OSI UK User Group Newsletter

Vol.1 No.1 December 1979

A beginning

Just what you always wanted for Christmas: an OSI User Group to go with your Challenger computer! And while we will be reviewing games software and similar lightweight matter, our real purpose for starting the User Group and this Newsletter is to share practical information on the use and uses of OSI small computers. OSI's hardware is superb, but their documentation ranges from the mediocre to simply appalling, particularly at the machine code level. We know that the BASIC is fast (and we'll discuss ways of making it still faster), but neither OSI nor Microsoft supply any information on how BASIC works, how it stores information, or the start addresses of even the more important routines such the screen and I/O handlers. We're already making a start in the Newsletter by serialising the notes on OSI's BASIC-in-ROM collected by the American Challenger software specialists Aardvark, but that is only a start — the rest is up to us as a group of users, and that, we imagine, includes you. (Minor problem, by the way: Microsoft don't like people publishing their machine code, since they don't want people pirating their hard work; but we can certainly publish lists of entry points to subroutines, since it is the code itself which is copyright.)

Our other main reason for being is to act as a forum for exchange of software, hardware and application notes and ideas, both directly through the User Group and rather less directly, but rather more publicly, through this Newsletter. We all use Challenger 2s, so the contents of this Newsletter apply mostly to those machines; but our aim is to cover the whole range of the OSI small systems, and also related systems like Comp's UK101. We all have our applications for our machines too: the Editor uses his as the central processor for an offline terminal to a typesetter in his design business (the reason why this Newsletter can be typeset!), and another colleague uses a C2 as the controller for an experimental sound system. How do you use your machine? Let us know. We want to look in detail at one application each issue; and the way in which you use your machine may well solve an application puzzle for someone else.

The same applies to software and hardware. OSI give no real information on how to connect their small systems to anything non-standard, while the computers themselves, with everything easily accessible, are ideal for that kind of work. And software. . .software is what the whole thing depends on. Let us know what *you* are doing! By the time this issue gets out 'onto the streets' the Editor's experimental typesetting terminal should be up-and-running — which will allow us to typeset BASIC program and assembly-language listings direct from Challenger- or KIM-format tape. (We'll also offer this as a low-cost service for people interested in selling software for OSI systems, both to encourage software houses to draft their software in an OSI-compatible form, and to encourage them to produce decent documentation while they're at it!)

228 4837

The Group and Newsletter arose originally from suggestions made by Mutek, at Box, and their help is reflected often in the contents of this issue. But we, and they, are keen that this venture should not be dependent on any dealer — we want every dealer to be involved. So dealers — both hardware and software — let us know who and where you are, and what services you offer. In addition to a 'Dealers' Notes' section (free! — we want to know about you), we'll also be making advertising space available — let us know soon if you're interested.

OSI UK User Group

Organisers:
George Chkiantz *and* Richard Elen: 12 Bennerley Road, London SW11 6DS.
(Newsletter subscriptions, technical queries, general discussion and other usual User Group matters)

G.E. Davies: 5 Houndelee Place, Newcastle-upon-Tyne NE5 8AJ.
E. Turnbull: 12 Cheyne Road, Prudoe, Northumberland NE42 6PF.
(Northern area contacts)

Newsletter
Tom Graves: 19a West End, Street, Somerset BA16 0LQ.
(Editorial matters, contributions, advertising and anything to do with the Newsletter other than subscriptions)

Prices, pirating and all that

Early in '79 I was looking around for a simple but versatile microcomputer to use as the controller for a typesetter terminal. The old PET with its unusable keyboard was out; likewise the TRS-80 with its awful lower-case characters and keyboard bounce. I'd thought of a Nascom-2, but nasty experience with a Nascom-1 that insisted on altering code by itself on Reset had made me rather wary — and anyway, as expected, the Nascom-2 didn't actually materialise until months after its 'launch' date. So an ad for OSI kit rather caught my eye — for a while. The spec for the C2 was right — just right — but the price. . . !

The disc version of the C2 was quoted at £1595. The basic 4K version was £620. Excluding VAT. While the new 16K PET (much improved, but still — for my purposes — useless), with a reasonable keyboard and built-in VDU was a mere £675. Forget it . . .

Until a friend told me about the computer he was using in his experimental sound system. A 4K C2-4P, he said. Very fast BASIC. Very adaptable, easily interfaced to his other kit. And very good value at £349, too.

£349? £620? Something very wrong here! Or something very right; or both.

I bought my C2-4P from Mutek at the above-mentioned £349, plus a further few pounds for a further 4K, plus the dreaded 15%. And I wondered how one dealer could quote a price of almost 60% of that of the 'official' dealers, and yet still give good service and make a living too.

But the answer is not quite as obvious as it might at first appear. Rip-off there almost certainly was, but it seems to have been more on the other side of the Atlantic than here. The British official dealers, of whom Mutek was once one, had been placed in a ludicrous position by OSI. If they wanted to be official dealers, they had to buy their systems (or rather hand over their money and wait. . .and wait) from the company whom OSI had contracted as their European distributors. This was, and still is, American Data Home and Office Computing (Adhoc for short). And Adhoc's wholesale price for OSI kit for those privileged British official dealers turned out to be identical to the American *retail* price for the same OSI kit (then $598 for the 4K C2-4P). Before shipment costs. Before Customs duty. Before adding a realistic amount for sales, for technical back-up and just plain living. No wonder the prices were high!

Let us simply say that, apart from the occasional Superboard, no dealer in this country buys OSI kit through Adhoc. (This also means that, except on those Superboards, no one is really entitled to call themselves an 'official OSI distributor'.) Everyone buys direct from American national wholesalers, who have the kit in stock and who give realistic dealer discounts. And while prices still vary, they are now far more realistic and competitive. Mutek may still be the cheapest, but most of the others are not far behind: Microcomputer Business Machines (a federation of some of the other former 'official' dealers, headed by Mark Strathern of Lotus Sound) offer the 4K C2-4P for £404 at present. We may still make asides about excessive profits in certain cases — the occasional 'get-rich-quick' dealer — but in general OSI kit is now being sold at the right sort of price. For anything other than absolutely standard off-the-shelf applications, OSI small computers are now better value than the equivalent PET, for example.

Where the PET and the other 'big-seller' machines still have the advantage is in packaged software (but then, in the case of the PET at least, they're not too brilliant for anything else . . .). And it is here that our other problem with prices and the like arises. Even experienced programmers need packaged utilities like assemblers and a decent monitor, and most of us like to play the odd idiot game once in a while at least. But where do we get them from? OSI's software — if you can get it from anyone — is cheap-ish but definitely 'curate's egg'. Aardvark — as far as I know the only other producer of packaged OSI-compatible software — produces tapes which could be called 'bishop's egg': generally excellent but occasional howlers! And again, the prices vary: Aardvark's American prices are $5.95, and while at least one British dealer sold those tapes at the usual — and excessive — dollars-equals-pounds conversion, Mutek sold theirs at a typical £2.60.

The gap is enormous, but Mutek's reasoning is simple: first, software is the 'drip-feed', and at their prices (which they make enough on if they sell in good quantity) most people will buy at least one or two on a visit; second, they regard reasonably priced software as part of the technical back-up for their systems (with good packaged software you can *prove* in many cases that apparent hardware problems are in fact complex programming errors by the user); and third, they feel that their prices are what the software is worth, and that selling the tapes for what they are worth is the best protection against pirating.

Pirating is something which worries every dealer in software (it worries the record companies too, for the same reason — they lose a lot of money on it). Like the record companies, software houses go to a great deal of trouble to protect their product in every way they can — which is probably the main reason why OSI's documentation of its machine-code is almost non-existent. But we have a problem where the record companies don't — namely, that there is almost no way to distinguish between a legitimate back-up copy, a mildly but still expensively illicit 'copy I did for Pete in Manchester', and the real illegal bulk copying of software. And for every programming master who invents a trick to stop you seeing his (or her) listing — and probably prevents you from taking that all-important back-up copy — there will be another programming master to work out a way of getting round it. Stalemate.

It seems to me that whole problem comes down to two areas: amateur versus professional attitudes, and respect.

There are so many erstwhile amateurs now working as professionals in the personal computing field that the distinction may seem odd; but the attitudes usually remain the same. The professional is in it for a living: that comes first, and any interest in what is being done for that living comes a definite second. Thus, as far as pirating of software and the like is concerned, the attitude is one of 'protect at any cost' — assuming the opposition can be made to pay the legal fees. The amateur is in it, literally, for the love of it; that comes first, and the fact of making a living out of it, while a real necessity, still comes second. Copying tapes is a matter of friendly exchange, of keeping in touch, of sharing — doing it for the love it — without actually noticing that those copies are hurting the original producers financially. Briefly, an over-professional attitude can — and does — develop into an arrogant state of 'get-rich-quick and damn the consumers'; an over-amateur attitude leads to bankruptcy for someone. A matter of respect all round, I think: it's certainly about time that some dealers at least realised that they're dealing with people, not with walking credit-cards; and amateurs, too, should think of what it really costs before making 'that copy for Pete'.

And if you were about to copy a tape for Pete, stop for a moment. Some of the programs you've developed are probably marketable: even if you might not think so, other people — the potential users — may well do. One dealer commented to me that a customer mentioned in passing that he'd developed a couple of games and quizzes, but didn't think they were particularly brilliant; but I've seen them, and I'm glad to say that his 'British Towns Quiz' will be on the market soon, for his handling of the map certainly *was* brilliant. The American software is generally mediocre — let's get some together to sell over there! Anyone interested in selling software they've developed for OSI small systems might like to get in touch with the User's Group — we've already arranged a provisional marketing deal with Mutek, and other dealers will, we hope, follow suit. Let's hear from you; let's see what you've been working on, and what your opinions are on the problems of prices, pirating and all that.

Tom Graves

Very useful BASIC routines

00BC
Works its way through a line of BASIC (or whatever C3, C4 points to) and gets the next character each time it is called. It will be pointing to the end of your USR statement if you call it from the USR; you can then use it to get stuff after X=USR(Y) — and BASIC will never be the wiser! BC leaves carry set if character is numeric.

00C2
Entry to the BC routine without incrementing C3, C4 before getting the character. Thus it gets the current character.

A477
Call this routine and then jump to ~~**AE52**~~ ASC2 and you'll be RUNning the current BASIC program — starting from machine language!

A925
Call this from a USR statement and you will be doing an INPUT statement — but BASIC will not echo the characters you type in, including the carriage return/line feed at the end. This gives a real BASIC INPUT statement that screw up your nice graphics by scrolling the screen one line! You must set loc 64 to $80 (set the CTRL O flag) before all this works. Do an LSR $64 to clear the flag to normal if you want BASIC print statements to work again.

AAAD
One you've been waiting for. This gets a 16-bit argument from the current BASIC line position (yes, like right after the ')' of your USR statement!), evaluating whatever expressions it finds, and leaves it where a call to **AE05** will find it and put it in AE, AF! (Use **AC01** to find a comma and then call **AAAD** again to get another value!)

AAC1
Like **AAAD** but no type mismatch check.

ABF5–ACOC
This series of routines (actually of entry points to one routine) uses the **BC** routine to check for various delimiters. If you disassemble the ROM here, it demonstrates a classic use of the 2C opcode as a combination NOP and immediate load, depending on where you jump in. **ABFB** checks for ')'; **ABFE** for '('; **AC01** for ','; **AC03** for whatever character you leave in A when you call it. **ABF5** checks for '(', calls **AAC1** to get a value, then checks for ')'. (Thoughts of a BASIC statement X=USR(Y)(Z) should be jumping into your head about now.)

AD0B
This routine uses the **BC** routine to find the name of the variable that's next in the BASIC line, and puts the address of the variable in locs 95, 96. It also leaves the address in A, Y. If you store A in 97 and Y in 98, you can call OUTVAR (**AFC1**) to store whatever 16-bit value you put in A and Y into that BASIC variable.

B3AE
This is like **AAC1**, but gives an error if the value is greater tnan 255_{10}. (Used by the POKE routine to keep you from putting a too-big number in memory.)

B962
Prints the decimal value of whatever 16-bit number is in AD, AE at the current cursor location on the screen, with normal BASIC checks for line length (does auto CR/LF if line is too long) etc.

(Courtesy of Aardvark Technical Services)

Memory locations containing items of interest

These notes were supplied by Aardvark Technical Services (in the States) via Mutek, and, to quote, "do not claim to be complete or even error-free". Even so, they and the list of BASIC-in-ROM subroutine addresses from the same source do give us a very good start. Let us know of any routines you discover, so that we can make them available to other OSI users.

000B,C	Address of USR routine
000D	Number of extra nulls to be inserted after carriage return.
000E	Number of characters since last carriage return.
000F	Terminal width (for auto CR/LF).
0010	Terminal width for comma-spaced columns.
0013-5A	Input buffer.
005F	String variable being processed flag (?)
0061	?
0064	CTRL O flag (bit 7 set = suppress printing).
0065	sometimes contains $68 (??)
0079,7A	Pointer to initial null of BASIC program workspace.
007B,7C	Pointer to beginning of BASIC variable storage space.
007D,7E	Pointer to beginning of BASIC array storage space.
007F,80	Pointer to end of array space/ beginning of free memory.
0081,82	Pointer to end of string space/top of free memory.
0085,86	Pointer to top of memory allowed to be used by BASIC.
0087,88	Current line number.
0089,8A	Sometimes next line number (?)
008F,90	DATA pointer.
0095,96	This is where *AD0B* leaves address of the variable it found.
0097,98	Address of variable to be assigned value by OUTVAR *(AFC1)*.
00AA,AB	Points to pointer of next BASIC line after LIST.
00AD,AE	The contents of this pair is printed in decimal by *B962.*
00AE,AF	This is where INVAR *(AE05)* leaves its argument.
00D1-D7	Clobbered by OSI's Extended Monitor disassembler; kills BASIC.
00E0-E6	Apparently unused page zero space.
00E8-FF	Apparently unused (by BASIC) page zero space.
00FB	ROM monitor load flag.
00FC	ROM monitor contents of current memory location.
00FE,FF	Address of current ROM monitor memory location.
0130	NMI routine.
01C0	IRQ routine (can be overwritten by stack being used by BASIC.
0200	Current screen cursor is at D700 + (0200); initialised to (FFE0).
0201	Save character to be printed.
0202	Temporary storage used by CRT driver *(BF2D).*
0203	LOAD flag ($80 = LOAD from tape).
0205	SAVE flag (0 = not SAVE mode).
0206	Time delay for slowing down CRT driver.
0207-0E	Variable execution block — code for screen scroll — not re-usable, but stored at BFF3-BFFA.
0212	CTRL C flag (not 0 = ignore CTRL C) (reset by RUN).
0213-16	Polled keyboard temporary storage and counter.

A000-37	BASIC initial word jump table (in token order; add 1 to each address).
A038-65	BASIC non-initial word jumps (real entry addresses).
A084-163	BASIC keywords in ASCII; bit 7 set as delimiter; in token order.
A164-86	Error messages with null as delimiter (printed by *A8C3*).
BE4E	"Written by" message.

Miscellaneous ROM BASIC routines

0000	Warmstart (4C 74 A2)
0003	Message printer (4C C3 A8).
00A1	General purpose JMP instruction: put target address in A2, A3.
00BC	Get next character in BASIC line.
00C2	Get current character in BASIC line.
A1A1	Look back through stack (??).
A212	Check for OM and stack overflow.
A24C	'OM' error.
A24E	Error caller; sets X-register to error code.
A274	Warmstart entry.
A357	Input and fill buffer; put null at end.
A386	Input from FFEB.
A399	Toggle CTRL-O flag.
A432	Find BASIC line whose number is in 11, 12; put address of pointer of that line in AA, AB.
A477	Point C3, C4 to 0301; reset string and array pointers; reset stack to (01)FC; put 0301 8F, 90; 0 in 8C; 0 in 61; 68 in C5 (?).
A491	Clear stack; 0 in 8C and 61.
A5C2	Top of main BASIC execution loop.
A5FC	Entry to BASIC execute loop.
A5FF	Do line of BASIC.
A629	Jump to FFF1 to check for CTRL C.
A636	CTRL C entry point.
A67B	Set null count at D0 (?).
A77F	Get decimal number from buffer; put value in 11, 12.
A866	Put null at end of buffer; CR/LF; nulls.
A86C	CR/LF with nulls from 0D.
A8C3	Message printer: A, Y registers point to message, which ends with null.
A8E0	Output " ".
A8E3	Output "?".
A8E5	Output character in A register; update 0E; check line length.
A925	Input routine less clear CTRL O.
A946	Output "? "; jump to A357.
AAAD	Get 16-bit argument from BASIC line; *AE05* will put value in AE, AF; does type-mismatch error check.
AAC1	Like *AAAD* but with no TM error check.
ABA0	Put 0 in 5F; get character; go to *B887* if numeric??
ABD8	16-bit complement using *AE05/AFC1*?
ABF5	Checks for "(", calls *AAC1*, checks for ")".
ABFB	Syntax error if next character not ")".
ABFE	Syntax error if next character not "(".
AC01	Syntax error if next character not ",".
AC03	Syntax error if next character not what's in A register.
AC0C	SN error printer.

AD0B	Get variable name from BASIC line; put address of variable in 95, 96 and A, Y.
AD53	Expects variable name in 93, 94; finds address of variable and puts it in 95, 96 and A, Y; 0 in 61.
AE05	INVAR: puts 15-bit signed value in AE, AF.
AE85	BS error.
AE88	Function call error.
AFC1	OUTVAR: 0 in 5F; contents of A register in AE; of Y register in AF; then to ??
B0AE	Part of message printer *(A8C3)*; calculates length of message?
B3AE	Put 8-bit argument from line in AE, AF.
B3F3	(BA, BB) to C3, C4.
B4D0	Arithmetic to normalize floating-point argument??
B887	Check for +, –, $, , ., E, etcetera... long!
B95A	Prints current line number.
B962	Prints contents of AD, AE as decimal number.
BD11	Coldstart entry point.
BEE4	UART input routine (S1883 chip at FB0X).
BEF3	UART output routine.
BEFE	UART initialisation.
BF07	ACIA input (6850 chip at FC0X — like C2-4P).
BF15	ACIA output routine.
BF22	ACIA initialisation.
BF2D	CRT driver.

Courtesy of Aardvark Technical Services

Disentangling USR

The description of the USR function in the ROM BASIC 'manual' supplied with C2-4Ps is a classic example of OSI's sludgeware. The description given on its pages 13 and 14 is essentially correct (although quite indecipherable) except for one glaring error. Despite what it says, don't POKE the low-and-high addresses of the start of your machine-code routine into 23E and 23F — it doesn't work. (Does anyone know what they *do* do? The addresses are given in the 'manual' in hex rather than decimal, which implies a machine-code rather than BASIC use for a start.) The locations that *do* work are 11_{10} and 12_{10} (Z-0B and 0C), as mentioned in the note on machine-code screen clear in the Graphics handbook.

In practice there appear to be two separate USR functions: X=USR(X) and P=USR(Q). X=USR(X) is a simple in-and-out call to a machine-code subroutine that carries no value from or to BASIC — a typical example being a screen-clear. P=USR(Q) is more tricky but more useful. As described in the manual, two routines embedded in the BASIC ROMs will pick up a value from Q and return it after processing to P, so that P does equal USR(Q). The two routines are INVAR *(AE05)* and OUTVAR *(AFC1)*. *AE05* processes the expression within Q's brackets ('Q' may be a full expression rather than a mere variable label) and places it as a fifteen-bit signed integer into zero-page AE and AF, the lower eight bits being in AF. To return the processed value to BASIC, *AFC1* is called: the A register contains the high order of the value, or else must be zeroed, and the low order is contained by the Y register; the combination is then loaded, as a fifteen-bit integer again, into the storage space for the BASIC variable P (or whatever precedes the = sign in the USR statement). The comment in the manual that 'if this function is not called USR(X) will equal X' does *not* mean that P will contain the value of the X register; in practice it is either unchanged, equals the value of X (or Q, or whatever) prior to the USR statement, or is loaded with garbage.

The comment that 'the routine pointed to by 6 and 7' (and later, 8 and 9) 'should be called' is a reference to the fact that the cold-start routine loads these addresses with 05,AE and C1,AF respectively. But there is no 'JSR-indirect' in 6502 code, and the use of a JSR to a JMP-indirect (the 6C opcode), while often used in BASIC because of the fixed nature of the ROM, is relatively inefficient (both in memory use and in programming complexity) compared to a simple JSR call. It's simpler to ignore these zero-page addresses!

Note that the specific form X=USR(X) appears to be a kind of reserved word: I've tried using it in the same way as for P=USR(Q), calling OUTVAR, and the result is a complicated garbage that I haven't yet disentangled. X=USR(Q) is fine, though, as are P=USR(X) and Q=USR(Q). I did my tests on these using an ASL A (0A opcode) to double the value supplied by ( ), and seeing what came out at the other end.

According to OSI only one value can be transferred to and from BASIC with a single USR call, but there are ways round this. In their notes which we are serialising in this newsletter, Aardvark point out that ROM BASIC subroutines can be used to transfer a limited array from BASIC to machine-code, via a statement of a form such as P=USR(Q)(R)(S)... or P=USR(Q),R,S..., and BASIC variables can be assigned values directly from machine code. Details are given in the section headed 'Very useful BASIC routines' elsewhere in this issue.

The tedious part of programming with USR is the string of POKEs needed to give the start of a string of machine-code routines. Calling the USR function without a preliminary POKE generates an FC (function call) error message and program crash. This is because the cold-start routine places the start-address of the FC routine *(AE88)* into 11_{10},12_{10}, which is used in default of further instructions. Perhaps the simplest way of organising the POKEs would be to lay each USR routine call as a series of BASIC subroutines, to be called by ON X GOSUB, where X is assigned its value either by BASIC or (by OUTVAR, *AFC1*) in machine-code.

Tom Graves

Software — some research needed

Stack requirements of ROM subroutines

USR calls from BASIC are only allowed 16 stack levels. At first glance this appears to be little limitation: eight nested levels of subroutines is a lot. But those code-saving ROM routines are often greedy on stack use, partly for internal subroutine calls, partly for PUSH instructions for protecting registers. *FEED* (or *FD00* — same thing), the keyboard input, uses seven stack levels including the call to the routine, and *A8C3*, the message printer, appears to use *nine*. So beware!

What we really need is a complete list of the stack requirements of subroutines — and, if possible, a complete list of the overall effects that the routines have on the registers and on obvious 'parking space' such as the zero page and the first part of page 02. I've already made a start on the monitor of the C2-4P and the display end of its ROM BASIC (which is also used for C1/Superboards and, to a large extent, UK101s), if anyone needs that info now. If you're playing around with a dis-assembler, note these things down and send them to us, so that we can publish a complete list as soon as possible.

Applications software, anyone?

In future issues we want to be looking in detail at not-so-off-the-shelf applications of OSI kit, which includes the software developed as part of those applications. Some of it gives useful spin-off for other possible applications: *Richard Elen* and *George Chkiantz* have some nice real-time bar-graph programs used in testing their experimental sound system, for example. Graeme Davies has the foundations of a distributed processing system. My main project is a modular text processing and formatting system to allow flexible input, code-conversion, editing, formatting and page-make-up of text for phototypesetters (initially A-M's Comp/Set® series, but later other types as well); the immediate spin-off from this project is a mixed BASIC and machine-code screen-editor, which could be used for a number of other applications.

So: please let us have detailed listings of your applications programs, for publication in this Newsletter? Or are there any takers for a software exchange?

Games Software — some reviews

The games under review all originated from Aardvark Technical Services in the States. Aardvark are the biggest(?) software house for Challenger software over there, and several UK dealers stock their tapes. My copies were supplied by Mutek, whose prices are quoted here; but prices do vary from dealer to dealer, so check first before sending your money!

I approached these reviews with two specific points in mind: Christmas, and the not-too-distant event of my kids' school fêtes. I therefore looked for speed, sustained interest, flexibility and, above all, crashproof programming for use by very inexperienced users!

In each case the programs come with a complete listing in BASIC, and while in only one out of the four cases would alteration be essential (*Slashball* — see later), all of them could do with joystick operation (which some of the programs offer as an option, but which I don't happen to have in hardware) and sound and colour if available. My only general gripe is that the instructions displayed for the games are all in upper-case only — I know that <SHIFT-LOCK> has to be locked down for BASIC operation, but upper-case is often almost illegible, especially on an unmodified TV. It's not *that* hard to move that shift-key, is it? Anyway, on with the reviews.

Fighter Pilot

Best described as a 'standard' arcade-type game — you shoot down targets (planes?) by steering your craft so that they fall in line with your sights. But everything seemed a little vague: the sights, as displayed on the screen, were large and, surprisingly, weren't particularly accurate, for example. You missed the target if it was in the upper portion of the sights, but you did hit it if it was just outside the lower part of the sights. Sometimes the target was still left for some time after you'd 'killed' it, too! Some kind of moving background would have helped the feel of the game, but this would probably best be done in a machine-code subroutine for speed.

There are a number of 'levels of difficulty' in this game, but these consist simply of more or less time in which to 'kill' ten targets, which move at random to a greater or lesser degree. There is also an 'automatic fire' option which hits the targets for you when they go into the sights area, and which takes much of the fun out of the game! A good idea, though, if the 'kill' graphics were tidied up a little.

Despite my criticisms, my nine-year-old son liked it the best of the four.

Slashball

We liked this one a lot, despite its serious bug which crashed the program in the middle of several really good games. Infuriating, that! The game was simple, too: the aim was to bounce a 'ball' onto a central target by placing angled barriers or 'slashes' in its path. The catch is that the slashes remain there for the duration of the game once you put them up, and you lose a point each time, too (it's a two-player game). The ball also bounces in different directions according to which way it comes up to it, so we often found ourselves putting up just one wrong slash and then spending several more careful slashes to get back towards the target. Tricky, but fun, especially if you're watching your opponent get into a mess!

The 'levels of difficulty' are again to do with time: if you take too long, the game ends, awarding your opponent an extra twenty-five points as it does so. As games are usually close, this is too much — ten extra would make it a better and fairer finish.

The bug is infuriating, since it wrecks about one in three games; and it is also a serious one, as it crashes BASIC as well as the game. But it should be easy enough to patch, and I hope the word gets back to Aardvark soon. The problem arises from the displayed limits of the board and the way in which the ball is served. The ball always bounces off the edge of the board; the ball is served, at the start of each round, from the baseline of the screen, which can be on or outside the limits of the board. If this happens, the ball is trapped outside the board area rather than within it, and is soon bounced into the memory area above the screen memory — into several of BASIC's flags, including those for the keyboard and screen. The result is a POKEd wide layout for the screen and a locked-out keyboard, calling for a reset and warm-start to BASIC (the program, fortunately, is usually still intact). The solution would be to tighten the limits for the serve so that it can only start within the board area — which should have been done in the first place, but never mind.

Apart from the bug, a very good game — definitely one for the school fête.

Killerbot

I once played a Z-80 machine-code version of this on a Nascom, but if anything this one is faster, even though it's in BASIC! It's an old classic: you have to cross 'a courtyard full of charged guard posts and killer robots'. Run into any of these, or if they run into you, and you lose — 'electrocuted', presumably. Up to ten robots appear from random places round the board or courtyard at the start (your choice) and chase your rather small blip at either half speed ('Regular') or the same speed as you ('Masters'). A winning run at the 'regular' level usually consists of a high speed sprint with tactical stops to let the robots blow themselves up on the guard posts in their haste to get you. 'Masters' is much trickier — you don't have time for tactical stops! And when what you thought was the last robot has blown itself up, leaving a nice clear run, another one appears afresh at the edge of the board — you're always running from at least one.

I could beat ten robots at the 'regulars' level quite quickly, but the 'masters' level is another game again — beating five robots is my best so far, and I'm having trouble doing that well again! Definitely a winner in my family — the only complaint was a demand for a punchball to relieve the built-up tension — and certainly a money-spinner for the school fête. Recommended.

Ten Tank Blitz
Not really for the school fête, this one: more for Christmas, as a good game could go on for half an hour or more. The instructions say 'you won't learn this game in five minutes', which is true (and the typing errors didn't help!); but the complexity is much of the fun. Each player has two sets of controls: one to select one of his/her five tanks, the other to move or fire with the selected tank. Targets can take several hits before being killed, and these values can be easily changed at the program level; the only displayed variable is one which selects the density of trees on the board. This last gives a surprising flexibility to the game, which can range from an open and rapid slam-it-all-out, through variants on skulking through the forest, to a really tricky maze game in which you have to burn your way through a mass of trees without blasting your own tanks and base to pieces in the process. One particularly nice touch is that the missiles each tank launches can be steered: turn the tank after firing a missile, and the missile turns too. (Allows you to fire at others while skulking behind trees — quite apart from encouraging nasty habits, it can also get you 'hoist by your own petard'.)

It's a very clever game with a lot of neat touches which you only learn as you play it. It only just fits into 4K, so there's little room for alteration on a 4K machine. But one thing I would like to change is the 'win' conditions: as supplied, you win only by blasting away at your opponent's 'block-house' until it decides to disappear (after seven hits or so). But in some ways this is too easy: you park a tank in front of it and, while your opponent is desperately pressing keys to find which one of his remaining tanks is close enough to knock yours off, you can easily rattle off the baker's half-dozen shots needed. Limited ammunition, replenished by returns to base in *Star Trek* fashion, would make the game more realistic and, for me, more enjoyable.

Again, definitely recommended. But be warned; a client came to visit me on a recent weekend and, forgetting about his business, played this game for several hours instead! It's as addictive as any version of *Star Trek* that I've seen, and with far more room for maneouvre both in the game and its display, and in its options for programming 'extras'.

Tom Graves

Prices as given by Mutek, Quarry Hill, Box, Wilts:
Slashball: £2.15
Fighter Pilot: £2.60
Killerbot: £2.60
Ten Tank Blitz: £4.35

Hardware mods

This issue's hardware mods were suggested by *Richard Elen.* All are for C2-4Ps.

2MHz operation
On Mod 3 versions of the C2-4P the system clock is derived from the 540 video board's oscillator, and is sent to the 502 CPU board on pin 18 of the bus. The oscillator drives a 74163 (labelled U5D — see sheet 3, top left, of 540 schematics) which puts out approximately 1MHz from its pin 13 (QB). Twice this frequency is available from pin 14 (QA) on the same chip. To change to approximately 2MHz operation, cut the track next to pin 13 of U5D, and jumper to pin 14 instead.

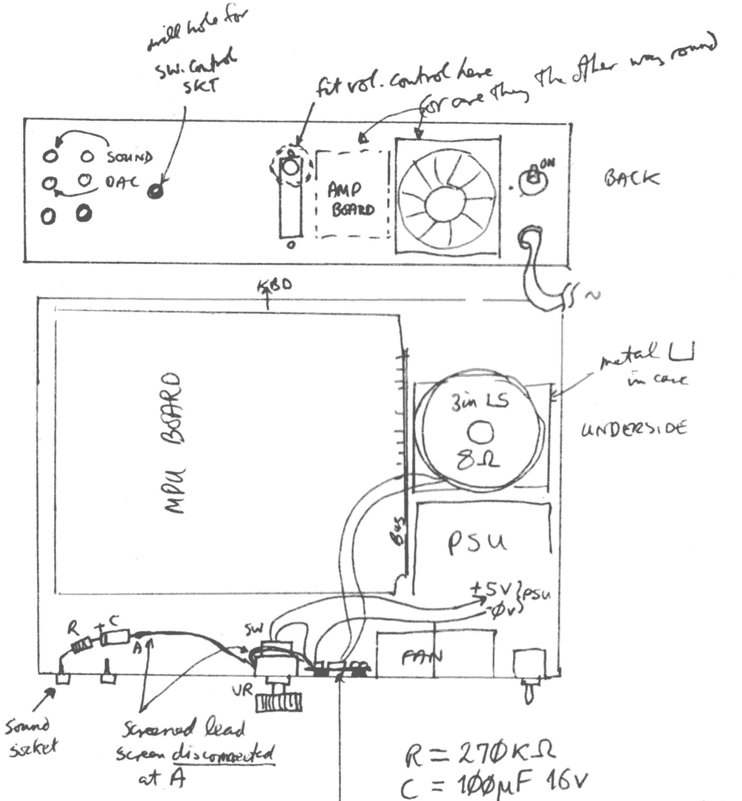
Everything then runs twice as fast as before. But note that while this means that BASIC calculations and operations run twice as fast, so does the keyboard and, in particular, its auto-repeat — it will start its auto-repeat cycle twice as quickly, giving an apparent keyboard bounce if you aren't careful. The clock rate for the cassette interface is supplied by a separate 555 timer chip, and will remain at 300 baud (assuming it hasn't been modified) — so you don't need to worry about your existing tapes becoming incompatible!

The CPU should be up to 2MHz operation, but your memory may not be, since some of the 2114s supplied by OSI are only 550nS rather than 450nS. A memory test is essential to identify chips with any problems — a simple test in BASIC, loading and checking with alternate 1s and 0s, is given here.

```
10  FOR X=900 TO 8192 (or whatever your total memory is)
20  POKE X, 85 (01010101_2)
30  IF PEEK(X)<>85 THEN PRINT X;
40  POKE X, 170 (10101010_2)
50  IF PEEK(X)<>170 THEN PRINT X;
60  NEXT X
```

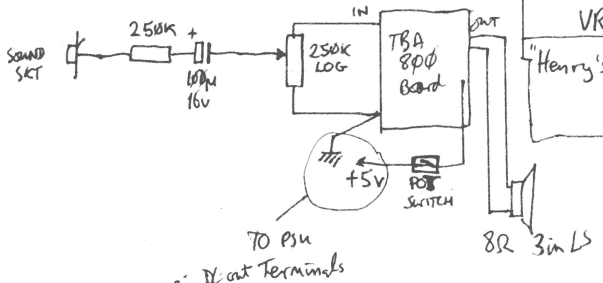
The program was supplied by *Graeme Davies* and takes about 20 seconds per K of memory to be tested. A BASIC program (in principle, at least) can only test memory above itself — a machine-code routine would be needed to test memory below 900_{10} in this case, or else a certain amount of chip-swapping to bring every chip in turn within the scope of the BASIC program.

Internal amp option *(for Mod 3 versions of C2-4P only)*
The Mod 3 versions of the C2-4P have three additional sockets compared to earlier versions. Two of these are for sound and digital-to-analogue signal outputs. Since these are compatible with standard amplifiers, several people have commented that an Apple-style internal amp would make the sound side of the system more compact. *Richard Elen's* diagram below should be largely self explanatory — it uses an amp board from *Henry's Radio* (about £2.70) which outputs about .5W at 5V. We had a very similar design from *Graeme Davies,* using an AI 20 from *Bi-Pak* (about £4.00) connected to the 12V supply across the smoothing capacitor. In Richard's example the amp is wired directly to the sound output socket from the inside, while Graeme suggested that it would be better to have a lead coming out through the back panel with a phono plug to be inserted into either sound or D/A socket.

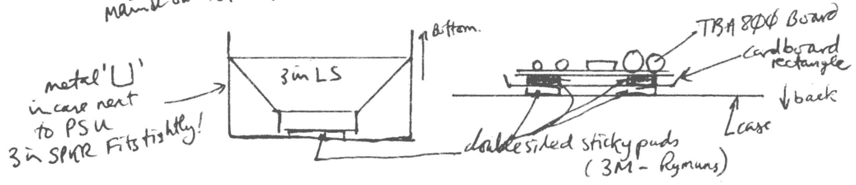


R = 27Ø KΩ
C = 1ØØμF 16V
VR = 25Ø KΩ Log with Switch

"Henry's" amp board (£2:7Ø)
uses TBA 8ØØ IC - ≈.5 W@5V

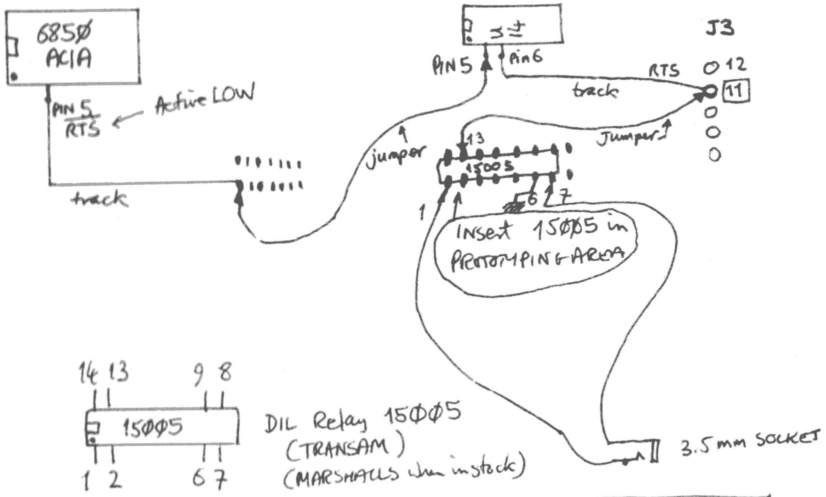


RE - 26 x 79

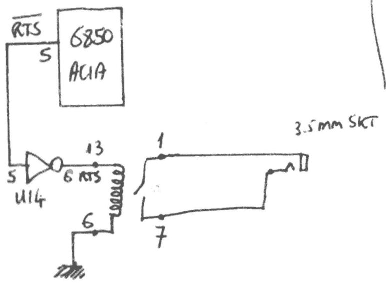


Remote switching option

This uses a 15005 DIL relay driven from the RTS (Ready To Send) line from the ACIA (pin 5 of the 6850 chip) on the 502 or 600 board (Superboard). The addresses for the ACIA on the boards are *FC00* and *F000* respectively. The version for the C2's 502 board is shown here. Note that in most versions of the C2 the RTS line is *active low*, and will need to be inverted — hence the connection to a spare circuit of U14 between the ACIA and relay.



C2-4P
POKE 64512, [$FCØØ]
81 ON
17 OFF



SUPERBOARD:
connect to same principle:
POKE 6144Ø, 81 ON
, 17 OFF
[$FØØØ]

RE - 26x79

Dealer notes

An assortment of notes on OSI dealers: who and where they are, what they sell, and what (if any) special services or facilities they offer.

Those listed in detail below are the ones I have been able to contact before the copy date for this issue — some seem impossible to contact by phone, at any rate. They are not given in any particular order — the order is that in which they occur in my rather erratic notes.

Intelligent Artefacts, Cambridge Road, Orwell, nr Royston, Herts.
Arrington (022 020) 689.

Hardware: C1/Superboard series in stock; full OSI small systems range to order.
Software: mainly disc software for OS 65D.
Specialities: debugging OSI disc systems! Associated company is a MAPCON consultant specialising in process control.

Watford Electronics, 33/35 Cardiff Road, Watford, Herts.
Watford (0923) 40588/9.

Hardware: Superboards only at present: 7 days delivery quoted.
Software: standard OSI and Aardvark software, for Superboards only.

Videotime Products, 56 Queens Road, Basingstoke, Hants.
Basingstoke (0256) 56417 & 26602.

As for *Watford Electronics.*

Calderbrook Technical Services *(CTS)*, 1 Higher Calderbrook, Littleborough, Lancs OL15 9NL. Littleborough (0706) 79332 (any time).

Hardware: Superboard/C1 and C2-4 from stock; full range to order. OSI expansion boards; own C1-to-OSI-bus interface board; other own boards under development.
Software: standard OSI and Aardvark software. Own extended monitor for Superboard/C1 (supplied with Superboards from CTS) includes screen clear, screen editing facilities. Some own software under development.
Specialities: full technical back-up: 'emphasis on service'.

Mutek, The Studio, Quarry Hill, Box, Wilts.
Box (0225) 743289.

Hardware: all C1, C2, C3 ranges normally from stock (Superboard *not* stocked). Memory boards and some others from stock; all OSI expansion boards/systems (including hard discs) to order — ask for delivery dates. RS-232 interface, 600 baud conversion for C2 series cassette interface; colour mods and joysticks for C2 Mod 3 series available shortly.
Software: Aardvark tapes for C1 and C2 series; new British games tapes and utilities available and under development (publishes new software on a royalty basis).
Specialities: full technical and programming back-up, particularly for non-standard applications.

Newbear, 40 Bartholomew Street, Newbury, Berks.
Newbury (0635) 30505.

Hardware: Superboard from stock; C1 series from stock early in '80; C2 and C3 series available from stock later in '80.
Software: standard OSI and Aardvark tapes.

Other dealers we know of are:

Microcomputer Business Machines, 4 Morgan Street, London E3 5AB. 01-981 3993.
Mainly aimed at business market.

Lotus Sound, 4 Morgan Street, London E3 5AB. 01-981 3993.
Superboards and related hardware/software only.

Cavern Electronics, 94 Stratford Road, Woolverton, Milton Keynes MK12 5LU.
C1 series only at present.

N.I.C. Models, 27 Sidney Road, London N22 4LT. 01-889 9736.
Superboard and UK101 advertised.

U-Microcomputers, Station Road, Weaverham, nr Northwich, Cheshire.
0606 853390.
Full OSI small systems range, with software from a variety of sources (catalogues available).

JEM Computing, 222 Pensby Road, Heswall, Merseyside.
Software only: games tapes for Superboard/C1 series.

No doubt I have missed out some dealers from this list — let us know who you are! And let us know in detail what you're selling and doing, so we can include the details in this section: your stock and specialities are what we want to know about.

Group Notes

Organisation and queries

At present the Group does not have much of an organisation — and in many ways it would not be right if it did. We have rough areas of specialism, but that is all. Richard Elen and George Chkiantz will be handling subscriptions and membership enquiries; Richard and George and, in the North, Graeme Davies, will handle technical queries and the like, as 'a forum for the exchange of practical ideas' etcetera; while I (Tom Graves, in case you weren't sure) seem to be landed with the Newsletter's editing, setting and production. You'll find our various addresses after the intro to this issue.

Independence and all that

We intend that the Group should be fully independent of OSI or any of its dealers. So in case you might have got the wrong idea, this is *not* meant to be a Mutek newsletter! I am aware of the number of times that their name has turned up in this issue, but in some ways that has been inevitable. Mutek's Dave Graham in particular gave us an enormous amount of time and help, contacting the right people to start the Group, and then supplying us with the essential material to get started. They also arranged the deal we've made for serialising Aardvark's BASIC Notes — almost the only material available on OSI's version of Microsoft's BASIC. (I happen to have a soft spot for Mutek for another reason — they risked their survival in order to break the absurd former price structure on OSI kit. The results can be seen in every current ad for OSI kit — realistic prices help everybody, dealers included.)

We feel that it's right to give a mention to people who help us, whether dealers or otherwise. And when any dealer — Mutek included — makes a blunder, they will also get a mention of a rather different kind in this Newsletter...

Quite simply, the Group would not be of much use unless it *is* independent, and that is how we intend it to be.

Subscriptions and prices

For this year at least, the annual subscription to the Group is £5.00, which includes four issues of this Newsletter and, we hope, quite a bit more. We cannot really make the Newsletter more frequent than quarterly without going at least partly professional, something we cannot afford as yet in several senses. Increased frequency means far more work, far more copy (or smaller content per issue) and far greater postal bills. To fill the gaps between the issues, you should soon be seeing a regular column of 'Professor Challenger's Notebook' in *Practical Computing*, compiled by Richard Elen from whatever you care to send in.

Five pounds is the same as the price of four issues of the Newsletter on its own. So you could buy it from a dealer if you prefer. But we'd rather you subscribed and became a member, for a number of reasons.

First, you'll get your issues more reliably, direct by post. We have to base our print run for the Newsletter on the number of member/subscribers we have, with only a small number left over for reserve and for sale through dealers. To put it simply, if you don't subscribe, there may not be a copy at your local dealers when you wander round there to get it — they'll all have been sold.

Second, we can use your money to give you more for your money. If you buy a copy from a dealer, he will keep the standard mark-up (about a third of the price): effectively, we lose rather more than that, because we have to post a batch off to the dealer, and spend time on admin chasing the dealers for our money too!

There is a definite break-even point, as far as the number of subscribers is concerned, at which the Group can make a profit. At the moment, that would be around the two hundred mark. Since we're not in it for the money, that profit can used for other things, a little extra money to give members a little more for their money. Exactly what that extra would be is up to you. Decent documentation would be one place to start — free to members, but for sale to others. Your suggestions, please!

We can arrange other things for members as a group, too — we've already started on some things, as described below. But to do this, we need to know the size of our membership, and to have a reasonable-sized membership, so as to guage quantities when negitiating deals for hardware, software and the like.

And finally on costs, our other big expense will be postage. So please, when writing to us, don't forget to enclose stamps or, preferably, a stamped addressed envelope for reply. If you forget, you'll be eating into that 'spare cash for extras', which is supposed to be for everyone.

Negotiations

Various things are already under way. As I mentioned earlier, you should soon be seeing a regular column of 'Professor Challenger's Notebook' along with 'Pet Corner' and 'Apple Pie' in *Practical Computing*. We're also working out the form of a standard contract for software publishing, based on book-publishing contracts, to give amateur software authors some degree of protection. Our current negotiations are, again, with Mutek, but we hope the form will be adopted by other dealers as well. In any case, it should give the software writers among our members a very good royalty deal (three or four times better than book publishing, because of smaller origination costs for the publisher) and a good market for their work.

What we haven't yet started on is hardware negotiations, either for marketing of members' modifications or board designs, or for bulk buying of hardware such as the 527 memory board. Suggestions, please?

Aardvark's BASIC Notes

For those of you in a hurry, the complete set of Aardvark's notes on BASIC should be available from most of their stockists — we know that CTS and Mutek at least have them in stock, but try your local dealer. No-one seems sure of the right price for them! Aardvark's own price for its September version was $9.95, to give you — and the dealers — some idea. Aardvark are currently updating and improving them, though, which may lead to an increase in price. The full set includes full descriptions of BASIC's method of storing programs, variables and arrays, the main program execution loop, locations and rough descriptions of most of the subroutines in BASIC, and many important ideas and comments on matters like recovery from accidental cold-start. Essential for OSI users, particularly those doing mixed BASIC/machine-code programming.

Writing for the Newsletter

This is *your* Newsletter.It is no doubt obvious that rather too much of this issue has trickled from my pen; but that's not surprising in a first issue of a new magazine, and I don't want to do it again!

So tell us what *you* are doing. Tell us in writing; tell us in pictures, drawn or photographic; tell us in programs, as listings or on tape.

If you're writing, please *type*, at least doubled-spaced lines, and preferably on A4 paper to simplify our filing later. We can decipher most manuscript, but in this field one error means the difference between something that runs and something that demands a few hours of bug-hunting. And people involved in computing have an infuriating habit of hand-writing everything in capitals... think of the poor typesetter who has to decode it! As for writing style, perhaps the best guide I've seen is in *Micro* 16 (September '79), which should still be available from most computer stores. As Editor, I reserve the right to prune excessive jargon — beware!

Illustrations make all the difference between a grey splodge of text and an interesting page. We've been a little short this issue, and the drawings we have included have not been too clear, but we rather ran out of time for re-drawing to a more presentable standard. Illustrations cost no extra to print in the process we use, but they do need to be good — a bad picture is worth a thousand garbled words, which we can well do without. We don't really want to publish endless photos of OSI kit, but interesting applications are worth showing *if* the photograph is clear and with an uncluttered background. Sometimes it's more interesting to draw from a photo, as we'll be doing with the front cover of the next issue. In short, we'd really like to use a lot of illustrative material, since it costs little extra; but it must be interesting to be worth putting in!

Programs, particularly for applications or as usable sub-routines, are going to be a major part of the Newsletter. As *Micro* put it, a hand-written program is one that probably hasn't been tested on a machine: we tend to be a little doubtful of them. We haven't much spare time to load and test programs from written listings: printer listings would be better, but best of all would be listings on tape in one of the standard OSI formats (300 baud, BASIC, machine code or assembler). At the moment the VDU for my Challenger is beside my typesetter, so I can set direct from screen to screen; but veryshortly we should be able to set direct from program tapes. We'd like to be able to take word-processor outputs, but technical problems — not to mention union problems — limit that at the moment, though we should have that side going within the next few months.

One problem is copyright. Unlike *Micro*, we are not a commercial organisation: we're not interested in sole commercial rights. The copyright on any material would remain yours. But we would like to retain the right to republish content from the Newsletter in book-form at a later date — we thus state that under that description the copyright of material published in this Newsletter will be held jointly by both parties, ourselves and you. If you object to a joint copyright of this kind, please say so and we will publish it with a copyright notice in your name only. We want to know what you're doing, but we don't want to cause any commercial problems for anyone.

In any case, get in touch with us — we'd all like to know what's being done with OSI kit.

Next issue

Copy for the next issue should be sent to me by the end of the second week in February '80. That allows us enough time to prepare the issue for publication in mid-March.

The next part of our serialisation of Aardvark's notes will be on BASIC's storage formats for programs, variables and arrays. We'll also have a memory map in as detailed a form as we can make it, for both C1s and C2s.

Our main theme for the next issue will be graphics and screen handling, as a start towards filling at least one of the holes in OSI's documentation. We'll be looking at graphics applications of Challengers. In BASIC we'll have 'print-at' and non-scrolling input and print routines, among others. For USR and machine-code buffs we'll explore alternative uses for some of the screen-handling routines in the ROM BASIC: *BFDE*, for example, can be used to control a non-destructive (but non-transparent) cursor, or as a true rubout.

And there'll be plenty of notes, ideas and programs. See you again in mid-March!