of totally incomprehensible decimal DATA statements.

Nicholas Spalding

Printout 1

```
63900  X=PEEK(123)+PEEK(124)*256
63910  X=X–1: y=PEEK(X): IF Y=37 OR Y=33 THEN POKE X,13
63920  IF Y<>33 THEN 63910
63930  FOR I=1 TO 12000: NEXT: POKE 15,72: SAVE: LIST
63940  FOR I=1 TO LEN(X$) STEP 2: Y=0: FOR J=0 TO 1: K=ASC(MID$(X$,I+J,1))–48
63950  Y=Y*16+K+7*(K>9): NEXT: POKE M,Y: M=M+1: RETURN
63960  !X$="A920A2009D00D39D00D2": M=546: GOSUB 63940%
63961  %X$="9D00D19D00D0CAD0F160": M=556: GOSUB 63940%
63998  %63950%63940%63930%63920%63910%63900
63999  %POKE 11,34: POKE 12,2: X=USR(X): POKE 15,22: POKE 515,0: RUN
```

Printout 2

```
63970  %POKE 133, 8000 AND 255: POKE 134, 8000/256
```

A named-file handler in BASIC

Quite a lot of people have asked us for some kind of file-handling system in BASIC. This short routine from *D.I. Swift* goes a long way towards solving that problem:
This program enables you to name all your programs and load them back from tape by name. It will work on all BASIC-in-ROM systems — UK101, Superboard/C1 and C2/C4.

To use it just add two lines to the beginning of your programs. Both are REMs: the first contains the program name, the second is just padding, to give BASIC time to get ready to load the actual program.

For example:

```
  5 REM *NAME#
  6 REM
```

in other words enclose the program name with * and #.

After putting these two lines at the beginning of your program, SAVE it in the usual way — i.e. type SAVE (Return), then LIST (Return).

The program will then be stored on tape as usual. When the computer comes back with its 'OK' prompt, stop the tape, type PRINT "POKE 515,0", restart the tape, and press Return. (This POKE is used to switch off the load flag when loading back the program, to stop any other information on the tape from being loaded).

If you use the same line numbers in your program as those in the 'SEARCH' program, not only will your program load correctly, but while it is loading, the 'SEARCH' routine will be automatically deleted, leaving the entire memory for use. [Another solution would be to place the routine at the 'top' of BASIC memory by using high line numbers — Ed.].

```
 10  REM *SEARCH#
 20  REM
 30  FOR X=1 TO 24: PRINT: NEXT: INPUT"Program name";N$
 40  PRINT"Press play on tape"
 50  FOR T=1 TO 3000: NEXT: PRINT"Searching for ";N$
 60  X$="":F=0:CA=61440: POKE 515,1: REM – CA=64512 on C2/C4
 70  WAIT CA,1: P=PEEK(CA+1)
 80  IF F=0 AND P<>42 THEN 70
 90  IF P=42 THEN F=1: GOTO 70
100  IF P=35 THEN 160
110  IF P=32 THEN 70
120  IF LEN(X$)>7 THEN 60
130  X$=X$+CHR$(P): IF X$<>N$ THEN 70
140  POKE 515,0: PRINT"Loading ";X$
150  GOTO 180
160  POKE 515,0: PRINT"Found '";X$;"'"
170  GOTO 60
180  LOAD
```

A sample RUN:

```
Program name? 3D
Press play on tape
Searching for 3D
Found 'SEARCH'
Found 'HEX'
Loading 3D
OK
```

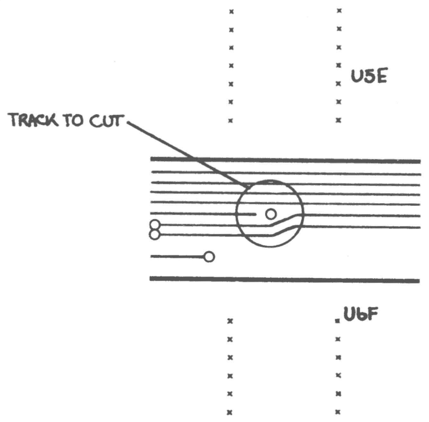
Hardware Mods

A number of suggestions from *John Partridge*, for alterations to the 540 Rev. B video board used on later C2 and C4 systems.

32 characters-per-line mode on start-up
If you would prefer your C2 to start up in 32 mode, cut the track to U4G pin 3, and connect the track to pin 2.

Reverse video for whole screen
1 Locate U2B position.
2 Remove the link from hole 5 to hole 6.
3 Install a 14 pin DIL socket.
4 On U1D location, link holes 1 and 2.
5 Install U2B (74LS86).

Page-swap when in 32 mode
With the capability of CEGMON to write into different 'windows' on the screen, this modification enables the 'hidden' page to be written to, and then instantly displayed.
1 Break the track leading from U5E pin 14 to U5K pin 10. This can be done most conveniently near the plated-through hole between U5E and U6F, on the underside of the board, as shown below.



2 The 'hole' side of the break is connected with thin wire to U2B pin 13.
3 The other side of the break is connected to U2B pin 11.
4 Connect U2B pin 12 to U4G pin 10. (A track from U4G pin 10 leads to a plated-through hole at U4F, and this provides a convenient connection point).

With these modifications carried out, POKEing 56900 will change:
(a) screen size with bit 1
(b) noise enable with bit 2
(c) change page with bit 3
(d) reverse video with bit 4

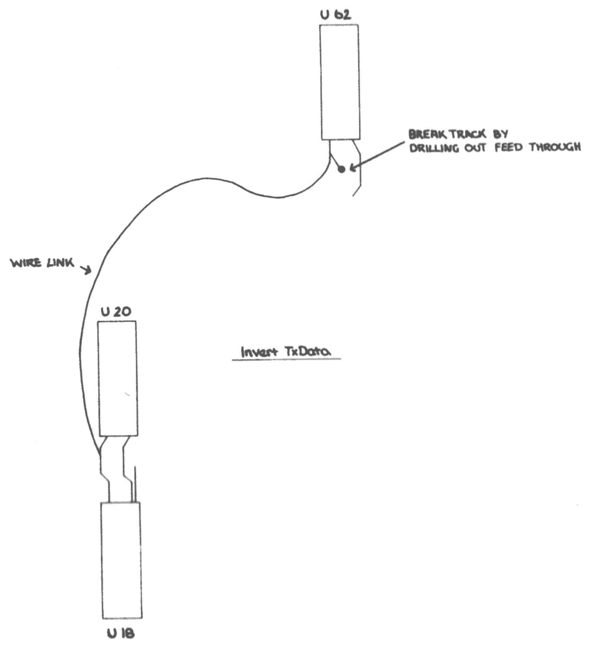
It should be noted that the modifications use colour clear and AC control signals for the reverse video and change page respectively.

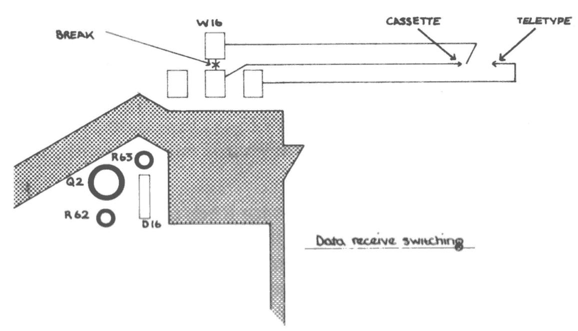
Reverse-video for selectable pixels
This feature is already built into the unused colour circuitry on the 540 board. By installing U1C, U1D, U1E, U1F, U1G, U1H, U2B, U6B, U6E, U6F, any pixel can be 'reversed' by POKEing into the new memory located from $E000 to $E7FF; e.g. POKEing 1 into 59021 ($E68D) will reverse the pixel at 54925 ($D68D) on the screen memory map. The reverses are enabled by POKEing 56900 with any number with bit 4 set.

Programmable alternative graphics set
The new memory added above also allows for Red, Green and Blue bits to be set for each pixel, and these are so far unused [but we have a major feature on colour implementation coming soon — *Ed.*]. Although I have yet to try it, one of these bits could almost certainly be used to select between two character-generator chips. If a 2716 EPROM is used as the second character generator it will be connected in parallel with the existing 2316, apart from pins 18 and 20 on each chip, which can all be connected together, but *not* connected to the existing circuitry, but instead to U1E pin 7. U1E, not U1D is used to get the timing right. Using a similar example to the previous one, POKEing 59021 with any number with bit 2 set will put the alternative character on the screen at pixel 54925.

ASR33 Teletype send and receive on UK101

A note from *Pete Targett*. If you 'scope the the RS232 output on pin 2 of J3 you will find that the line is always low when no information is being transmitted. Teletypes don't like this and start talking to themselves! They require the line to be high (marking) in the wait state. You can remedy this by supplying U62 pin 13 with (TxData) instead of ($\overline{\text{TxData}}$), from U18 pin 10 (Fig.1). Clock the ACIA by one of the many published methods and you have your Teletype receiving.

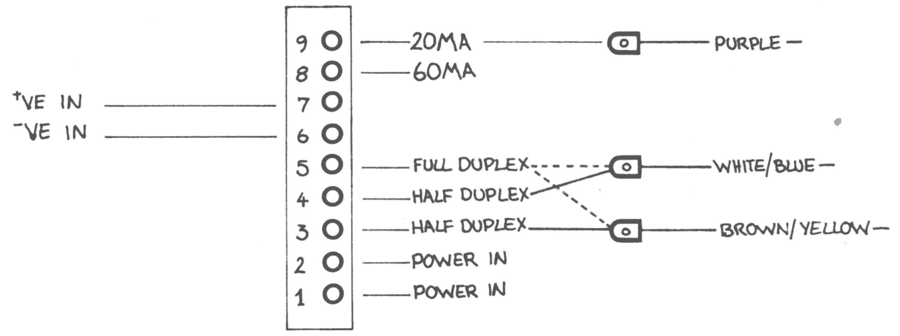




Data receive switching

Now for the extremely easy method of reading from punched tape, or the Teletype's keyboard, into the UK101.

1 Fit Q2 and all associated components (BC 108 is OK).
2 Break the link on pad W18 (just up and to the right of Q2).
3 Fit a 2-pole 2-way switch to the pad W18 (Fig.2). The other side of the switch can be used for clock rate switching.
4 Fit an 8μf capacitor between J3 pin 2 (+ve) and J3 pin 3 (–ve).



Connections on the rear of Teletype (Near the Power In Point) for 20MA half Duplex operation.

5 Connect Teletype up for 20ma half-duplex operation (Fig.3).
6 Connect J3 pin 2 to Teletype +ve and J3 pin 1 to Teletype –ve.

Switch on the system and give it a go. Under 'LOAD' conditions the UK101 will read from punched tape or the Teletype keyboard, and under 'SAVE' conditions will output to the Teletype.

I have this working on two different Teletypes and two UK101s. It works without the use of a –9V supply as Ray Fox suggests. Someone may find technical error in this, but mine has been running fault-free for nine months!

System expansion

Having got your microcomputer to work (not always easy!) and to run games etc. your thoughts may be turning towards other applications. Typically these could include one or more of the following: driving printers, discs, tape drives, etc., running the house, connecting to other computers or as a RTTY terminal if you're into amateur radio. As in the case of my machine (Superboard) the basic design is lacking in I/O and other facilities, system expansion is definitely required. This will involve adding additional hardware to provide more RAM, ROM and various parallel and serial interfaces. There are to my mind four possibilities available at this stage:

(1) Use the OSI range of expansion boards.
(2) Adapt other manufacturers' boards.
(3) Make your own.
(4) All of 1-3.

Of course a whole host of other questions will now arise, mainly to do with the format of the bus, As, if any course other than (1) is selected, the OSI bus is unlikely to be suitable. In my case I'm following route (4), although I've only made my own boards so far. I've decided to use a 19" racking system and my own 43-way bus. The parts for this system are all readily available (Verospeed, RS, Comps or large dealers). As an alternative the Eurocard racking system could be used, some ready made boards exist for this. An advantage of using the larger 19" rack cards is that existing Eurocards can be 'piggy backed' onto them to convert to different bus structures. At this point there will no doubt be anguished cries about the expense of racking systems or motherboards. My answer to this is, that the eventual savings in reliability of the resultant systems will outweigh the costs involved. My system at present comprises a Superboard 2 in a case, with the full 8K of on board RAM. The power supply is a 5 amp affair in a separate box, in my opinion the best place for it! Before embarking on any expansion, make sure the power supply will stand it....

My first board contains full buffering for the address, Ø2 and R/$\bar{W}$ lines plus two PIAs. These are a Motorola 6821 and a Syntertek 6522. The 6522 contains two counter/timers which are very useful for clocks as they will generate interrupts. The above will give a total of 32 I/O lines on one board. The connector used for this board is double-sided so as to provide a fully buffered backplane. A pair of 8T28 bus transceivers will be required for the main board to put in the existing sockets. The address buffers can be almost any non-inverting types, low power high speed types being the most suitable. I used 74LS367's but a quick study of any data book will reveal many other varieties. As regards the rest of the boards, LS TTL should be used where possible, also use IC sockets as fried chips don't function very well.

Returning briefly to the main board (Superboard) you will notice that there is a line labelled D/D (data direction). This is connected to the 8T28 bus drivers and the 40-way expansion socket. This line is very important and must be used correctly. Being tri-state the 8T28's will effectively isolate the main board from the expanded data bus unless enabled by D/D. The actual logic is the inverse of the R/$\bar{W}$ (active low write) signal from the processor R/$\bar{W}$ pin. DON'T invert the R/W signal from the processor and connect it to D/D as suggested in at least one magazine I've read. The external bus read or write must only be enabled when there is valid data to be written onto the bus. Enabling at any other time, especially during a processor read cycle, will dump garbage into the system and stop it! The best way of generating a valid D/D signal is from a combination of Ø2, R/$\bar{W}$ and the relevant chip enable signal. Note that each peripheral must generate its own D/D signal to ensure correct data transfer timing. I won't go into further details as requirements will obviously vary for different systems. However, the 'THINK' slogan could usefully be employed here.

As regards a suitable interconnection method, wirewrap is probably the best, although I used solid wire and patience! At this point I ought to emphasise the importance of checking everything before switching on, as mistakes can be expensive! I hope this article will inspire a few original ideas, don't be put off if you're not an electronics wizard, I'm only a chemist! I've included a few references that I found useful at the end, as most manuals omit such useful information. I'd strongly recommend reference (2) for non-technical people. I learnt a lot from it. It's rather expensive, but your local library might help, if you're nice to them.

References

(1) Texas TTL Data Book.

(2) Microprocessors and Microcomputers, Hardware and Software, Tocci and Lakowski (Prentice Hall), ISBN O-13-581330-1. A very useful book this, as it deals exclusively with 6502 software.

(3) Programming the 6502. Rodnay Zaks (Sybex).

(4) 6502 Assembly Language Programming. Leventhal (Osborne/McGraw-Hill).

(5) Useful journals are: MICRO, Microcomputing, BYTE, Personal Computer World, Practical Computing, Computing.

(6) 6422 Counting and Timing Techniques. M.L. de Jong.

(7) MICRO, October 1979, No. 17, 27-39.