OSI UK User Group Newsletter

Vol.1 No.3 **June 1980**

Documentation □ BASIC program execution □ List sorting
Firmware design □ OEM ideas □ and much other information

Editorial

Back again! with a more general issue this time: a lively miscellany of comments and ideas on documentation and practical solutions and problems, for both BASIC-in-ROM and disc systems. We've no 'main item' this issue, but there should be something for everyone. And if not — it's up to you to provide it! If you have done something or discovered something, and it's not in either the 'manuals' or in this Newsletter, we almost certainly don't know it. So *tell us* — we are learning as much as anyone, we *aren't* omniscient, and we *do* need help and information from everyone in order to help everyone. (And many thanks, of course, to those members who have helped so far!) When all is said and done, this is *your* Newsletter.

A change of format
There is one format change in this issue: we've moved the *User Group Notes* and *Dealer Notes* from the end, to the middle four pages. Their information tends to be impermanent, so we thought you'd prefer to have it as a pull-out section that you can remove if you no longer want it.

Developments
As can be seen in this issue's *Dealer Notes*, the number of dealers and software support services seems to be expanding — a good sign for us all, giving us more choice and, we would hope, more local service. OSI's own range is expanding too, with the new version of the old Challenger 2, the C4, not merely promised but *here*; and with smaller hard-disc systems (the C2-D and C3-D) and a new version of the C1 known to be on the way. There is also a complete new range of experimenter's boards, the CA-20 series, most of which will operate either internally, plugged into the OSI bus, and/or externally, as 'remotes'. The published range includes PIA boards (a new version of the reliable CA-12 board), high-speed analogue-digital interfaces, a multi-function real-time clock, the AC control system, and an 'experimenter's interface board' with LEDs, switches and a large solderless bread-board — a very useful range, and at very good prices.

All in all, a worthwhile range of systems and peripherals. And *if* OSI's New Business Manager, James A. Pike, can be believed, it does look as though OSI really are going to take the UK market seriously at last.

Documentation corner

CLEARer thoughts
The sad saga of the unclear CLEAR continues. *Richard Elen* writes: We've made a few discoveries about the OSI BASIC's CLEAR command since the last issue's note. After you've called CLEAR, you *can* redimension arrays without invoking a DD ERROR. But if you so much as mention an array element after CLEAR but before re-dimensioning, you will get the DD ERROR when you redimension, because calling an array element in an undimensioned array automatically dimensions it at the default value of 10. If you use, CLEAR, check that you don't call any array variable until you have redimensioned.

Dave Caine adds: CLEAR seems not only to reset all variables, strings and arrays — it also destroys any DEFs you may have set up (guess how I know!) and if you use it within a GOSUB, the program won't know where to RETURN to (presumably applies to FOR:NEXT also?). Where on earth could CLEAR be useful? I'm totally baffled and my only suggestion is that it could be used to allow utility routines which may be required to run either individually or chained or after a run of the main program they're working on, but permitting them to use non-unique array or function names.

Any ideas, anyone?

NULL
NULL currently appears to be the main contender for the title of 'most useless OSI BASIC command'. NULL simply POKEs the null-counter (13_{10} in BASIC-in-ROM systems) with the number after the NULL command (i.e. NULL 3). This allows delays for interfacing with relatively slow devices, such as for the delay a bufferless printer needs during its carriage-return cycle. The catch is that greater than 10 after NULL generates an FC (function-call) error — although 255 *is* allowed! — which rather limits things for people designing high-speed cassette interfaces and the like. In practice, ignore NULL; use POKEs instead, they're simpler!

INPUT and null entries
As most readers will have discovered, typing a CR as the sole response to an INPUT statement in OSI's BASIC results in an apparent crash — the ominous letters 'OK' appearing, to imply that all is *not* OK!

Users of disc systems will know that OS-65D and -65U have POKE locations that disable this, allowing a CR response to be treated as a null string or zero value instead. *There is no equivalent 'disable' in BASIC-in-ROM.* However, the INPUT routine does look specifically for the CR (as a null at the beginning of the buffer), and stores the current line-number where CONT can find it, before returning to the warm-start routine. Thus, as UK101 users will know, but OSI users will not, CONT after the apparent crash restarts execution at the *beginning* of (and not after) the relevant INPUT statement, allowing the normal run of the program to be resumed without any real interruption. Normal CONT rules apply: you can print the value of any variable, LIST the program or whatever, but any change to the program itself will result in a CN (continue not allowed) error and true program crash to be issued.

FOR:NEXT loops
OSI don't tell you, but there are shorthand ways of describing the NEXT in nested FOR:NEXT loops that end together. NEXT X: NEXT Y: NEXT Z may be stated more simply as NEXT X, Y, Z; or even as NEXT: NEXT: NEXT (since, for speed, BASIC does not check the variable name, but simply returns to the previous apparent FOR in the nested set) — but, for reasons of legibility and proper documentation, this isn't recommended.

PRINT AT on C2 systems
A limited two-line PRINT AT is available on C2 systems, because the start-location stored in 512_{10} is not zero, as on C1s, but 64, 40_{16}.

On C2s, POKE 512,0 allows PRINTing on the line above the normal line: useful for headings for normal lines of the PRINT; format. A CR (carriage return) forces the value of 512 back to 64. Any value up to 127 is allowed — above that, the system forces a CR on receipt of the first character, so this PRINT AT will not work on lines *below* the normal line.

The same arguments apply to the C1: the line above cannot be reached because the counter cannot take a negative number; and the lines below cannot be reached because a CR is forced. The principle of POKEing 512 could be used within the normal line, but it is simpler to use TAB statements!

> and < as logical operators

The recent issues of *Aardvark's* catalogue have a brief note on this and on the other uses of AND, NOT and OR.

If > or < are used in the form (A > X) the function returns the *logical* value rather than the arithmetic one: -1 if true (A is greater than X), 0 if false (A is equal to or less than X). This would be useful for certain types of game work, where the comparison could force a GOSUB, through a form like ON (A > X) + 2 GOSUB 1000, 2000.

IF and NOT

As a result of the logical comparisons of > and < above, the IF and NOT statements can work without any =*n* sub-statement. For example:

IF A THEN... checks for a 'not-false' condition;
it is thus equivalent to IF A <> 0 THEN...

IF NOT A THEN... checks for a 'not-true' value, i.e. not -1;
it is thus equivalent to IF A <> -1 THEN...

The two functions are normally used for checking flags; but IF A... is also useful for checking if A has any non-zero value.

AND and OR

There is the briefest of mentions in the OSI BASIC 'manual' that AND and OR can be used for 'bitwise' arithmetical operations as well as logical comparisons of the type IF (A AND B) = 0 OR (A AND C) = 1 THEN.... But the two functions arrive at their logical true/false values by bitwise comparisons, which can be used arithmetically instead. If you don't know how these work, look it up in a book on assembly language programming, such as Leventhal's *6502 Assembly Language Programming* (Osborne/McGraw-Hill) — not Zaks' *Programming the 6502* (Sybex), since he assumes that you already know! An AND comparison returns a value only if the relevant bits are set on *both* the comparator and the compared values — thus, as far as BASIC is concerned, 64 AND 32 equals 0, while 24 AND 15 gives 8 (bit 3 is set in each case here, so the 'result' is bit 3 set, i.e. 2^3, or 8). ORing forces bits to be set; thus 32 OR 64 gives 96, but 96 OR 32 leaves 96 unchanged (since bit 5, 2^5, is already set). The function we do not have in this BASIC is EOR, exclusive OR, which clears bits if set, and sets bits if clear — this is available, though, on the 19K Altair BASIC used in OSI's version of CP/M.

David Cannon comments that the Microsoft BASIC appears to be very lax in interpreting logical and relational operators. In principle there is a definite order of priority in which these are done, as given in the manual; but even so, he says, it seems to be essential to use copious brackets to ensure correct operation. For example:

IF THISDAY = WEEKDAY AND TIME = 9 THEN GOTO *sleep?* has to be written as:
IF (THISDAY = WEEKDAY) AND (TIME = 9) THEN *do whatever.*

The important point here is to watch the order of priorities when using OR and AND — in principle at least, they are evaluated *after* = operations, not before. So, again in principle, David's example should evaluate correctly without brackets; but something A = I AND 1 may well result with A equalling I (by evaluating the = and 'losing' the AND by ending the implied LET), rather than returning either 1 or 0 if I was odd or even respectively.

Double-spaced lines

Most people will know this by now, but in case you don't: you can double-space the lines of print output (i.e. insert a blank line between each printed line) by a POKE 15,0 (set the apparent terminal width to zero).

LIST formats

Again, most people should know this, but it does cause some confusion.
LIST 100 lists line 100 only; prints a blank if line 100 doesn't exist.
LIST -100 lists all lines up to and including line 100.
LIST 100-200 lists all lines from line 100 to line 200 inclusive.
LIST 100- lists all lines from line 100 to the end of the program.
LIST lists all lines.

During a SAVE, this allows you to list (and therefore save) only part of a program, such as a set of DATA lines; or else save only the working program if you have any utility programs, for example, using higher line numbers, which you will not want to save along with the working program.

: and , as delimiters in INPUT statements

Jack Pike set us a problem which so far we have not been able to solve — how to trick BASIC into accepting commas and colons in INPUT without losing everything which follows them (followed by that nasty message 'EXTRA IGNORED'). This is easy in disc BASIC, since the delimiters can be changed in RAM; but it's not so easy in ROM.

Because the program is in ROM, we cannot change the delimiters to something that we know is not going to be used in normal text (% or [, for example) — we have to use the program as it stands, which almost certainly means writing a USR routine to do much the same job. Studying the disassembly shows us where the : and , are in the routine: the INPUT routine starts at *A923*, jumps over part of the READ routine, and picks up these two delimiters (along with ") at *A98E* onward. The : is stored in *5B*, the , stored in *5C*. (This all assumes that the INPUT is for a string — this is set by a BIT *$5F*, where the top bit is set high if a string input). The routine then goes through three more subroutines, *B0B4* (which I think assesses the length of the input string), *B3F3* and *A7D5*, before jumping back into a later part of the INPUT routine at *A9B6*. It's going to be a long job to sort this one out, but it's going to be extremely important for the word-processor writers amongst us. Any takers?

Fast screen clear

I was taken to task by *David Caine* for my comment that the Aardvark fast screen clear described last issue scrolled the screen. It appears to, but in fact does not; instead, as Dave pointed out, BASIC is tricked into thinking that the screen is string storage space, and stores strings of spaces there. Thus all that is needed is to ensure that the string-length (of spaces, or whatever) multiplied by the FOR:NEXT count is just a little bigger than the screen memory — 1024 bytes on the C1, 2048 on the C2. 65*32, as given by the routine listed in the last issue, gives 2080; but 59*35, as Dave pointed out, gives an adequate 2065, and gets round the problem of fitting the whole string of spaces and its 'supports' into one program line. C1 users can shorten

the FOR:NEXT loop, of course, to half that needed for the C2; and the routine applies only to memory-mapped systems, not serial terminals as on the larger C2 and C3 systems. For these, however, *Velvet Software* (see Dealer Notes) pointed out that PRINT SPC(0); SPC(0): PRINT scrolls the screen up faster than normal.

RND(X)

Ray Fox writes: Some BASICs use the command RANDOMIZE in conjunction with RND. In OSI's BASIC this is unnecessary as each time a program is RUN, RND(X) with the same X generates the same *sequence* of numbers but starting with a different *number* each time. If however the requirement is for the same sequence of random numbers starting with the *same* number each (RUN) very helpful in debugging programs) then a statement like Z = RND(-Y) must be inserted at the beginning of the program. Z is a dummy variable. For the same Y the numbers will start with the same number each RUN. Changing Y will change the starting point in the sequence.

(Ed.) Someone sent me a superb piece on the machine-code aspect of RND, and explained why it would not completely fill a C2's screen — the repeat cycle for any given number is only 1836 'random' numbers! But I have, of course, lost the notes... so would whoever it was write to me again, please!

Input buffer length

Another garbled piece of OSI documentation confused me and several others about the length of the input buffer. On BASIC-in-ROM systems, POKEing 15 with a value sets the terminal width to that value (the equivalent location is 23 in OS-65D disc BASIC). But *this does not alter the length of the input buffer: the maximum line length as far as a typed line is concerned is always 72 characters.* This is built in to the limitations of all Microsoft's small BASICs (but not its big ones — the C3 CP/M BASIC has a user-defined buffer), since it is an allocated hole, in this case, in the 6502's all-important zero-page. (The Apple Integer BASIC is not by Microsoft, and uses another rather larger area of memory instead).

Ray Fox writes on this point: As detailed in the first Newsletter, location 15_{10} sets the terminal width, for auto CR/LF. Values greater than 72 make no difference to the maximum length of line that can be typed in BASIC; the length of 72 here presumably being limited by the size of the input buffer. However, any value up to 255 inclusive does affect the maximum length of line that can be PRINTed to the screen, and of course to a printer if fitted. Lines longer than 72 characters can of course be PRINTed by PRINT statements separated by ';' or PRINTing the result of a string 'addition'.

An assortment of functions

Amongst a vast quantity of notes and comments from *Matthew Soar* were the following two sets of BASIC functions. (If any other members have sets of useful functions like these, please send them in).

Justification:

TAB(x) statements align numbers on the blank space or sign that precedes the left-hand (highest) digit.

To align on the decimal point, use PRINT TAB(Y+FNP(N)); N — where FNP(N) is:

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

To align numbers on the *right*, use PRINT TAB(Y+FNR(N));N — where FNR(N) is:

```
DEF FNR(X)=1-LEN(STR$(X))
```

'Bit-twiddling'

Nibble functions, where A is the value of the byte, and N the value of the relevant nibble:

```
FNH(N)=INT(A/16) — reads high nibble (upper four bits)
FNL(N)=(A AND 15) — reads low nibble
FNA(N)=16*N+FNL(N) — sets upper nibble to N
FNB(N)=FNH(N)+N — sets lower nibble to N.
```

Nibble functions are useful for packing and unpacking decimal digits, two to a byte.

Bit functions: where N is the bit number of the required bit (0-7):

```
FNQ(N)=INT(A/2^N)-2*INT(A/2^(N+1))
FNR(N)=A-FNQ(N)*2^N
FNS(N)=A+(1-FNQ(N))*2^N
FNT(N)=FNR(N)+FNS(N)-A
```

Respectively, these: read the Nth bit; clears bit N (sets Nth bit to zero); sets the Nth bit to 'one'; and toggles the Nth bit (clears if set, sets if clear).

Machine-code save in BASIC

Several people wrote in to comment on the reason why the *Aardvark* machine-code save (published last issue) doesn't work — it could be said to be a case of 'spot the deliberate mistake'? The error was that regardless of how wide the terminal width is set, the forced CR eventually generated will screw up the save by adding an extra apparent memory byte (as far as the subsequent load is concerned), throwing everything up one byte further than it should be, and leaving a 'gap' repeating the previous memory byte before the CR — guaranteed program crash! The solution, as everyone pointed out, is to reset the other counter — the 'characters since last CR' counter, 14_{10} — between each hex pair, so no extraneous CR is ever forced. Thus: delete the existing line 10 shown in the version printed in the last issue, and insert

```
145 POKE 14,0.
```

C1/Superboard and UK101

We now have a fair number of members with UK101 (because the 'official' user's group for 101s appears to have died); and well over half of our members have Superboards or C1s. But at the moment we (in other words *Tom Graves, Richard Elen* and *George Chkiantz*) have none of these machines — and this is necessarily reflected in the applicability of some of our comments. I take the point made by *G. N. West* that we ought to make every effort to ensure that information is accurate before we print it, since that is the failing of OSI that we are complaining about. But we cannot infer everything from a disassembly listing or a comparison on our C2s or C3 — so would any members with C1/Superboards and/or (especially) 101s please help us in the documentation work where it relates to their machines, by testing out ideas as they arrive? Get in touch with us as soon as practicable if you're interested in this... many thanks!

List sorting

using ROM BASIC's machine-code

I have just stumbled on a novel use of the BASIC ROMs to generate ordered lists and save them on tape in a useful format for input to other programs. I am using it to create source tapes in assembly language for a prototype two-pass Assembler, but it could also be used to supply a list of cheques for a home-made accounts program, or any other application where it is necessary or desirable to edit the data before committing it to tape.

My problem was to be able to store quite a lot of information, in the correct order (i.e. *not* the order I first think of it in!), and be able to edit and delete items at will before finally saving the data on tape, with a suitable delay between each item to allow for processing by the recipient program. After considerable 'boning up' on linked lists, trees and the like, it suddenly occurred to me that what I wanted was exactly the facilities offered by the BASIC ROMs when typing in a program, editing and saving it on tape.

A quick check confirmed that, so long as each statement began with a number and followed this with a non-numeric character, BASIC took no notice of what the line contained and happily stored it away, to be listed, altered, deleted, or whatever. And of course, SAVE followed by LIST would put it all on tape in the right order, and I could use the NULL command to precede each line with a sufficient number of nulls to allow processing of the previous line (NULL 255 or POKE 13, 255 gives almost 20 seconds delay at standard cassette speed!).

By this time thoroughly pleased with myself, I made up a short data tape, loaded my assembler, waited for the input prompt and played back the tape. BASIC isn't tricked that easily though; as soon as the listing began, the Monitor jumped out of my program and loaded the tape as if it were a program, overwriting my assembler in the process! A little experimenting and listening to the tape showed that there are characters preceding the first line which are not printed on the screen, but which obviously tell the monitor to expect a new program. Starting the tape at the beginning of the second line of data was successful and I was rewarded by the sight of a string of machine code instructions as the program decoded the assembly language mnemonics.

To get around this problem I now, having got together my data as dummy BASIC lines, insert an extra line at the top of the list, such as 10 XXXX, type NULL 255 (or whatever), SAVE, LIST, and as soon as the cursor returns after printing the X's, start recording. This results in a usable data tape beginning with a string of nulls then the first line of data.

Using the data is quite easy. Arrange for the program to include, near the start, a few lines such as:

```
110 POKE 515, 255
120 INPUT X$
130 POKE 515, 0
```

If the line number is not required, a simple search along the string via an IF MID$(X$,I,1)=" " will locate the first space after the line number (which BASIC puts in if you forget to do it yourself) and the line number can be deleted by THEN X$ = MID$(X$,I+1), but I keep and use the line number to identify the location of assembly errors.

Make up the data tape, load up your program and type RUN but not >RETURN<. Play back the tape until you hear the beginning of the string of nulls then press >RETURN<, so that the INPUT statement receives nulls as soon as it looks at the cassette port. Failure to do this means a stream of meaningless characters (with a lot of a's) and, unless you're lucky, a crashed program or even system. With these precautions the method has so far been 100% reliable. By the way, the 'OK' at the end of the listing can be used to good effect to terminate your program at the end of the data by adding 140 IF X$="OK" THEN END to the listing given above.

One final tip: before typing NEW to erase the data you have just saved, save it again but without the nulls. That way you can load it back quickly later on if (or in my case, when) you find that it still contains an error. Waiting for all those extra nulls from the data tape is extremely tedious.

To sum up, the method enables the creation of data files of up to 7423 bytes (with an 8K C1) with all the normal facilities of line editing normally enjoyed when entering a BASIC program. Once mastered, the technique is quick and easy to use, and varying the number of nulls means that the data can be saved and loaded at any desired speed.

John Attwood

Designing new firmware for OSI systems

There is a sad little comment in the *First Book of OSI* that 'our thanks and compliments go to the hardware designers at OSI — and to almost no-one else in Ohio'. OSI's design and construction of its hardware are superb — but almost everything else is a shambles. The failings of its documentation we are all only too well aware of! The same goes for its own software — the ideas are there, but even in the 'professional' packages like DMS the all-important attention to detail simply is *not* there. Where OSI have 'bought-in' other people's work, from Microsoft, Digital and others, the results are usually good — but the essential interfacing patches are sometimes left with bizarre loopholes: the outdated Teletype-like 'backspace' in BASIC-in-ROM, for example, or the stupid input/output error in OSI's version of CP/M (which tries to send parallel I/O through a serial port! — see this issue's Disc System Notes). Where OSI have written their own firmware the results are as patchy as ever: it works, after a fashion, but never quite in the way you would expect! A good instance is the keyboard routine for polled-keyboard systems: everything seems normal until you release Shift-lock — and then it's chaos! And who on earth allowed the design of the software and firmware system for the BASIC-in-ROM machines such that it can only load — but not save — its machine-code; whose supposedly 'better' checksum load system turns out (in our experience) to be less reliable than a straight load; and whose assembler will only dump the machine-code that it generates in a checksum format that neither it nor the ROM monitor can read? Are OSI's software designers just utterly stupid, or more than a little insane?

Be that as it may, it was obvious from the start that as soon as we (as organisers of the User Group) had made a reasonable start on patching up the wreckage of OSI's documentation, our next priority must be to replace the support firmware — particularly for the BASIC-in-ROM machines, as disc systems are somewhat freer with all their language software in RAM. We have already gone a long way with this,

and have most of the routines that we would like to use already mapped out and assembled, but not yet linked into a whole. Before we commit ourselves to the relative rigidity of EPROM, it is essential that we ask members' opinions on what *you* feel ought to be in a new standard monitor ROM for the BASIC-in-ROM machines. *David Cannon*, for example suggested that we need: a fast screen clear; a resident editor/resequencer; fool-proof cassette read/write (preferably with named files); cassette machine-code dump, to match existing loader; hex-dec and dec-hex conversions; and longer hex program listings. Our own ideas are described below: what are *your* ideas, *your* requirements?

The main difficulty in designing any monitor is that of packing as much as possible into the limited amount of space available — the space being limited by cost as well as by hardware restrictions. OSI took cost to be their main consideration, and designed their firmware in 256-byte (1-page) modules, to fit combinations for as many different machines as possible onto the one (at that time expensive) ROM chip. This means a vast amount of repetition and redundancy: the C1 also has a complete C2 monitor at the other end of the chip, the older serial chip has pages for hard-disc support even though it's used on a C2 (see *Alan Garrett's* article elsewhere in this issue), and both the reset and machine-code monitor pages have separate screen clear routines (using different methods), neither of which can be called as subroutines. On the assumption that no-one would ever really want to use the full 2K monitor space, OSI's hardware designers even limited the accessible range on the C2 to 3 pages (¾K), and placed the ACIA chip to occupy the next page down, right in the middle of the 2K range of the monitor chip.

The net result is that on C1/Superboard machines the whole 2K is available, whereas only 1¾K is available on C2s. However, the 'extra' page on the C1 is needed for the keyboard-complement subroutines (because the keyboard values are inverted relative to the previous designs) and for the disc bootstrap (because C1/Superboards still use the BASIC-in-ROM under the Pico-DOS system, and thus *will* still need extended monitor facilities in ROM). A sensible use of resources, and a ten-minute hardware mod on the C2, frees 1¾K on each type of machine to do everything else we need. Given that we can remove much of the duplication with judicious use of subroutines, we are still tightly limited by what we *have* to do in that space — and 1¾K is not all that much of a space in 6502 code.

If possible, we also have to maintain compatibility with all the existing software. That's not as easy as it sounds: a brief study of the conflicting requirements of ROM BASIC, monitor, EXMON and assembler will point out some of the difficulties. For a start, only BASIC recognises any kind of back-space, and BASIC's keyboard buffer is limited to 72 characters stored at 0013_{16}-*5A*, while the assembler's buffer starts at 0080_{16} and varies in length because of a packing system used for repeat-characters. And so on...

We could, of course, bypass all this by replacing the lot...! Which would be rather expensive, and almost certainly a breach of copyright unless we can get it covered by the terms of OSI's licence. But there are a number of arguments in favour of this as a long-term exercise: replacing just the first chip, from *A000-A7FF*, would give us access not only to the 'editor' and most of the input/output calls from BASIC, but also all the command and look-up tables — which would allow us to define a new version of BASIC, perhaps even a user-definable one like *Xtal BASIC* for the Nascom and Sharp. This is worth thinking about; but at the moment it's definitely a long-term proposition.

Which brings us back to the monitor, and to the realisation that, for the moment, everything that we want has to be done within the one chip, and within not more than 1¾K. What we *have* to do, if we want to retain compatability with the existing system (but see the *OEM Notes* elsewhere this issue) is the following:

a) provide the Reset entry, leading to Cold and Warm starts to BASIC, to the Monitor, and to the Disc bootstrap on the C1 version (residing in the 'extra' page *FC*);
b) provide the five BASIC support vectors and routines (character-in, character-out, set SAVE, set LOAD, *ctrl*-C check), and the look-up tables for the video routine support;
c) provide a keyboard routine to support the character-in routine;
d) provide some machine-code facilities for the Monitor statement to jump to.

As we see it, *a)* must stay pretty much as it is — it need not necessarily be in the same place, although the former start location of *FF00* must still be able to call it, to maintain compatability with the existing ROM. But everything else could — and, we feel, should — be extended.

Again, there are trade-offs: people who want big extensions to BASIC, like machine-code versions of trace, renumber, line-delete, search and the like, can only have them at the expense of expansions in other directions. Our feelings at the moment are that many of the BASIC utilities — particularly renumber and search — are easy to implement in BASIC, but not at all easy and/or greedy in memory in machine-code; a number of BASIC features would also demand a parser to decode the commands, since the BASIC-in-ROM parser is inaccessible, and that too is greedy on memory. We would thus prefer to limit the BASIC expansions to those which are easily implemented without a parser, and to devote the remainder of the space to things which are either infuriatingly slow in BASIC, or which can't be done at all in BASIC — namely an editor, and a proper keyboard routine and extensions to the machine-code monitor.

One type of editor which already works on OSI kit is the *Sirius Cybernetics* type with twin cursors — this has some limitations, but it does run all the time that keyboard input is required, without needing to be called. We have, of course, re-written the keyboard routine — it now *does* do what you expect! — and while, at the moment, it is slightly larger than the existing routine, a little algorithm-juggling should trim it down to the requisite size, as well as freeing its delay-routine as a subroutine for use in other programs. Other routines — like screen-clear! — have also been re-written as subroutines for use elsewhere. On the machine-code side, my own experience of a Nascom 1 (which had an un-endearing habit of scrambling memory on reset, but which had superb firmware in the T2 and T4 monitors) has given me a pretty clear idea of what a proper machine-code monitor should have. I've thus written a proper modify routine that allows input of both hex and ASCII; a tabular memory display dump; hex arith routines, including relative jump; breakpoint handler; intelligent block move; and some other routines, including (of course) save and load. *Richard Elen* and *George Chkiantz* have written some other extensions, like a tape-header routine for named tape files. On the BASIC-support side, we are undoubtedly going to vector all the support routines through RAM — which the C1/Superboard does at the moment, but which the C2 does not. This is to allow user-defined I/O — which at the moment, as I know to my cost, is almost impossible on the C2 series without re-writing almost the whole of BASIC! I have also written an easily implemented BASIC-trace (switched on by a single POKE) and am currently struggling with the one other BASIC extension I feel is essential but

impossible to implement in BASIC, namely block-delete of program lines. Other items under consideration include ways of leaving open 'holes' for expanding the monitor, to append proper 'extended monitor' facilities like a disassembler and a disassembling machine-code trace; and also various experiments with a real-time clock; and...

But anyway, what we need to know, if this is to be the Group's monitor rather than our own indulgence, is a clear idea of what members want of a ROM support monitor. What we need to know is what *you* want, what *you* are working on or interested in along these lines. Would you let us have your ideas or comments as soon as possible — we'd like to have something to soothe some of our members' firmware headaches by the time the next issue comes round!

Tom Graves

The range of OSI monitors

C2 and C3 range

For those who possess a C2 or C3 machine you will most often find a *SYNMON V1.0* monitor on your CPU board (I've never seen any other version, or anything else!). However, OSI claim to produce two versions of this monitor ROM, one for video systems (C2) and one for C3 and other serial systems. Now this may well be so, but there are no external markings on any of our SYNMON V1.0's to show this! Anyway, to continue:

Page	Contents	Run-time address
1	Monitor: 540 video board, ASCII keyboard	FExx
2	Reset support ('C/W/M'): 540 board and ASCII keyboard	FFxx
3	Polled keyboard routine (542 keyboard)	FDxx
4	Monitor: 540 board with 542 polled keyboard	FExx
5	Reset support ('C/W/M'): 540 board and 542 keyboard	FFxx
6	CD74 hard-disc bootstrap	FDxx
7	Monitor: serial systems	FExx
8	Reset support ('H/D/M'): disc systems	FFxx

If you look at a 502 CPU board (C2 series) you will see a patch socket with 3 links, from pins 1, 2, 3 to pins 12, 11, 10. These call up pages 3, 4 and 5 respectively. So far fairly straightforward. The monitor ROM is actually addressed by separately decoding the high address lines of addresses *FDxx, FExx, FFxx*, and then converting them back to a binary address. (This is why you have a 2K monitor but can only 'see' 3 pages of it).

On a 505 (not Rev. B) board out of a C2-4P MF, you will find a similar patch socket, with links from pins 1, 2, 3 to pins 12, 11, 7 (note the difference). Hence a C2-4P MF will display 'H/D/M' on power, reset or break, but has the same keyboard and monitor routines.

On a 505 (not Rev. B) board out of a C2-OEM machine you will find the same patch socket but with only *two* links, from pins 2, 3 to pins 7 and 8. This brings out an 'H/D/M' message and uses a serial monitor.

At this point a very interesting point comes to light, namely that the C2-4P MF's 505 board does not produce the 'H/D/M' message in the same way as on the C2-OEM's 505 board, which drives a serial video system rather than a built-in 540 video board. This is fairly obvious and directly leads to the conclusion that there are two different reset support pages producing 'H/D/M' messages; and because the boards in question have the same page (page 8) for the reset support code linked in, then there must be two different ROMs: one basically intended for video systems in which pages 1, 2, 6, 7 are not usually used; and one for serial systems which apparently usually use only pages 6, 7, 8 — and 6 (the hard-disc bootstrap) only on C3s. On a 510 board from a C3 machine a similar patch socket exists with pages 6, 7, 8 of the monitor ROM selected: this displays the 'H/D/M' message on a serial video system.

In answer to the question 'What is in my monitor ROM?', we have come a fair way towards an answer. The definitive answer will only come when someone with a little time to spare manages to do a disassembly of the whole of the 2K monitor ROM, for both a 540 board video system and a serial-based system. Which leads to a further question of 'How can this be done?' On a ROM BASIC video system (i.e. a C2-4P) a machine-code disassembler might be used, via the monitor, to disassemble other monitor ROMs, perhaps placed in one of the BASIC-ROM sockets — a suggestion I leave for those who wish to try this. (I can provide a serial monitor ROM and a monitor ROM from a video system for a day or two if anyone wants to try — contact me at Mutek).

One final point: the two monitor ROMs, both called SYNMON V1.0 — one from a video system, the other from a serial system — must, by my argument, be different inside; yet outside they are the same device, with the same name and pack. As ever with OSI, confusing, isn't it!

Alan Garrett

C1/Superboard

(Added afterwards by your editor!) The monitor ROM used on the C1/Superboard series does actually have a different name (!) and contains what appears to be both a C1 and C2 monitor — which may resolve Alan's problem. The C1's monitor set-up, as run, contains a disc-bootstrap and keyboard-complement routines at *FCxx*, and then much the same as the C2's monitor: keyboard routine at *FDxx*, machine-code 'monitor' at *FExx*, and reset support at *FFxx*. The catch is that it is *not* the same as the C2's monitor: there are some important differences. On reset, the C1 version loads in a table of vectors from *FEF0* to *FEF9*, for the BASIC support routines, and stores them at 0218_{16} upwards; these vectors are later called via 'JMP-indirect' (6C) opcodes when BASIC requires the monitor's support. The C2 version uses 'JMP-direct' (4C) opcodes for these vectors, making user-defined input-output almost impossible, and uses the C1 version's vector-table space for a vague attempt at a 'GETKEY' routine. (Both versions, incidentally, have redundant Reset, IRQ and NMI vectors at the top of the *FExx* page, apparently in case the page is switched for use as page *FFxx* — though the monitor would not work at all under those circumstances!). Other major differences occur in the keyboard-polling routine in page *FDxx*: the C2 version does a straight STA/LDX (or /LDX) sequence to look at the keyboard switches, whilst the C1 version has to call a set of complementing routines, stored in the 'disc bootstrap' page at the upper end of *FCxx*, because the keyboard switches invert the key values as defined by the earlier 540-board system. Serial I/O through the ACIA also has to be dealt with by separate routines in the 'disc-bootstrap' page rather than within BASIC, because BASIC expects either a UART at *FBxx* or an ACIA at *FCxx*, not an ACIA at *F0xx*! The bootstrap routines occupy the same location as the ACIA that BASIC expects, of course... which is why everything had to be re-written

to by-pass BASIC. But exactly why the C1's monitor should also have a C2 version within it, when the C2 is already served by the other monitor, is something of a minor mystery — perhaps they just needed to fill the space with something. There are also apparently two different C1 monitors, with the same contents, but different chip-types — the board has links to support two different types of ROM.

All in all, the situation with the monitors is, as usual, a tangled mess. As explained elsewhere, we are currently trying to re-write the monitors for the smaller systems — any comments or ideas to help us in that direction would be most welcome!

Letter from America

User Group organiser *Richard Elen* was over in California on business recently, and was able to grab a little time off to talk to people about the OSI scene over there. This is his report...

The end of May saw the 1980 National Computer Conference and exhibition at the Anaheim Convention Center. The show was exceptionally large, with over nine halls in the Center itself and two floors — housing the Personal Computer Festival — in the nearby Disneyland Hotel.

While at the Show, I visited the OSI stand, tucked away in a corner of one of the main halls. On show was, as one might expect, a *C1*, in its new-style case; a *C4MF*, with an excellent set of business programs running on it; and a *C2-8P* with a whole battery of add-ons, including a music program running via the D/A, a home security system, light and appliance controllers, the works. A Votrax voice synth card told you what was going on and the system was ably demonstrated by one of OSI's engineers. A pair of *C3s* were also on hand, running a number of powerful business systems.

I had a useful discussion with *James Pike*, OSI's New Business co-ordinator, about the company's attitude to the UK market, and I'm pleased to say they seem to be most interested in looking after us lot over here. Lines of communication between the UK User Group and OSI are now well and truly open, and we can hope to be kept informed with useful data.

I also visited a couple of Californian dealers/distributors, and got a chance to see some of the new developments, notably the new *540 rev.B1* colour video card, now being fitted to the *C4*. You can retrofit one to a *C2*, of course, and at least two people in the UK have recently completed PAL colour conversions so we'll soon be able to have access to all 16 colours, alphanumerics and graphics, foreground and background colour. The *540* colour card allows each screen position to be set to a foreground and background colour: a simple command changes the entire screen background to a chosen colour. For more details on this and other aspects of the *C4* you should try to get a copy of the current issue of *onComputing*, McGraw-Hill's new 'beginners' magazine. A multi-page colour review therein gives a good appraisal of the new system. Shortly we hope to present a review of the machine for members in the Newsletter; as many of the facilities utilise modifications to existing boards, *C2* and *C3* owners should also benefit, as new boards can be obtained with the new facilities, hopefully with some kind of trade-in deal. New lines of communication were also opened up with regard to getting better facilities from

continues after Group Notes and Dealer Notes

User Group Notes

Join the club?

We know that several members would like to see the Group emphasising the 'club' aspect more — arranging local meetings and the like. The main problem, as far as we are concerned, is that we have more than enough of a job already! Clubs are certainly a good idea, but we realise that, because we're operating on a somewhat remote national scale, we are not really up to organising things on the local scale that clubs really need.

So we have had a couple of members asking us if we would supply them with a list of other members in their locality. But this immediately raises an ethical problem: that we should not hand out members' names and addresses without their permission beforehand — to do so would be a breach of privacy. So we will *not* be issuing your address to anyone who asks for it! A much better solution, that should be acceptable all round, would be if we publish the names and the like of any members who would like to organise club activities in their local area, leaving other members to contact them direct. So if anyone would like to organise something in their locality, let us know fairly soon, so that we can assemble a 'Contacts' list for the next issue.

Disassembly and development

Serious programming work — in which I include serious games programming! — needs practical tools, and we do have access to most of the tools and information here if members need it. But one of the main tools, the disassembler, presents with another ethical problem: that of copyright. Computer copyright law is a shambles at the moment, but even so, it is clear that both Microsoft and OSI would get more than a little upset if we published the source-code of their major works. And a disassembly of ROM-BASIC, for example, occupies the best part of 5000 lines of code, or well over 100 pages of printer listing — not a minor item. *For reference purposes only* we do hold disassemblies of ROM-BASIC, EXMON, assembler (cassette and disc versions) and various ROM monitors, and we intend to extend this to cover the other major software like disc BASIC and OS-65D and -65U. We can handle specific enquiries, but be warned — we're only doing this in our minimal amount of 'spare' time, so don't expect a speedy response!

Planning cards

In last issue's Notes I mentioned the idea of producing write-on/wipe-off planning cards, to cover a variety of needs. Several people wrote in to say they were interested, so these are now being prepared, to be printed just after this issue goes to press. You'll find a list/order form as a loose insert in this issue; the price of 50p each, to members only, is as low as we dare make it — you'll also find them in the computer shops later on, and at a rather higher price! The designs we've done so far would seem to cover most general needs — video planning, opcode lists, hex-dec conversions, program planning — but no doubt there will be others. If you can think of other card designs that would help, let us know. And we'll also be issuing some of the designs as pads, for more permanent records.

Small ads?

At the moment we have no advertising, but since these Notes and the *Dealer Notes* have moved to this pull-out-able section, advertising becomes practicable. You'll see a comment in the *Dealer Notes* about the general advertising cost — but would

members like a 'small ads' section, to advertise their odds and sods? A realistic price, given our production costs, would be about 2p per word, with a minimum charge of £1.00. This, with the dealers' advertising, would help pay for a bigger issue — your comments, please?

. . .and a small ad!
User group organiser *Richard Elen* wants to sell his *Elekterminal*-based serial terminal. Complete with George Risk ASCII keyboard and Thompson-CSF video driver to drive either a video monitor or TV, it's currently used to drive Richard's C2 system; but he's replaced the terminal with the 540/542 standard C2 video-and-keyboard combination. Richard would like around £100 for it; more details from him at the Group's London address, *12 Bennerley Road, London SW11 6DS.*

Dealer Notes

We're glad to note an increase in the number of dealers for both OSI hardware and OSI-compatible software. We've listed below all those new dealers (new to us, at least) whose addresses we've found in the magazines and the like; where we've been able to get in touch with them, we've also included some details of what they are doing. As before, they are in no particular order!

Mighty Micro, 33 Cardiff Road, Watford, Herts. Tel: (0923) 38923.
Mail order section: P.O. Box 17, Basingstoke, Hants. Tel: (0256) 56417.

A consortium formed by *Watford Electronics* and *Videotime Products* (and possibly *Lotus Sound?*) to handle the computing side of their business. Deal mainly in Superboards, but also sell other OSI kit such as the C4 (currently in stock!). Also peripherals like the SuperPrint 800 (alias Base-2) printer and Softy firmware development kit. Probably the largest volume seller of OSI — low prices offered as a result!

Simple Software Ltd, 15 Havelock Road, Brighton, Sussex BN1 6GL.
Brighton (0273) 504879.

Best known for their *Microcase* ABS cases for Superboard, Compukit and others; but actually in business to sell software! Limited range of (low-priced) software is currently being expanded. (They sent us a sample tape, which we'll review next issue).

Velvet Software, 26 Colesbourne Close, Worcester WR3 9XF. Tel: 056 885 453.

Range of tape software for Superboard/C1 and UK101 (*Startrek* to be reviewed next issue). Also set of peripheral interface kits for those systems, for reed relays, parallel ports and programmable sound generator — kit with all facilities is under £50. (Would any member using this unit care to send us a review? — it sounds interesting and well priced).

J.M. Electronics, P.O. Box 71, Norwich NR6 7JE. Tel: (0603) 412222.

Tape software, particularly for UK101, mostly games and priced from around £3 to £7. Software publisher — keen to publish any good software for UK101 and C1/Superboard (also Apple). Also disc system under development for UK101 — should be ready by the time this issue goes out. S.A.E. appreciated with enquiries!

Philbrand Associates, Great Oak House, 2 Albany Close, Esher, Surrey KT10 9JR.
Tel: (0372) 62072.

Large systems for business use — C8-P upwards. Specialises in word-processing systems based on C3 and *Wordstar* software.

Millbank Computers, East Lane, Kingston upon Thames, Surrey.
Tel: 01-549 7262.

Specialist in business systems, using C8-P upwards. (Parent of *Philbrand Associates*?)

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

Advertise full 'personal' OSI range — Superboard to C4; also games software for OSI range, and for UK101 and TRS-80. (No further details as yet).

Easicomp, 57 Parana Court, Sprowston, Norwich.
Tel: (0603) 407923, also (0508) 46484.

Superboard and own-cased Superboard in particular; also software for Superboard, and Pet, Nascom, Research Machines. Commissions and publishes software. (No further details as yet).

New systems
It's nice to see that we haven't had a repeat of the Superboard saga with the new C4 — it arrived less than a couple of months after it was first advertised, and several buyers already have them in this country. There still seems to be a little trouble with the PAL colour conversions (dealers would *definitely* like members with colour implementations for any OSI kit to contact them!), but apart from that the new machine is very neat indeed, and well worth the small extra above the old C2 price. I haven't been able to play with one in detail; but the amount of built-in I/O ports and the like will make all sorts of specialist interfacing very easy, and the disc system works out at the same price as an Apple *without* a disc. All right, the Apple has better colour handling and finer graphics (although a dot-addressable graphics card for the OSI bus is on the way, I'm told) — but have you *seen* the price Apple charge for a single RS-232 port?

There is another version of the Superboard (Superboard III?) now on release in the States, but we've no details as yet. And also on the way, in the bigger league, are two new hard-disc systems, the C2-D and C3-D. These are essentially the same as the existing C2-OEM and C3-OEM (i.e. 48K RAM, twin 8" discs, and 6502 or triple processor board respectively), but replacing one of the floppy drives with a built-in 10 Megabyte hard-disc (Shugart, I think). Pricing is interesting: it's likely to be around a thousand quid cheaper than its nearest equivalent, Cromemco's Z2-H.

Wheeling and dealing?
Various things have been going on on the dealer front, in relation to the UK dealers' somewhat tortuous relationship with OSI, and between the UK dealers themselves.

The relationship between OSI and the UK dealers is 'estranged' — as far as I can work it out, no-one — strictly speaking — is an 'official dealer'. OSI's official European distributors are ADHOC; but their original terms for UK dealers (trade terms in the UK, before shipment or Customs duties, were slightly higher than

American domestic *retail* prices), none of the original UK dealers are willing to buy from them, although some of the newer dealers are apparently doing so. (If you thought their prices were high, look at what ADHOC are getting away with in France and Germany: French prices are nearly twice ours, and German ones are almost three times — so it's no wonder that one dealer said that a French customer flew over especially just to buy a Superboard!) ADHOC are still trying to pump life into a promised maintenance service company; but they've withdrawn their European manager from London when last we heard, so it's unlikely that much will come of it. In case you're wondering where the UK dealers get their systems from, it seems that they have two options: some, like *Mighty Micro* and *Millbank*, have managed to bypass ADHOC's 'official distributorship', and seem to be buying direct from OSI; while others, like *CTS* and *Mutek*, buy from the American wholesale market, and apparently at much the same price as buying direct from OSI, with immediate availability too.

The other matter in hand is a certain amount of wheeling and dealing between the UK dealers — with the intent of helping everyone, dealers and customers. A kind of 'dealers' convention' held in May at the instigation of Bill Unsworth of *U-Microcomputers* set up an agreement whereby dealers are sharing technical notes and, to a certain extent, software; they're also covering each other for emergencies when one dealer urgently needs a board, for example. I gather there was also a certain amount of 'price-fixing', but on the whole these have been rounded down rather than up! But with the removal of any real threat of a price-war between dealers, they can settle down with more certainty to develop specialities of their own, and, we would hope, to offer a better and more local service network. It'll be interesting to see how this all develops.

The User Group has an important part to play in all this, incidentally. The dealers recognise that OSI's documentation is so poor that a complete rewrite is necessary, and have asked us to co-ordinate the collection of any notes with a view to producing proper documentation. At the moment they're particularly keen on producing notes for DMS and OS-65U on the larger systems — so we'd appreciate any notes and comments on these from any members using them.

Advertising

At present there is no advertising in the Newsletter — and I have no doubt that many members would prefer to keep it that way. But the increasing size of the Newsletter — which I presume, again, that most members would prefer — is sending our production cost for each issue much higher, and only just within the Group's current budget. We have moved the Group Notes and Dealer Notes to the middle of the Newsletter, as a pull-out section: and if we are to have any advertising, that is where it should be, since it can then be thrown away when it is no longer relevant. And we now have enough of a readership to make specialist advertising worth while. So I would like to know from members whether they feel that advertising, in a pull-out section, would be acceptable (to prevent us having to push up our membership fees!); and whether members and/or dealers would be interested in advertising their wares in the Newsletter. Assuming that artwork was supplied, a rate of £40 per page (and pro rata for half and quarter pages) would be realistic for us, and we hope for advertisers as well. (You could see it as subsidising our good works, perhaps...). Your comments, please — and any takers?

OSI for UK dealers, and with the major UK dealers now working together more we should soon see some dramatic developments in terms of availability and backup. Our dealers in the UK have really pulled their fingers out and are doing a great job of handling the equipment: soon it should be even better.

I also picked up copies of some new publications related to OSI gear. *TIS*, well known for their PET literature, have produced a book called *The C1 Workbook*. This is a great guide for beginners and explains not only BASIC, but all the vagaries of the *C1/Superboard*. New from a Californian publisher called *Elcomp* is *The First Book of OSI, Vol.1*. This book, culled primarily from OSI's technical news-sheets, enquiries and information from *Aardvark Technical Services* (who published what must be an earlier version of this — *Ed.*), lays out a lot of useful information on the machines. The first volume covers mainly *C1* and *C2* information, including discs. Although there is a lot of information there, and it is well indexed, there is a certain unpredictable quality about the articles: they are presented in no real order, but there are some real gems among them. One example is a simple BASIC program which POKEs machine code into the top of memory to clear up the bug in OSI's BASIC-in-ROM string-space housecleaning routine, making string usage a little less annoying. I've tested it, and it works; we'll print it for you shortly. Also given is the hardware mod to give the *C1/Superboard* a 54 character by 32 line display. Some UK dealers are already offering this mod; if you fancy doing it yourself, the information is in the Elcomp book. Two volumes are to follow: *Volume 2* details the 'higher-level' DOS systems, *OS-65U* et al., and the *WP-2* word processor; and *Volume 3* will cover interfacing and how to hang your own things on the end of OSI machines and boards.

Another useful arrival is the publication of three Howard Sams service manuals: one for the C1 (which also serves the *Superboard* and includes discs); one for the *C4* (which will also cover the *C2*, and, once again, the minifloppy systems); and one for the *C3*. The latter book also includes data on the *470* Disc-Controller board, so if any of you have been thinking of upgrading a *C2* with the *470* board, to retain the BASIC-in-ROM with disc expansion (it's faster than OSI's disc BASIC), you'll need it. The three manuals are all reasonably priced and give clear, colour-coded diagrams and photos of the board layouts, plus full servicing and testing details. A must if you're into the hardware side of your machine.

All these publications are currently being ordered from the US and will hopefully be made available to members in the near future.

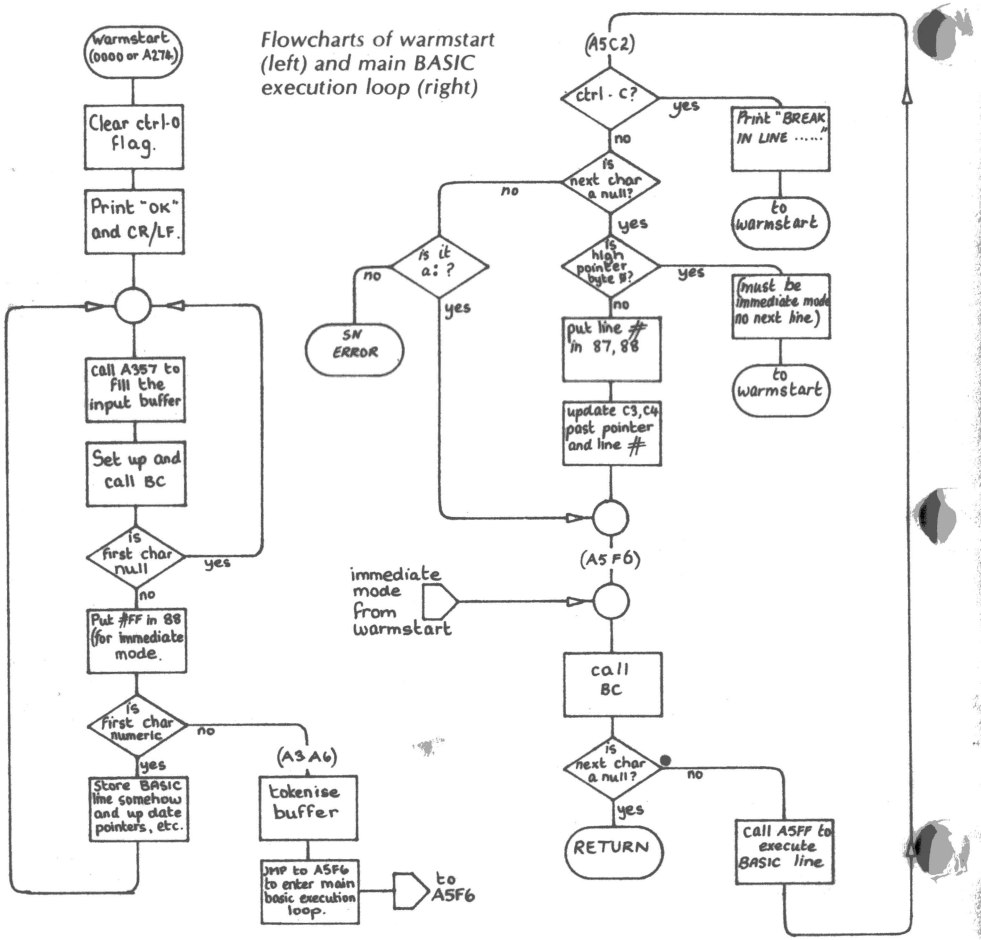
(*Ed.* — I hate to sound cynical, but we seem to have heard a lot of those promises from OSI before: better service, better deal for the UK market, freer supply of information, and so on. OSI aren't exactly an open company — most members probably don't know that it is OSI's *company policy* to refuse to answer phone calls from anyone other than a very select group of dealers. So I'll only believe those promises when we start to see hard results... and if anything really *does* happen in the next three months, we'll let you know in the next issue!)

BASIC program interpretation

Courtesy of Aardvark Technical Services

(The description that follows is derived from Aardvark's BASIC Notes; it applies in particular to OSI's ROM BASIC Version 1.0 Revision 3.2 — the 'BASIC-in-ROM — but also in general to Microsoft's 6502-series BASICs. Addresses and values given are those for the BASIC-in-ROM, and are in hexadecimal unless otherwise stated).

A good place to start exploring is the warmstart entry at *A274*. BASIC can also be warmstarted by a jump to *0000* — where the system puts *4C/74/A2* during the coldstart sequence. In warmstart, BASIC is looking at the keyboard, waiting for immediate-mode commands or BASIC instructions with line numbers to be entered.



Flowcharts of warmstart (left) and main BASIC execution loop (right)

See the warmstart flowchart. BASIC first clears the *ctrl-O* flag (LSR $64 clears the flag — the top bit of *64*) to allow printing; invokes the message printer at *A8C3* via an indirect call through *0003* to print the 'OK' message — the A and Y registers hold the low and high bytes of the message's address, and the message ends with a null. (The 'OK' itself is stored at *A192, 3*). Now the 'fill the input buffer' routine is called. This routine (at *A357*) inputs from either keyboard or ACIA, via the vector at *FFEB*, and depending on whether the top bit of the LOAD flag (*0203*) is clear or set respectively. The 'fill buffer routine' keeps a count of the characters in the X register, stores the characters in the input buffer (*13-5A* in the zero-page — hence the limit of 72_{10} characters on input), handles the very limited 'editing' functions of false-backspace, and line delete (with **a**); checks for the 'no-print' code *ctrl-O*; and ends the input on a CR, jumping to *A866*. This places a null at the end of the buffer instead of a CR, and then runs on into the CR/LF routine at *A86C*, which also 'hangs around' for any extra nulls that the system finds in *0D*. (Nulls are put in the output stream after CR/LF, if needed for a slow device, by placing the number of nulls in *0D*. This normally defaults to 10_{10}, and is changed by a POKE 13, *n* or a NULL *n* command (the latter vying with CLEAR for OSI's most idiotic BASIC command, since it doesn't do what is expected of it at all — see Documentation Corner in this issue!).

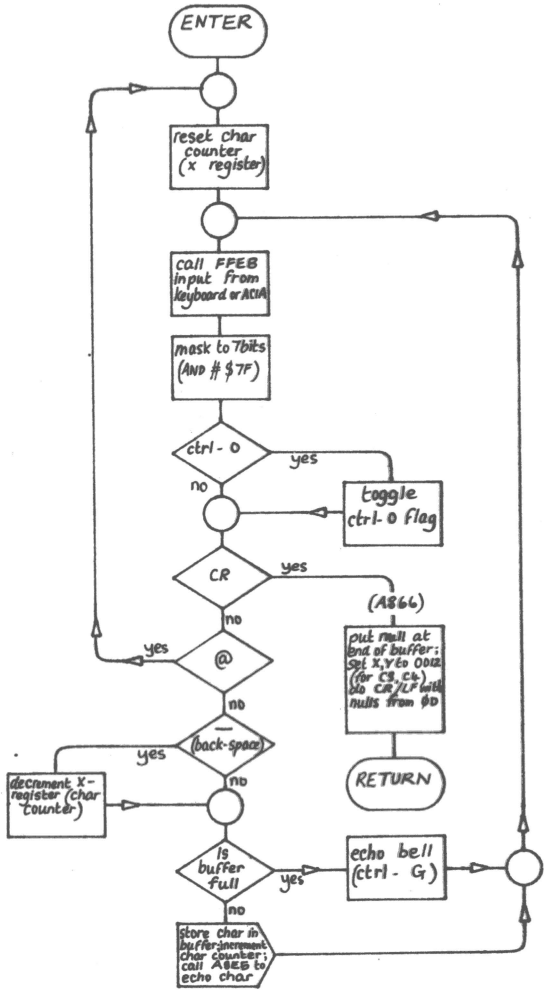
Study the flowchart for the *A357* routine for how this works. The same routine is also called by the INPUT routine.

There exists a vital routine callable at *00BC* (the code for which is copied during coldstart from *BCEE-BD05* in ROM) that puts the *next* character in the current BASIC line being worked on into the accumulator. The routine is sometimes referred to as CHRGET; the *current* character may be pulled into the accumulator by calling *00C2* rather than *00BC*, as CHRGOT. The *BC* routine jumps overand ignores any spaces, and also sets the carry flag if the character being passed is *not* numeric, for the information of the routine that called this subroutine. The address of the current character is in *C3, C4* — the address portion of an LDA instruction. All the main routines use the *BC* subroutine to find out what's next; *(C3)* is constantly being changed by the users of the subroutine, in addition to being incremented by the *BC* version of the subroutine each time it is called.

During the warmstart sequence, the *BC* routine is used to work through the ASCII in the input buffer as it is tokenised. *(C3)* is set by the call to *A866* to point to the input buffer. If the first character is numeric, the buffer must contain a numbered line of BASIC source code, so the routine jumps to *A295* to do the 'tokenise and store in BASIC workspace, updating necessary pointers' job on the input buffer. If the first character is not numeric, the statement is assumed to be immediate-mode, *A3A6* is called to tokenise the buffer, leaving the tokenised line in the buffer; the routine then jumps out of the warm-start loop to *A5F6*, the main entry to the 'execute BASIC statements' loop.

At *A5F6*, the 'execute BASIC statements' loop calls the *BC* subroutine to check if the next character in the current line is a null (i.e. in immediate mode, just a CR in the buffer). If it's not a null, it must be a BASIC command, so the loop then jumps back to its beginning at *A5C2*. When a program is RUN (from the beginning), the command RUN is executed as a BASIC statement, calling the RUN routine at *A477*. This *a)* points *(C3)* to the contents of the *(79)* pair, pointing to the beginning of BASIC workspace (set to *0301* by the cold-start routine); *b)* resets the string pointer pair *(81)* to the top of memory as recorded in the *(85)* pair; *c)* resets the array pointer to the end of BASIC program space as pointed to by *(7B)* — so all array, variable and

The 'input and fill buffer' routine (A357)



string pointers are reset to their start positions; *d)* the stack pointer is reset to *FC*, which on return from that subroutine means that it is pointing at *01FE*; *e)* a *00* is stored in *008C* and *0061* (we're not yet sure why); and *f)* a **$68** is stored in *0065* (again, we're not sure why, but it probably has something to do with **$68** being the opcode for PLA, pull the accumulator value from the stack). On returning from the RUN routine, the program jumps to *A5C2*, the start of the 'do the next BASIC line or statement' loop. See the 'main BASIC execution loop' flowchart.

Starting at *A5C2*, the program first does a *ctrl*-C check, calling the subroutine at *A629*; if *ctrl*-C is found, execution stops, printing 'BREAK IN LINE *(contents of the (87) pair)*' before returning to the warm-start loop. If *ctrl*-C is not pressed, the program checks to see if the next character of the line is a null (the beginning of a new BASIC line). If it isn't, and if it isn't a ':' to indicate an additional statement in this line, the program jumps to the syntax-error printer, and returns to the warm-start loop. If the next character is a null, the high byte of the pointer that follows it will have a null also if the line just executed is the last in the program. If this is found, the program returns to warmstart. Otherwise, there must be another line of BASIC: so the program picks up the line number and stores this in the *(87)* pair, and then increments the *(C3)* pointer past the 'next-line-is-at' and line-number pairs to point at the first program character of the BASIC line. (If the last character looked at was a ':' rather than a null, the program would have jumped to this point). The next sequential instruction in ROM is at *A5FC*, which is where we came in with the immediate-mode statement RUN.

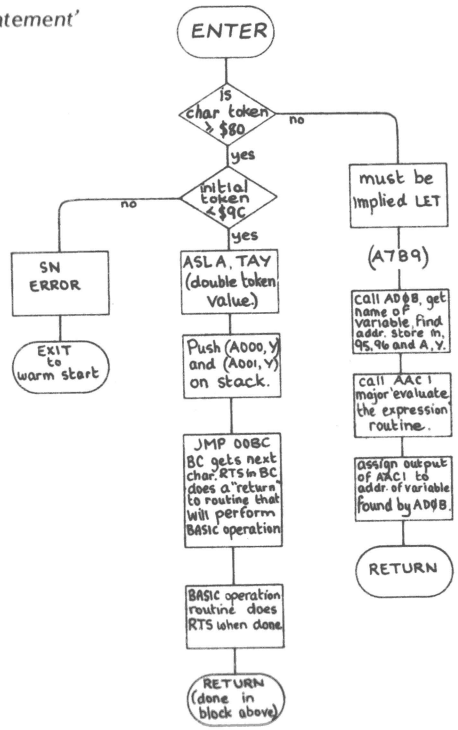
As mentioned earlier, *A5FC* calls the *BC* subroutine to check for a null; if not a null, it calls the subroutine at *A5FF* to do the actual work of executing the BASIC statement that it finds in the line, before returning to the start of the loop at *A5C2*.

A5FF calls the *BC* subroutine and checks to see if the first character is greater than 80_{16}. If not, it is not a normal 'token' for a BASIC command, and it is thus assumed to be an implied LET command. In this case, the token for LET is inserted, to call the LET subroutine at *A7B9*. This then calls *AD0B*, a very important subroutine that finds the name of the variable to be assigned by the LET, finds its address in the variables storage space, puts that address in the *(95)* pair, and also leaves the address in the A and Y registers. In this case the LET routine at *A7B9* then stores the variable's address in the *(97)* pair, and checks for an '=' (using the *BC* routine, of course, to look for the next character); if it doesn't find one, the program jumps out to the syntax-error routine. If the '=' is found, the important subroutine *AAC1* — the 'evaluate an expression' routine — is called, which leaves the value from the expression in the four-byte floating-point accumulator *00AC-AF*. The LET routine returns by way of a JMP to *B774*, the 'store the floating-point accumulator at the address pointed to by *(97)*' routine. Since the LET routine was itself called as a subroutine, the sequence ends with a return to the top of the 'execute BASIC statement' loop at *A5FC*, which restarts the sequence again at the beginning (at *A5C2*) with the *ctrl*-C check.

If *A5FF* does find that the first character is greater than 80_{16}, the character is a token for a BASIC command — presumably other than LET! The program then checks that the token is an 'initial' one — a command rather than a function or an operator — by checking that the value of the token is less than $9C_{16}$ (the token for TAB(, the first of the 'non-initial' tokens). If the first character is neither an implied LET nor one of the initial tokens, the program ends via the syntax-error routine. If it is an initial token, the program does a little juggling with the token's value (explained below) to pick up the address for the relevant command's subroutine; then places the two bytes of this address on the stack; and then *jumps* (rather than calls) to the *BC* subroutine, so that the *BC* routine 'returns' to the BASIC command's routine rather than to the routine which actually called it. The RTS at the end of the BASIC command's subroutine *does* return to the original calling routine — the *A5FF* routine, so that the program then loops round to look for another BASIC statement. This juggling with the stack is a little complicated, especially as the addresses placed onto the stack (the apparent addresses for the command routines) are all one byte less than they should be — an RTS machine-code instruction adds one to the address it finds on the stack.

Tokens are functionally divided into three different groups: the 'initial words', the commands, from 80_{16} (END) to $9B_{16}$ (NEW); the operators (+, –, / and the others) and a few assorted functions like TAB(, THEN and STEP, from $9C_{16}$ to AC_{16}; and the

The 'execute BASIC statement' subroutine (A5FF)



functions, from AD_{16} (SGN) to $C3_{16}$ (MID$). The addresses for the commands are found by ignoring the top bit of the token (so that FOR's token 81_{16} becomes 01_{16}, for example) and then doubling it with an ASL A opcode; a TAY opcode then transfers this to the Y register; the address is then picked up by the machine-code commands LDA (A000),Y : PHA : LDA (A001),Y : PHA, leaving the routine's address (minus one!) on the stack, for the 'return' at the end of the *BC* routine to find. There are twenty-eight 'initial' tokens; their addresses thus occupy the first fifty-six bytes in the table, from *A000* to *A037*.

The next group stored in the table is not the operators, but the functions. These are called *within* the execution of the main BASIC commands, or rather by the 'evaluate an expression' and similar routines, and are processed by a subroutine at *AC27*. Much the same technique is used to get the address, using the ASL, TAY technique to double the token number (with high bit ignored) and point to the right pair of addresses in the table. The ASL is at *AC27*, the TAY at *AC55*; but the addresses are the true ones, not 'minus-one', because they are stored in memory rather than on the stack. The program stores them at *00A2, A3*, and jumps to them by doing a JSR 00A1 — a 4C (JMP) opcode being placed there by the cold-start routine, so the whole thing works out as a rather neat 'JSR indirect'. A confusing point here is that because there is a 'gap' in the table — the addresses for the operators' tokens have been skipped — the doubled token value is added not to *A000* but to an invented base address of *9FDE* in order to pick up the token's addresses. The address table for the function subroutines is stored from *A038* (SGN) to *A065* (MID$).

The reserved words, with the top bit set on the last letter of each, are stored in a table beginning at *A084*, and the error messages are stored in the same way (hence the odd graphics for the second letter of error messages on video systems) along with other assorted messages like ERROR and BREAK in a table that follows the reserved-word list from *A164* to *A1A0*. But there is an odd gap between the end of the function address table and the reserved-word list — and this turns out to be an odd list for the operators.

The first batch of operators, from $9C_{16}$ (TAB) to $A2_{16}$ (STEP) are called only within one command routine, and thus need no subroutines of their own. But the others *do* have their own subroutines; but the table stores *three* bytes for each, not two — the first byte being some kind of constant whose function we haven't yet deciphered. The addresses shown for each of the arithmetic and logical operators, from $A3_{16}$ (+) to AC_{16} (<), are collected in the same way as for the commands, by placing them on the stack; the addresses in the table are thus again one lower than the actual address of the relevant subroutine for each.

The following table, which summarises all this, should give you some indication of where to point your disassembler in the next stage of disentangling OSI's BASIC!

BASIC-word location	BASIC keyword	Token	Subroutine location	Location in table
A084	END	80	A639+1	A000
A087	FOR	81	A555+1	A002
A08A	NEXT	82	AA3F+1	A004
A08E	DATA	83	A70B+1	A006
A092	INPUT	84	A922+1	A008
A097	DIM	85	AD00+1	A00A
A09A	READ	86	A94E+1	A00C
A09E	LET	87	A7B8+1	A00E
A0A1	GOTO	88	A6B8+1	A010
A0A5	RUN	89	A690+1	A012
A0A8	IF	8A	A73B+1	A014
A0AA	RESTORE	8B	A619+1	A016
A0B1	GOSUB	8C	A69B+1	A018
A0B6	RETURN	8D	A6E5+1	A01A
A0BC	REM	8E	A74E+1	A01C
A0BF	STOP	8F	A637+1	A01E
A0C3	ON	90	A75E+1	A020
A0C5	NULL	91	A67A+1	A022
A0C9	WAIT	92	B431+1	A024
A0CD	LOAD	93	FFF3+1	A026
A0D1	SAVE	94	FFF6+1	A028
A0D5	DEF	95	AFDD+1	A02A
A0D8	POKE	96	B428+1	A02C
A0DC	PRINT	97	A82E+1	A02E
A0E1	CONT	98	A660+1	A030
A0E5	LIST	99	A4B4+1	A032
A0E9	CLEAR	9A	A68B+1	A034
A0EE	NEW	9B	A460+1	A036

A0F1	TAB(	9C	*(none)*	
A0F5	TO	9D	*(none)*	
A0F7	FN	9E	*(none)*	
A0F9	SPC(	9F	*(none)*	
A0FD	THEN	A0	*(none)*	
A101	NOT	A1	*(none)*	
A104	STEP	A2	*(none)*	
A108	+	A3	B46E+1	A067
A109	-	A4	B457+1	A06A
A10A	*	A5	B5FD+1	A06D
A10B	/	A6	B6CC+1	A070
A10C	^	A7	BAB5+1	A073
A10D	AND	A8	AC68+1	A076
A110	OR	A9	AC65+1	A079
A112	>	AA	BAEE+1	A07C
A113	=	AB	ABD7+1	A07F
A114	<	AC	AC95+1	A082
A115	SGN	AD	B7D8	A038
A118	INT	AE	B862	A03A
A11B	ABS	AF	B7F5	A03C
A11E	USR	B0	000A	A03E
A121	FRE	B1	AFAD	A040
A124	POS	B2	AFCE	A042
A127	SQR	B3	BAAC	A044
A12A	RND	B4	BBC0	A046
A12D	LOG	B5	B5BD	A048
A130	EXP	B6	BB1B	A04A
A133	COS	B7	BBFC	A04C
A136	SIN	B8	BC03	A04E
A139	TAN	B9	BC4C	A050
A13C	ATN	BA	BC99	A052
A13F	PEEK	BB	B41E	A054
A143	LEN	BC	B38C	A056
A146	STR$	BD	B08C	A058
A14A	VAL	BE	B3BD	A05A
A14D	ASC	BF	B39B	A05C
A150	CHR$	C0	B2FC	A05E
A154	LEFT$	C1	B310	A060
A159	RIGHT$	C2	B33C	A062
A15F	MID$	C3	B347	A064

Designs on OEM

— or how to dedicate your Superboard

A string of fairly random comments and discussions with dealers and with one of our members who uses Superboards as weighing-machine controllers started me thinking along these lines — that the Superboard can be modified surprisingly easily to work as a process controller in applications where BASIC is fast enough for the job.

This is not as daft as it sounds. Sure, a Superboard costs a lot more than, say the Acorn as a process controller; but designing and implementing a control program that takes more than half a dozen factors into account is no joke in machine-code, especially if one of the factors is collecting operator input. Hardware is soft, software is hard — as the expression goes — and anything that is hard is expensive. For many applications, the relative simplicity of software design in BASIC easily recovers the higher initial cost of the hardware.

So how can this be done with the Superboard? The answer, of course, is to start at the beginning — in this case the 6502's reset vector — and throw away anything which is not actually needed for the job.

On reset, the existing monitor jumps to the message 'D/C/W/M' — of which we can presume that a small controller will be needing neither discs nor user-access to any machine-code monitor. This leaves Cold or Warm start for BASIC. It should not be difficult to implement a hardware 'power-on reset', leaving the 'reset', as far as the user is concerned, to be a direct warm-start to the BASIC program rather than a choice of entry to cold or warm start. The existing vectors for cold and warm start are to *BD11* and *0000* respectively.

The trick here would be to duplicate the functions of the existing cold-start routine without bothering with the existing user-defined calls for memory-size or terminal width — since these will be predetermined by the application in hand rather than the operator's whims. As long as your new routine (presumably held in a replacement for the existing monitor ROM) does everything that the existing cold-start routine does, setting up the various flags and limits like memory size, and finishes by placing the warm-start vector in *0001, 02*, a new entry with your own messages is practicable. The other vector that BASIC wants is a warm-start one, stored in *0004, 05* by the cold-start routine. In the existing system this points to the message-printer routine at *A8C3*, in order to print the 'OK' message on warm-start; but there is no reason why it should not be reset by a new cold-start routine to point somewhere else, such as a direct RUNning of a BASIC program in EPROM.

There is space on the board, and the address decoding, for up to 6K of additional PROM — if you know where to look. The board has been designed to handle a single 8K BASIC chip as well as the existing four 2K ROMs; the address decoding for the 8K chip goes to the last of the four ROM sockets. There is also a chip-select line on the schematic that is described as 'not-BAS', and appears to select the other three sockets independently of the fourth. By inserting a small board with four sockets into the last BASIC socket, and moving the BASIC ROMs up to it, changing the board links accordingly, would free the other three sockets for three additional 2K PROMs — enough for a 6K BASIC program. It does not seem possible, though to replace the 2114 RAMs with EPROMs — to my knowledge there is no pin-compatible EPROM.

Designing BASIC software to go into an EPROM should not be too difficult. The

only tricky point is that the BASIC pointers for variables, arrays, strings and the like all point *above* the program space. If the existing BASIC is to be used, any scratchpad RAM for BASIC use will have to be *above* the program. This means that the program will have to end on the boundary of a PROM, or else variables and the like will be lost by trying to write into the PROM space instead of into the RAM above. This is easily solved, though, by 'padding out' the final program with REM statements. For the same reason, it would also be sensible to reset the start of BASIC program space (currently defined , by the existing cold-start routine, at *0301*, preceded by a null) to a higher page boundary on the boundary of a ROM, such as at *0400* or, more practically, at *0800*, during the new cold-start routine.

A detailed study of the existing cold-start routine from *BD11* to *BE38* should clarify most of the problems involved in designing software/firmware for OEM applications. Let us know how you get on!

Tom Graves

Disc System Notes

A warning on hard discs

Hard discs sound like a really nice idea — enormous mass memory that's fast enough to use as RAM. But there's an old saying about 'all your eggs in one basket'. . . — in other words reliable back-up. Since the whole point of hard discs is that they are enormous and fast, any back-up system is thus, by comparison, small and slow. But back-up is essential — don't skimp it!

A nastier problem also arises for UK users of hard discs, regardless of make of either hard disc or computer system. If, or rather when, they develop any sort of fault, *they cannot be repaired in the UK*. They have to be dismantled in a special 'clean room' — and the nearest one is in Munich. . . while most, of course, are in the States. Shugart, we're told, intend building a 'clean room' here sometime, but there's no definite date as yet. Sending the disc to the States means a long, expensive, disc-less wait — and unless you've done that back-up that you should have done, you'll be without your vast data-base too, since that, of course, is on the disc. . .

OS-CP/M — an assortment of errors

The early versions of OS-CP/M had their assorted bugs, as can be expected; the trouble is that the newer versions have newer bugs. The COBOL implementation apparently *still* doesn't work (but I'm told that it's not all that unusual for COBOLs to crash). There are, however, other errors of a more common kind.

The May '80 issue of *Dr Dobbs Journal* (Vol.5 No.5) had an article and a letter pointing out several of these problems. One is a problem with TAB (as CHR$(9), ASCII's HT command); but more serious are the two bad I/O bungles. The first is that the serial ports starting at *CF00* (the multi-user 550 board) are initialised correctly on boot-up by the track-0 routine, but are then 'clobbered' by part of the loading sequence for CP/M — the system locks up if any attempt is made to use those ports. The other and even more stupid error is that OSI made no allowance for the fact that the Microsoft Altair BASIC's IN, OUT and WAIT commands are written to use the Z-80's parallel ports. But the C3 doesn't have parallel ports; it uses the 6502's serial ports (the ACIAs) instead; so these commands lock up — not surprisingly — in trying to pass parallel information through serial ports!

The article in *Dr Dobbs* describes fixes for these and a few other problems — so get hold of a copy while you can. The series of articles on C and tiny-C in that issue are interesting, too.

65U — POKEs about INPUT

We had a note from *SJC Fox of Microcode* (65 Landswood Park, Hartford, Northwich, Cheshire) about some useful POKEs in 65U. He writes:

POKE 2972,13: POKE 2976,13 enables BASIC to input a complete line, commas and all. Values of 58 (ASCII :) and 44 (ASCII ,) respectively restore the normal operation of INPUT. This came from one of OSI's programs — DMS I think.

POKE 2888,0 disables the break on carriage-return-only in INPUT and returns a null to string inputs and 0 to numeric inputs. A value of 27 restores the normal behaviour. This is my own modification of an OS-65D POKE and *appears* to have no bad side effects.

Following on from last issue's notes on date storage in 65U, does anyone know of any other 'empty' areas that can be used as permanent storage? And does anyone know of a POKE to simulate *ctrl-D* in 65U (the 'one page at a time' effect), and *ctrl-W* to clear *ctrl-D*?

65D (V3.1) — I/O distribution

For users interested in writing their own I/O routines, we found a note from OSI that describes the look-up tables for the input and output routines. The input and output routines are selected by bits being set on a flag byte (at 8993 for input, 8994 for output — both locations decimal). 65D collects the addresses of the chosen I/O subroutines from a table (from 2301-2320_{16}), shown below. Note that the stored 'address' is always *one lower* than the true address of each routine. The system uses a 'stack trick' to reach the routine, like BASIC-in-ROM's way of collecting its commands — see the article on BASIC execution elsewhere in this issue.

Flag (at 2337_{16}) Input table

Flag bit	*Addr.*	*shows:*	*Subroutine and address*	
0	2301	F5 24	Console serial port	24F6
1	03	2A 25	Polled keyboard	252B
2	05	17 25	430 board	2518
3	07	85 23	Null input	2386
4	09	88 23	Memory	2389
5	0B	A0 23	Disk device □6	23A1
6	0D	EF 23	Disk device □7	23F0
7	0F	AF 24	550 board (serial ports)	24B0

Flag (at 2338_{16}) Output table

Flag bit	*Addr.*	*shows:*	*Subroutine and address*	
0	2311	CC 24	Console serial port	24CD
1	13	98 25	440/540 video	2599
2	15	0C 25	430 board	250D
3	17	9E 24	Line printer	249F
4	19	8F 23	Memory	2390
5	1B	B1 23	Disk device □6	23B2
6	1D	02 24	Disk device □7	2403
7	1F	BC 24	550 serial board	24BD

65D — copying mini-floppies on single-drive systems
Another note from OSI purported to 'explain the proper procedure' for doing this. It doesn't, of course; it was clear (because the note *wasn't* clear) that if you followed their instructions to the letter you would end up very quickly with an initialised — and thus erased — original system diskette.

Since the procedure is obviously a little long-winded under 65D, would a member with practical experience of this please send us a note on the true 'proper procedure'?

65D — failure on write retry DONE
Another OSI note, this time a little clearer. A problem that apparently manifests sometimes on both 5" and 8" floppy versions of both 65D V3.0 and V3.1 (the current issue), manifesting as a system failure if a retry occurred on a write to disc. OSI's notes cover the whole 'conversation', including all machine output; we show only the operator input, with any essential machine prompts (and comments) in italics. Use a CR (carriage-return) on each line unless otherwise stated. From boot:

UNLOCK
EXIT
EM *enter extended monitor, for use later*
EX *exit to DOS, but leave ExMon in language area*
CALL 0200=01,2 *diskette utilities: for mini-floppies* CALL 0200=13,1
GO 0200
2 *select track-0 read/write utility*
R4200 *read track-0 into* 4200_{16} *on*
E
RET EM *back to ExMon*
a4886
4886/13 28
a4898
4898/D0 4C *(line-feed)*
4899/6F 09 *(line-feed)*
489A/60 28
EX
GO 0200
2
W4200/2200,8 *write new version into track-0*
E
BASIC *and exit to BASIC* DONE

65D — fix for assembler 'hang-up' on C1/Superboard disc only
Another OSI note: to quote, 'the sequence below will correct the problem with the assembler 'hanging up' on a CH-1P'. (OS-65D V3.0 and V3.1). After boot, unlock BASIC, exit to DOS and call EM (ExMon). Then:

a1563
1563/6D 9F *(line-feed)* WRONG
1564/15 24
EX *exit to DOS* CALL 4200=06,1
SAVE 06,1=1200/5
4A63
4A64
3500

65D — fix for ExMon to print 6502's registers on breakpoint
Also from OSI, for V3.0 and V3.1, and evidently for mini and full floppies. Get into ExMon after boot, then:

!CALL 4700=07,1 *for mini-floppy use* !CALL 4700=10,1
a4B68
4B68/B8 C2
!SAVE 07,1=4700/9 *for mini-floppy use* !SAVE 10,1=4700/8
and exit as required.

65D — fix to permit random file access beyond □383
Another OSI dealer-issue note, for V3.0 and V3.1. Enter ExMon after boot, then:

!CALL 4E79=08,4 *for mini-floppy use* !CALL 4E79=12,4
a4F00
4F00/30 30 *(line-feed)*
4F01/16 65
a4F18
4F18/AD EA *(line-feed)*
4F19/92 EA *(line-feed)*
4F1A/2F EA *(line-feed)*
4F1B/18 F8 *(line-feed)*
4F1C/F8 18
a4F67
4F67/00 AD *(line-feed)*
4F68/00 92 *(line-feed)*
4F69/00 2F *(line-feed)*
4F6A/00 F0 *(line-feed)*
4F6B/00 AF *(line-feed)*
4F6C/00 F8 *(line-feed)*
4F6D/00 18 *(line-feed)*
4F6E/00 AA *(line-feed)*
4F6F/00 A9 *(line-feed)*
4F70/00 00 *(line-feed)*
4F71/00 69 *(line-feed)*
4F72/00 01 *(line-feed)*
4F73/00 CA *(line-feed)*
4F74/00 D0 *(line-feed)*
4F75/00 FB *(line-feed)*
4F76/00 F0 *(line-feed)*
4F77/00 A4
EXIT
SAVE 08,4=4E79,1 *for mini-floppy use* SAVE 12,4=4E79,1

65D and WP-2 — fix for output problems
OSI note: corrects problem with (540 board) video based systems 'hanging-up' under WP-2 whenever an L command was given; also corrects a general problem in 65D and WP-2 when outputting to higher numbered devices. The problem surfaced if a *ctrl-C* or *ctrl-S* was entered while outputting to a given device and a higher numbered device. The output would appear on the lowest numbered device, but not on the higher numbered device. To install the fix, enter ExMon after boot, then:

!CALL 0200=01,2 *diskette utilities: for mini-floppy use* !CALL 0200=13,1
If changes are being made to WP-2, insert WP-2 into drive 'A' at this point.
!GO 0200
2 *select track-0 read/write utility*
R4200 *read track-0 into* 4200_{16} *onwards*
E
RET EM *return to ExMon*
a4339
4339/AD A0 *(line-feed)*
433A/21 00 *(line-feed)*
433B/23 AD *(line-feed)*
433C/A0 21 *(line-feed)*
433D/00 23 *(line-feed)*
433E/F0 D0
a434D
434D/4A D0 *(line-feed)*
434E/E8 22 *(line-feed)*
434F/90 E8 *(line-feed)*
4350/09 4A *(line-feed)*
4351/48 90 *(line-feed)*
4352/8A 09 *(line-feed)*
4353/48 48 *(line-feed)*
4354/20 8A *(line-feed)*
4355/71 48 *(line-feed)*
4356/23 20 *(line-feed)*
4357/68 76 *(line-feed)*
4358/AA 23 *(line-feed)*
4359/68 68 *(line-feed)*
435A/E0 AA *(line-feed)*
435B/07 68 *(line-feed)*
435C/D0 D0 *(line-feed)*
435D/EF F1
a4371
4371/0A 8C *(line-feed)*
4372/8D 78 *(line-feed)*
4373/78 23 *(line-feed)*
4374/23 D0 *(line-feed)*
4375/98 D9 *(line-feed)*
4376/18 0A
!GO 0200
2 *recall track-0 read/write utility*
W4200/2200,8 *write new version into track-0*
E *back to DOS*
and exit as required.

DONE

Note: our typesetter has most but not all ASCII characters: two substitutions have to be made, namely a for the 'at' character (ASCII 40_{16}) and □ for 'hash' (ASCII 23_{16}).