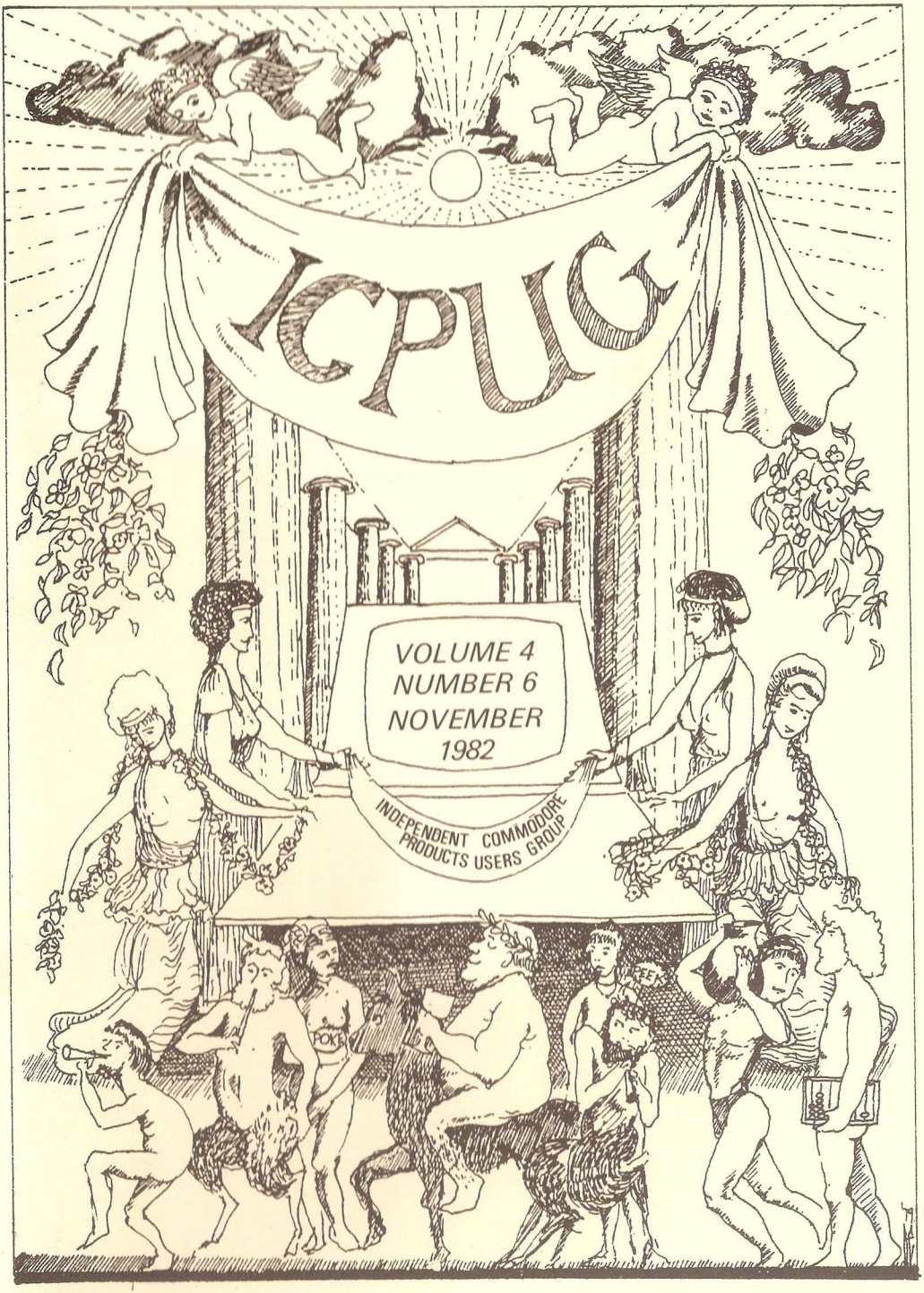

ICPUG
VOLUME 4
NUMBER 6
NOVEMBER
1982
INDEPENDENT COMMODORE
PRODUCTS USERS GROUP
POKE
PEEK

HONORARY NATIONAL OFFICIALS

Chairman:
Wing Cdr. Mick Ryan
164 Chesterfield Drive,
Riverhead,
Sevenoaks, Kent TN13 2EH
Telephone: Sevenoaks (0732) 453530

Technical Queries Secretary:
Jim Tierney
11 Collison Place,
Tenterden,
Kent TN30 7BU
Telephone: 058-06 2711

Regional Co-ordinator:
Terry Devereux
32 Windmill Lane,
Southall,
Middlesex UB2 4ND

Treasurer:
Joseph Gabbott

Software Librarian:
Bob Wood
13 Bowland Crescent,
Ward Green,
Barnsley, South Yorks S70 5JP
Telephone: (0246) 811585 (work)
(0226) 85084 (home)

Membership Secretary:
Jack Cohen
30 Brancaster Road,
Newbury Park,
Ilford, Essex IG2 7EP
Telephone: 01-597 1229

Editor:
Ron Geere
109 York Road,
Farnborough,
Hants GU14 6NQ

VIC Co-ordinator:
Mike Todd
27 Nursery Gardens,
Lodgefield,
Welwyn Garden City,
Herts AL7 1SF

Discounts Officer:
John Bickerstaff
48 Martin Down Road,
Whitstable,
Kent CT5 4PR
Telephone: (0227) 272702

Assistant Editor:
Tom Cranstoun

INDEPENDENT COMMODORE PRODUCTS USERS GROUP

Vol 4 No. 6 **Newsletter** Nov 1982

Europe's first independent magazine for PET users

Page	Contents

EDITORS NOTEBOOK

Each issue of the Newsletter seems to be larger than the previous and I am led to wonder how much further this growth can be sustained. Each revision of production methods improves 'productivity', but each breathing space so created is soon plugged by the expansion. At the A.G.M. there was the customary two minutes silence when nominations for editor were sought, and so here I am again for another 12 months. Several members were kind enough to offer to assist and this will ease the load. In addition a number of background projects which have laid dormant could now well see fruition in the new year. Tom Cranstoun has volunteered to take on the task of assistant editor to succeed Mike Todd who is almost 100% committed to the 'Vic' range product information. Suggestions as to how you can help are outlined on p372.

This issue marks the end of volume 4 (can it really be over four years since Norman Fox founded the Group and I became editor ?) and next issue there will be a number of changes to the Newsletter, a new cover design perhaps, and other improvements that as yet I cannot disclose.

Meanwhile please see that correspondence is directed to the appropriate person. I only handle Newsletter input. For output, i.e. mailing queries and back-numbers (at £ 1.00 each), contact Jack Cohen. Comal matters are dealt with by Brian Grainger, Vic series and CBM 64 by Mike Todd, but with those exceptions, all input should be addressed to the editor.

This issue was nearly 30pp shorter. When Mike Todd's disk of text arrived, the Post Office had neatly folded it in half. Fortunately I was able to soften it in warm coffee and iron it flat (nylon setting) again and with a bit of WD-40 it was soon rotating in my drive unit....

R.D.G.

--oOo--

MIXING IT - MACHINE CODE WITHIN BASIC

Often one needs to add a short machine-code routine to a BASIC program, either for speed, or to do some function that is impracticable in BASIC. There are several methods, and each has limitations and disadvantages. I will not attempt a discussion of the relative merits of placing the code a) in high memory, b) in the second cassette buffer, c) at the end of the BASIC program and d) at the start of the program. This brief note is concerned with method d) and one method of placing the code there.

If an assembler is used to produce the code, such as Supersoft's MIKRO, the following example shows how to create code in a REM statement in line 0 of the program.

```
100                     ;example to put m/code routine
120                     ;in rem at line 0
140                     *=1024
160  0400  000d04   start       byt  0,<link,>link,0,0,$8f
180  0406  a902     sysrtn      lda  #2 ;example routine
200  0408  8d0880                sta  $8008
220  040b  60                    rts
240  040c  00                    byt  0
260  040d  0000     link         byt  0,0
```

This will create the necessary line 0 to which the rest of BASIC may be added and edited without ill-effect. This method, like all others, has disadvantages. In this instance the code is limited to about 250 bytes and the code may not contain any zero values, eg BRK or LDA #0.

R.D.G.

--o0o--

- from an I.E.E. circular.....

PRESTEL ON YOUR PET

By David Annal.

It is now possible to couple your PET, any model, to an RS232 Tantel adaptor, discussed in last issue, and use it to call up Prestel pages, view them on the PET screen, save them on disk and, what is particularly exciting, download computer programs and convert them to BASIC. Once converted, they can then be saved or run in the usual way. Programs are already available together with club information and news -see ICPUG pages starting on 80061819.

UART Board.

The means whereby all this can be achieved is the UART coupling board and associated software produced by Y2 Computing Ltd. UART stands for Universal Asynchronous Receiver-Transmitter (which can now be forgotten) and its function is to convert parallel to serial signals and vice versa. This particular chip is mounted on a small circuit board, together with a few associated components. It is supplied to plug into the middle (UD4) socket of your PET. If you have already got a utility chip in this socket, it is still possible to use it as the board has a duplicate socket included. Simply remove the IC already there and plug it into the socket on the board. This will work for all 2K Chips, such as Toolkit, etc. The board can be supplied to plug into other sockets if required. Flying leads are connected from the board to other points of the PET circuit board via shakeproof prods. A lead from the board comes out from the side of the PET to connect into the Tantel via a DIN plug. All these connections and installation instructions are described in a booklet accompanying the board. They are full and detailed and include easy to follow diagrams. The whole installation takes only a few minutes.

What you need.

The Telesoftware Tantel Adaptor is a MODEM device (MOdulator/DEModulator) which will plug into a Prestel jack provided by British Telecom next to your 'phone. A colour

TV set will plug directly into the adaptor via its ordinary aerial socket. These two will now function as a Prestel terminal and the phone nos. of the nearest Prestel computers can be programmed into the adaptor. The Prestel computer can now be dialled automatically using the keys on the adaptor, and your requested pages will be shown on the TV in colour. (Various registration procedures have to be followed in the first instance. They are not discussed here but are adequately covered in the papers accompanying the Tantel). The phone line can be acquired or disconnected by pressing appropriate keys. Other makes of modem are likely to appear before long and some already exist, although more expensive - see also, article by Bob Denton, p274, last issue.

Now, although the above set-up is satisfactory for viewing pages of your choice and for answering back in a simple way, such as revealing hidden answers, etc., it cannot make use of other superb facilities now being provided on Prestel. These include the sending of messages to other users and the general use of the alpha-numeric keys in answering questions, placing orders, etc. A Tantel is available with keys, but why not make use of those you already have on your computer and use the Y2 board ? Several other advantages follow which we will discuss below.

Character Set.

First, it is necessary to deal with the display which appears on your computer screen. Unlike the TV set, this will obviously be in monochrome. This need not be a disadvantage as any colour TV set connected to the aerial socket will still show the display in full colour even though the PET has taken control. The TV is not now essential and could be removed back to the heart of the family and thus avoid arguments! The PET screen will continue to function as the display, but there is a snag..... Prestel graphics are not the same as PET graphics and a full display of some Prestel characters is not possible. This may not matter much if you are only receiving the printed page, but graphic pictures will not look right. Y2 will provide a chip, at extra cost, which

replaces the character generator in your PET. In the case of the 'fat 40' and 8032, this provides the full Prestel set and all graphics will reproduce correctly on the screen. On 9" screens, the chip will not interfere with normal PET graphics used in upper case mode as these are still there. However, in lower case mode, used by Prestel, the graphics associated with shifting the top row keys and numeric keys now become the Prestel set. Fractions and pound signs also appear! For most applications, the Prestel character ROM can remain in situ all the time.

What will it do?

Once the computer had been plugged into the RS232 socket of the Tantel, all control is possible from the computer keyboard and the Tantel keys are not required, although they will still function in an emergency. In order to achieve this transformation, the first program on the software disk supplied with the UART board is loaded in the manner appropriate to your model.

Off-line mode.

When first loaded, the program enters the off-line mode and displays a menu which gives the user the choice between a display showing all the normally hidden Prestel control characters and a normal screen. The former would only be of interest to those editing their own pages. You are next asked if you wish to display PET characters or Prestel ones. The answer depends on whether or not the Prestel character ROM has been fitted as described above. In the off-line mode, facilities are provided to recall a previously recorded page from disk, alter, or add to, the pre-programmed telephone numbers of Prestel computers, change the modes above, exit the program altogether, or enter the on-line mode. (Note that phone numbers are stored in the Tantel, which has a battery backup. They will not therefore be lost when the computer, or Tantel, are switched off).

On-line mode.

This allows you to call up the Prestel computer of your choice, using the numbers programmed above. You may then: view pages at will, copy any interesting pages to disk, write letters to friends, answer advertisements, book tickets and generally make use of all the Prestel facilities normally available. The ability to store pages on disk means that you are connected to the phone line for a shorter time and can view your selections at leisure when off-line and not clocking up phone bills! Normal extra facilities are readily available such as 'reveal', 'double height', etc, but will only be seen on a TV connected to the Tantel.

Telesoftware.

The great advantage of using a computer connected in this way is that programs, especially provided for the purpose, can be downloaded directly into your machine, saved on disk and then converted to BASIC to be stored or run at will when off-line. The procedures to enable this to be done are fully detailed in the instructions and are accomplished fairly simply. Two steps are required. Once the telesoftware program page has been located in the usual way, it is accessed by the control program which then saves it on disk as an ASCII file. Full error checking is implemented and if this is satisfactory, the program returns to the normal on-line mode. After any further programs have been saved, and at the end of viewing, the off-line mode is entered and the whole program exited. A utility program, also supplied on the same control disk, is now loaded and by this means, the previously saved ASCII file is converted to PET BASIC which can be re-stored for future use and run in the usual way.

Summary.

A nice little board giving several extremely useful extra facilities for use with a combination of PET, Tantel adaptor and Prestel. Installation is simple and easy to follow diagrams make it hard to go wrong. Operation is straightfoward and the programs do what they are supposed to do. Documentation is adequate. Y2 will support the board

and software and models are available for different sockets. Models will also be available for different modems as they are produced.

Contact - Y2 Computing, 5, Kenilworth Court, Watford, Herts (mail only) or phone Watford 50161. Mailbox no. on Prestel - 092350161. The board and adaptor is also available from Prestel itself at Telephone House, Temple Avenue, London EC4Y OHL.

--oOo--

COMAL CORNER

By Brian Grainger.

There has not been much COMAL news in the last two months, no doubt the summer lull. The news in the last Newsletter that I had versions for the 8096 SuperPET produced a larger than expected response and it is heartening to note that at least one business user has started serious work on business programs using COMAL. A software house has also considered the use of COMAL for its work.

I mentioned last time that the COMAL board was available from Ellis Horwood. I now have the price, 195 pounds ex.VAT. Not very cheap but it does give the serious non 8096 owner nearly 31K user space on a 32K PET. It is understood that a firmware implementation is being worked on for the Commodore 64. Few details are available yet but there will be a price tag attached. Talking of the Commodore 64 leads me into mentioning that Commodore will be marketing an extension to the BASIC operating system. Written by Dave Simons this has some sytructured BASIC extensions but they do not appear to compare with the facilities of COMAL although it IS a step forward.

From a business user I heard that a sort routine based on QUICKSORT had been implemented by him in COMAL. On an 8096, random 10-character strings were set up and 800 of them could be sorted in 2.5mins. Apart from the fact the user could not set up such a sort routine in BASIC he found

that it was much faster than what he could set up in BASIC. Fast enough to make the program suitable for use.

The latest versions of COMAL are 0.12 for 4032/8032 and 1.02 for 8096. Programs are NOT load and save compatible between the two versions. For example the token for OPEN FILE in 0.12 turns out to be INTERRUPT in 1.02. Other inconsistencies exist so those users who are trying to use 0.12 programs on 1.02 must LIST the programs from 0.12 and ENTER them to 1.02.

I have spoken to the distributors of Len Lindsay's 'COMAL Handbook' (Prentice Hall) who tell me that it will be available in the UK in December at a cost of £15.15

The most amusing bug in version 0.11 has been found. If one uses the TAN function it calculates the reciprocal of the true answer. This is true for the original version 0.11 COMAL as well as the versions for BASIC2 and cassette users.

To finish this time here is a Shellsort routine from Nick Higham. This sorts N items of the array A().

```
1320 PROC SHELLSORT(N,REF ())
1330  D:=INT((N+1)/3)
1340  REPEAT
1350   FOR I:=1 TO D DO
1360    FOR J:=1 TO N-D STEP D DO
1370     TEMP:=A(J+D)
1380     FOR K:=J TO 1 STEP -D DO
1400      IF TEMP>=A(K) THEN GOTO LABEL8
1410      A(K+D):=A(K)
1430     NEXT K
1440 LABEL8:
1450     A(K+D):=TEMP
1470    NEXT J
1480   NEXT I
1490   D:=INT((D+1)/3)
1500  UNTIL D=0
1510 ENDPROC SHELLSORT
```

--oOo--

VIC MATTERS

by Mike Todd

Before I go any further, who spotted the mistake in the reconfiguration program in the last issue? I omitted the T$ from line 1030 which would cause an error when the program was run. Line 1030 on page 229 should read:

```
1030 : READ X,Y,Z,T$
```

There was also an error due to a bug in the Vic. This is to do with the "FILE NOT FOUND ERROR" that you should get if you attempt to OPEN a file for reading beyond the End-of-Tape marker. Instead you get a "DEVICE NOT PRESENT" error which could be a bit confusing! If you attempt to LOAD beyond this End-of-Tape marker, you WILL get the "FILE NOT FOUND ERROR" as the bug only occurs in the OPEN routine.

I discovered to my horror that part of the SUPER-EXPANDER wouldn't work when the Vic was reconfigured to have +0k RAM. This is probably because the reconfiguration to a Vic with no expansion RAM but with the screen moved around as if it were, is an unlikely configuration. I would therefore suggest that this configuration only be set up for experimenting with and not used for any serious applications. I don't know why the expander gets upset at this configuration and I will try to look inside for a clue.

ICPUG & VICSOFT

Up until now, we have not put much effort into recruiting - most have joined as a result of our stands at the exhibitions or through personal contact. The next issue of VICSOFT, which many of you will be receiving soon, contains details of the Group and this should have a significant effect on our membership in 1983.

On the subject of VICSOFT I would like to clear up any confusion about VICSOFT being a user group like ourselves. It is Commodore's intention to have VICSOFT as a "discount club"

and as such it will not be running in direct competition with us. Instead, it is intended that both groups would work closely together, with the overriding consideration that ICPUG is INDEPENDENT. In fact, with all our very close contacts with Commodore, this independence is continuously stressed, and the encouragement we receive from Commodore (which I might add extends from the very top of the Company) has always been with their conviction that our independence is of paramount importance.

MEMORY MAPS AGAIN

I know that I've spelt out the organisation of the Vic's RAM on more than one occasion, but judging by my mailbag there is still some confusion about the use of memory expansion so here, I hope, is the final and definitive description of the Vic's memory layout.

First some terminology. The Vic is theoretically capable of having 65536 storage locations (BYTES). Each is ADDRESSED by its number from 0 to 65535 and is capable of holding 8 BITS, that is a binary number with eight Binary digITS (hence "BITS") from 00000000 to 11111111 or 0 to 255 in decimal.

Some of these locations are allocated when you buy the Vic and you can see a "map" of these in the accompanying diagram. Locations 0-1023 and 4096-8191 already have Random Access Memory (RAM) installed which means that these locations can be used to store numbers and then have these values read back as required, although any numbers stored will be lost after the power is removed.

On the other hand, locations 49152-65535 contain Read Only Memory (ROM) which has had all its locations permanently preprogrammed so that, although you can't put a number into these locations, you can get numbers back again. These numbers represent the instructions to the microprocessor at the heart of the Vic which tell it how to interpret the BASIC language program that you type in.

A second section of ROM from 32768 to 36863 contains details of the shape of all the characters you can put on the screen and is called the character generator. There is also a small section of memory space from location 36864 which is used to access the special chips used to control the screen, keyboard, cassette machine and so on.

You will see from the diagram that the numbers stored in locations 0-1023 are called "system variables". This means that these locations are used by the Vic to keep a track of the different functions of the Vic itself such as where the BASIC program is stored, or where the variables used in the BASIC program are stored. These locations are not of much use to BASIC programmers, although PEEKs and POKEs (which allow you to actually see what numbers are stored in each location or to change them) can sometimes be used to get at these locations to good effect. For instance, the location 650 controls which keys will repeat when held down long enough and normally contains the number 0 but if you change it to 64 (by POKE 650,64) then no keys will repeat or if you change it to 128 (by POKE 650,128) then all keys will repeat.

RAM from 4096 to 7679 is used to store BASIC programs when you type or LOAD them in, and the section from 7680 to 8191 is used to store the characters on the screen and these locations are not available to store BASIC programs.

Because of the way that the video chip (the VIC chip itself) accesses memory to get at the section which holds the screen information or the character generator, these areas of memory must reside within the Vic itself and cannot be contained in expansion memory outside the Vic. This may seem an odd thing to say at this stage, but it is important to appreciate this when trying to understand why the Vic's memory swaps around under certain circumstances.

The simplest type of expansion is to fill in the gap between 1024 and 4095 and this is what the normal 3K expansion modules do. When you plug this in and then switch on the Vic, the space available for storing your BASIC

programs now starts at 1024 instead of 4096 allowing you to write bigger (and better) programs.

Although the screen memory remains in the same place, it is possible to tell the VIC chip that the character generator has moved into the section of RAM from 4096 to 7679 and, since this is RAM and not ROM, it would be possible to change the characters on the screen and make up your own. You don't need to have the 3K expansion to do this, but without it there would be very little room for BASIC programs.

The other type of RAM expansion available is in multiples of 8K, although some companies do sell it in 4K or even 2K multiples.

This memory could be addressed anywhere between 8192 and 32767, the precise position is determined by the memory module itself and is often set by means of small switches or wired links on the circuit board. If you have an expansion unit such as the Arfon or Commodore unit with several slots, it makes no difference which slots the modules are plugged into since it is the module itself which determines where it should appear in the memory map.

To be usable by BASIC, this memory must be contiguous (that's a computer word for adjoining!) from 8192 onwards and so your first 8K expansion must reside from 8192 to 16383, the second from 16384 to 24575 and the third from 24576 to 32767. These sections are often referred to in 8K blocks as block 1, block 2 and block 3 respectively.

As soon as the Vic sees contiguous RAM after location 8191 it makes the decision to move the screen memory from 7680-8191 to 4096-4607. It does this to ensure that the largest available contiguous block of RAM is available for BASIC programs - from 4608 through 8191 and into the expansion RAM.

Although theoretically possible with the 3K expansion fitted, to move the screen down to 1024-1535 giving even more

RAM for BASIC programs, this is not possible in practice since, as already mentioned, the video chip cannot get at memory contained within expansion RAM. As a result, although the 3K may be installed, as soon as RAM is installed above 8191, it will be ignored as far as BASIC is concerned, although it is still there and can be used for machine code or accessed by PEEKing and POKEing.

There is a special section of memory space between 40960 and 49151 (block 5) reserved for plug-in ROM packs which normally contain games programs. These can be made to start up automatically as soon as the Vic is switched on. Although not normally used for RAM, it is possible to buy "cartridge simulators" which allow RAM to be inserted in this space. This has the advantage that you can leave the module in place and simply load programs from disk or cassette as required instead of swapping cartridges - cassettes are a bit tricky to use up at this end of memory, but it can be done.

These cartridge simulators usually have a built-in battery to keep power applied to the RAM even when the Vic is turned off and thereby preserving the contents of the memory for the next time you turn the Vic on - almost like a ROM.

Finally, because the Vic must see only one byte of RAM at any given address, it is not possible to have two or more cartridges with the same addresses installed at the same time. Some expansion boards have switches to disable unused cartridges, and this effectively removes them from the memory map - otherwise it is important to remove unused cartridges. This occurs most obviously when using several different games cartridges as they normally occupy block 5 (40960-49151). However, some cartridges (such as Commodore's Programmer's Aid and Machine Code Monitor cartridges) use block 3 (24576-32767) and therefore any 8K RAM in this section must be removed to allow these cartridges to work correctly.

A GUIDED TOUR

As promised, here's a quick guided tour of a few of the memory locations listed on pages 188-198 of the July Newsletter. I can't possibly describe all the locations in detail, so I'll only look at the more interesting ones. Many are in constant use by the Vic and their contents when PEEKed or POKEd may bear no relationship to their actual value when the interpreter uses them. Typical is the floating point accumulator (FAC at $61-$65/97-101) the contents of which will change as soon as the Vic has to evaluate a number such as the parameters in the PEEK or POKE commands themselves.

In order to get at some of the pointers and so on, it is useful to define a function to get a 16 bit number from RAM as follows: DEF FN DEEK(X) = PEEK(X)+256*PEEK(X+1)

$00-$02 000-002 USRPOK

This is a JMP to the programmer's own machine code routine and is executed as soon as the USR(X) function is encountered within an expression. The floating point value of X is available in FAC (the floating point accumulator at $61-$65/97-101), and, after processing, the value of FAC will be returned as the value of the USR(X) function.

If A=10+USR(5) were executed, and a machine code program was written which doubled the value of FAC, then A would be evaluated to 10+(5+5). Note that the start address of the machine code routine would be placed in $01/$02 and the routine would end with an RTS instruction.

$03-$06 003-006 FACINT and INTFAC

FAC can be converted to an integer or vice versa by a JMP ($0003) or JMP ($0005). This is only of use to a machine code program and could be useful when using the USR function.

These two vectors are never actually accessed by the Vic itself once set up.

$013 019 CHANNL

This location is normally 0, but is set to the logical file number during INPUT#, PRINT#, GET# and CMD and is restored to zero following them all except CMD. If it is zero then input and output is from the keyboard and screen in the normal manner; if non-zero then prompts are suppressed (to avoid them appearing on an active output device) and the normal INPUT and PRINT routines (which are shared by the file input and output routines) will behave accordingly.

$2B-$2C 043-044 TXTTAB

This pointer (the first byte of which is the LEAST significant byte in normal 6502 tradition) indicates the first byte of the BASIC program in RAM. Its value can be accessed by FN DEEK(43) and this would normally be 4097 for an unexpanded Vic, 1025 for a Vic with only 3K expansion and 4609 for a Vic with +8k or more expansion.

$2D-$2E 045-046 VARTAB

Points to the start of the space in RAM where the identity and value of ordinary variables (that is anything other than arrays), are stored. This normally starts at the byte following the end of the BASIC program and is also used to indicate the last byte to be saved to cassette or disk.

$2F-$30 047-048 ARYTAB

Points to the first byte after the ordinary variables and indicates the start of the space used to store arrays.

As soon as an ordinary variable is used for the first time, this pointer must be increased to make space for it, and so all the arrays must be moved up too. This can take some time, and it is therefore best to DIM arrays after all ordinary variables have been used at least once - if necessary, dummy statements like A=0:B=0 and so on could be used. Of course this is only of use if speed is important.

$31-$32 049-050 STREND

This indicates the first byte after the array storage space and is considered the last byte of the memory being used. However, strings occupy RAM, but working from the highest available RAM location downward - so this pointer is actually the lowest location available for strings, or the end of string space pointer. If strings are in danger of going below this location, a garbage collection is initiated (see page 185 of July Newsletter).

$33-$34 051-052 FRETOP

As strings work downwards from the top of RAM, FRETOP keeps a track of the lowest byte used and it is this pointer which must not go below STREND as described above. The FRE(0) value is calculated by subtracting STREND from FRETOP.

$37-$38 055-056 MEMSIZ

This points to the first "unusable" byte and is determines the topmost byte available for strings. It can be lowered if necessary to allow space at the top of RAM for machine code, character generator tables and so on, but FRETOP is only set from it when a CLR is executed. Until then, the Vic continues to store strings according to FRETOP and may overwrite anything stored in that space.

Note that MEMSIZ does not indicate the uppermost byte of RAM - it is the limit of space USABLE by BASIC.

$39-$3A 057-58 CURLIN

If you need to know the line number currently being executed, then FN DEEK(57) will tell you.

$3F-$40 063-064 DATLIN

FN DEEK(63) will tell you the line number of the last item of DATA read.

$61-$65 095-101 FAC

This is the main floating point accumulator where nearly all numeric computation in the Vic takes place.

Numbers are stored here in a special form with the first byte holding the exponent and the other four the mantissa of the floating point number. The actual construction of this type of numeric storage is fairly complex and I hope to explain how this is done in the not too distant future.

$73-$8A 115-138 CHRGET

This is a machine code subroutine which is at the heart of the interpreter. It contains a two byte pointer ($7A-$7B 122-123 TXTPTR) which points to the last character read.

When the interpreter needs to read a character from the program, it calls CHRGET which first increments TXTPTR, gets the character at that location, ignores it if it was a space, gets the next character, and sets up flags to indicate whether the character read was a numeric digit or a statement terminator (that is a character indicating the end of a BASIC statement - a colon or end of line marker).

It is this routine which some programs modify to allow the inclusion of new BASIC commands.

$8B-$8F 139-143 RNDX

These five bytes hold the last random number generated and are used as a seed to generate the next random number.

If you use RND(1) then the random number generated is based on the contents of RNDX. RND(-R) will store R in RNDX and then generate a random number from it - this way you can force a random number to be the same each time the program is run. RND(0) will cause RNDX to be set at random from internal registers and can be used as a RANDOMISE command - it should ideally only be used at the start of a program.

$91 145 STOPFL

When the STOP/RUN key is pressed, this location is set to $FE (254) by the keyboard scanning routine. It is this flag which is disabled when the normal STOP-key disable routine is used.

STOPFL is checked before each statement is executed and not during the execution of a statement. If a statement were stopped in the middle, it may not be possible to recover properly when CONT is typed.

However, certain operations (notably some input/output routines) include their own special check of the STOP key and this is done directly and not using STOPFL.

$9D 157 MSGFLG

This byte is normally 128 and is used to control the printing of Kernal input/output error messages.

There are two categories of error messages - the normal error messages such as "DEVICE NOT PRESENT", and the equivalent kernal message "I/O ERROR #5".

If set to 0 then NO input/output error messages will be printed. However, if bit 6 is set to 1 (POKE 157,64) then the kernal error message will replace the normal error message.

The only exceptions are the "PRESS RECORD" or "PRESS PLAY" messages and the "EXTRA IGNORED" and "REDO FROM START" error messages which will always appear.

$C5 197 LSTX

If no key is pressed this location holds the value 64, but if any key (except SHIFT, STOP, CBM, CTRL or RESTORE) is pressed, it will contain the matrix value of the key. For instance the number "1" gives a value of 0 and the letter "F" gives "42".

To convert from this keyboard value to an ASCII character value requires the use of the keyboard conversion tables in the Vic's ROM.

These are at $EC5E (60510) for normal keys, $EC9F (60575) for shifted keys, $ECF0 (60656) for CBM graphics keys and $EDA3 (60835) for CTRL keys. To set A to the character value for a normal key, use: A = PEEK(60510+PEEK(197))

Note that $CB 203 KEYVAL contains the same information although it is used slightly differently by the Vic.

$C6 198 NDX

This hold the number of characters in the keyboard buffer. It can be used to clear the buffer if you suspect that spurious keys have been pressed prior to a GET or INPUT - this is done with: POKE 198,0

$D1-$D2 209-210 SCRPTR

This points to the RAM location of the first character of the current screen line. If the line is a continuation line, it points to the first character of the first line of the complete long line.

$D3 211 CPOS

Contains the current position of the cursor on the line. By adding it to SCRPTR it is possible to obtain the address of the the cursor or the position where the next character will be printed. The first position on the line is 0 and the value of CPOS can go up to 87 as the cursor wraps around onto continuation lines.

$D5 213 SLINEL

Holds the actual length of the current screen line, including wrap around continuation lines. Note that the value is actually one less than the number of characters.

$D6 214 SLNUM

The current line number of the cursor - this time the lines are the physical lines on the screen and not the logical line numbers.

$D9-$F1 217-241 SCRPTH

The start address of each screen line is held in two parts. The most significant byte is held in the SCRPTH table, the least signifcant byte in ROM at $EDFD (60925).

As soon as anything is typed beyond the end of a line on the screen, the rest of the screen is pushed down and the new line is "attached" to the original line. This is the so-called wrap-around facility and, at least until the screen is cleared, these long lines will remain joined up.

To identify which lines are continuation lines and which are not, the SCRPTH table is used. Bit 7 of each byte is set to 1 for each screen line. As soon as a screen line becomes part of a long line then it has bit 7 cleared. The only address that matters in this instance is the start address of the very first line of the group since the continuation lines are from then on considered as part of this first line.

There may be occasions when it is necessary to force all lines to be "single" lines and to cancel any long lines. This can be done by clearing the screen (NOT homing the cursor!) but this may not be appropriate. The following routine will cause the desired effect:

```
FOR I = 217 TO 241: POKE I, PEEK(I) OR 128 : NEXT
```

This wrap-around technique leads to some quirks in handling the screen. For instance, you may have noticed that, as you write off the bottom of the screen, sometimes the screen will scroll up more than one line - it does this if there is a long line at the top of the screen so that it shifts the whole of the line off the screen.

$F3-$F4 243-244 KEYMTX

As already mentioned, there are several decoding matrices to convert the key code into a character code depending on whether the normal, shifted, CBM or CTRL characters are being accessed.

KEYMTX holds the address of the last matrix used - unfortunately it is not possible to alter this address from BASIC although it is possible to examine the address (using FN DEEK(243)) which will show the most recent matrix used.

$281-$282 641-642 LORAM

Points to the lowest USABLE RAM byte and is set up by the switch-on initialisation routines. FN DEEK(641) will return a value of 4096, 1024 or 4608.

$283-$284 643-644 HIRAM

After finding the first usable byte of RAM, the initialisation routines then test every subsequent byte until no more RAM is found. The address at which this occurs is placed in HIRAM and is the value which decides where the screen memory will begin.

$288 648 SCRAMH

This is the most significant byte of the start address of the screen memory. The actual start address can be found by PEEK(648)*256.

$28D 653 SHFFLG

Keeps a track of whether the SHIFT, CBM or CTRL keys are pressed. Bit 0 is set if SHIFT, bit 1 if CBM and bit 2 if CTRL. So that is PEEK(653) is 3 then both the CTRL and CBM keys are being pressed together.

ANOTHER BUG!

Users of the machine code monitor cartridge may have noticed that, if they exit to BASIC with the "X" command, any BASIC program already in the Vic is likely to be corrupted.

It would appear that the monitor changes location $2B (the first byte of TXTTAB) under certain circumstances, and as a result, the Vic thinks the program is starting somewhere other than its true start.

There is a quick and easy fix - just POKE 43,1 after leaving the monitor to regain the BASIC program.

COMMODORE'S COMPETITION

Many of you may have read in the press of the Commodore software competition earlier in the year. The rules required original programs that were "inventive and instructive" and were written for the Vic or the 4032, and I was asked to assist in the judging, together with Gail Wellington and Graham Sullivan of Commodore.

I spent a fascinating afternoon looking at the entries, of which there were over eighty. I have to say that I was rather disappointed by the standard of inventiveness since most programs were only variations on unoriginal themes.

The winner was Bob Tulloch of Kings Lynn in Norfolk and his entry was a very simple program for the Vic, designed to help him teach his very young daughter to count - different numbers of simple, but beautifully and colourfully designed, objects would appear on the screen. The child hits the keyboard, once for each object, and the numbers appear over the top of the objects. Although very simple, its design was superb and the fact that it was used as a teaching aid and not as a teaching program appealed to all the judges. You will read more in the press, but it just goes to show how a successful idea can be a simple idea.

THE COMMODORE 64

Following my brief discussion of the Commodore-64 last time, I would like to give a little more information about the 64K RAM available.

The 64 comes with 64K RAM, but only 38K contiguous bytes of RAM usable by BASIC, and I suspect that the practical limit will be 32K. The BASIC ROM is at $A000-$BFFF and the Kernal at $E000-$FFFF, with the space $C000-$DFFF being used by the I/O chips and some "spare" RAM. Plug-in ROM packs are designed to fit in anywhere between $8000-$9FFF.

Obviously, there is more memory available than can be practically used at any one time - for instance there is RAM at the same address as both the BASIC and Kernal ROMs but this is not normally accessible. However, it is possible to "bank-switch" these ROMs out of the 64's memory space and use the RAM instead, but it must be stressed that this can only be done for machine code programs, and only machine code can therefore take full advantage of the full 64K RAM.

To allow the memory map to be moved around there are a variety of control lines available. For instance there are three available to software through the new I/O register at $01 ($\overline{\text{LORAM}}$, $\overline{\text{HIRAM}}$ and $\overline{\text{CHAREN}}$); there are two hardware control lines on the cartridge socket ($\overline{\text{GAME}}$ and $\overline{\text{EXROM}}$) and there are two on an I/O register in one of the new interface chips ($\overline{\text{VA14}}$ and $\overline{\text{VA15}}$).

These last two are only used to determine where in the main address space the video chip will access the screen RAM, colour RAM and character generator. The character generator sits between $D000 and $DFFF and is not normally available since the I/O chips etc., are situated here. If the programmer does need to get at the character generator then this can be done by switching it into the address space using $\overline{\text{CHAREN}}$; this is only useful if it is necessary to transfer the character generator into RAM.

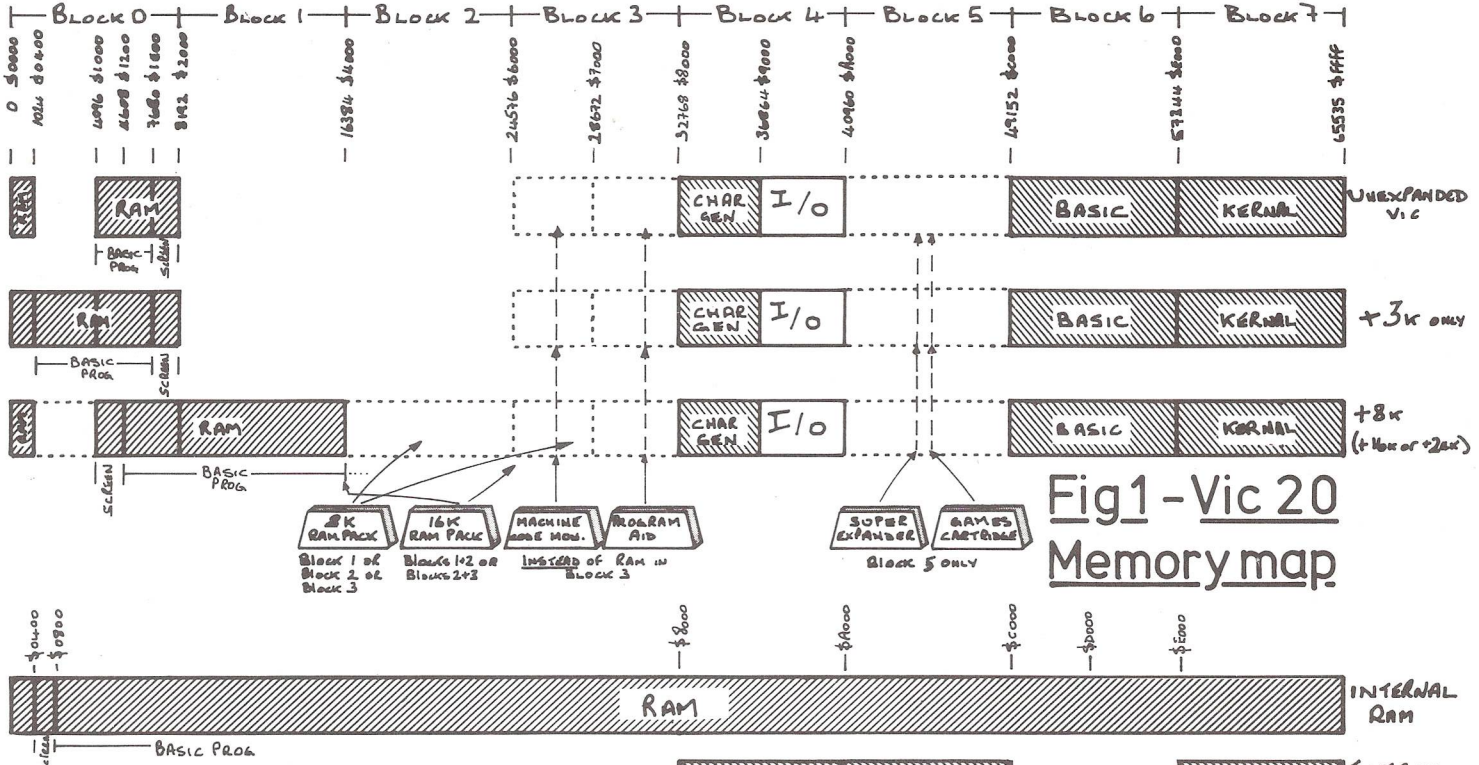


Fig 1 - Vic 20 Memory map

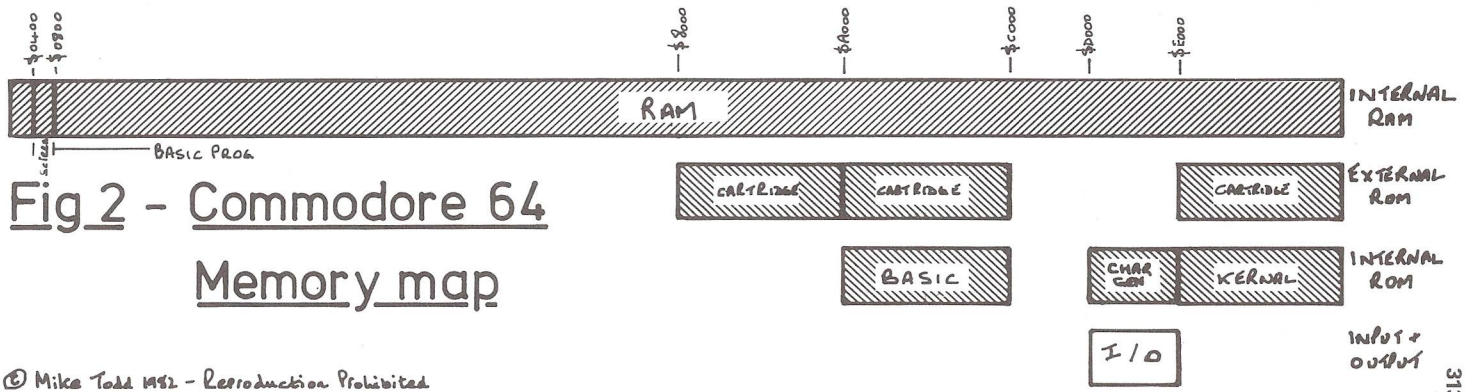
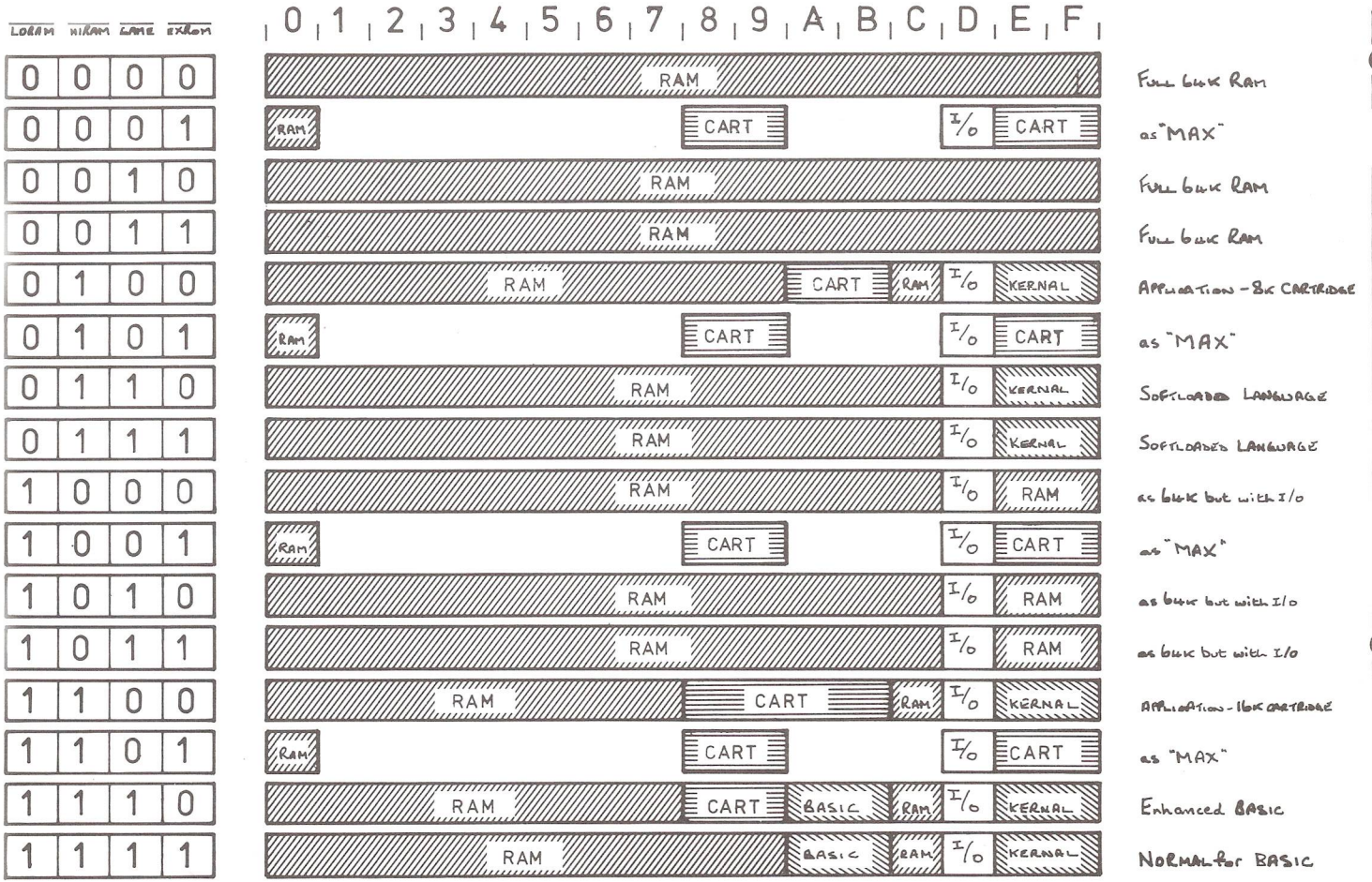


Fig 2 - Commodore 64 Memory map

Fig 3 - Commodore 64 configurations



$\overline{\text{LORAM}}$ and $\overline{\text{HIRAM}}$ are used to replace the BASIC and KERNAL ROMs with RAM - GAME is used by MAX-type cartridges to change the memory map from the 64 to the MAX memory map, and $\overline{\text{EXROM}}$ allows the top 8K of BASIC RAM ($8000-$9FFF) to be replaced by ROM. Unfortunately, all these memory controls interact upon each other and Fig.3 shows the various combinations of $\overline{\text{LORAM}}$, $\overline{\text{HIRAM}}$, $\overline{\text{GAME}}$ and $\overline{\text{EXROM}}$.

With the normal BASIC and KERNAL ROMs still in place, it is still possible to write to the "parallel" RAM since a write to one of these addresses will always go into the underlying RAM, while a read will, of course, read the ROM itself. This could be useful if the hi-resolution screen is used which needs a full 8K of RAM and which could access this hidden RAM; it would mean that bank-switching of this RAM into the memory map would not be necessary for writing to the hi-res screen.

Fig 2 actually shows the overall memory map and what can be accessed at each point. The space from $D000-$DFFF contains the I/O devices as follows:

$D000-$D02E Video controller chip
$D400-$D41C Sound controller
$D800-$DBFF Screen colour RAM
$DC00-$DC0F Complex interface adapter (CIA) #1
$DD00-$DD0F Complex interface adapter (CIA) #2
$DE00-$DFFF Available for expansion (CP/M, IEEE maybe?)

Finally on the 64, the power supply is slightly different to that on the Vic-20, which only had a low voltage AC input. The 64 uses a 7-pin DIN plug with a 5-volt DC supply and a 9-volt AC supply from the power unit.

Most of the information given comes from playing around with an NTSC (American) 64 in the Spring, backed up by some of Commodore's preliminary documentation. Next time I hope to have had a PAL (UK) 64 and should be able to give even more details of this fascinating machine.

AND FINALLY?

Many members have been writing to me about problems that they've been having with the Vic (and indeed other Commodore machines) and I've not really been able to answer all of them personally.

Not only do I not have the same configurations as these people, but I don't have the time to do the, often substantial, research needed.

Therefore, elsewhere in the Newsletter you'll find a PROBLEMS page, which also includes three Vic problems. I hope to try to keep this idea going, although we do have a techical queries panel and I don't want to do them out of a job - mind you, I don't recall ever seeing their names and addresses published in the Newsletter, do you? [Watch it! - Ed].

Anyway, for the time being I think that the PROBLEMS column could be a useful source of information and you'll find more details in the column itself.

AN ADVERT! - NEW VIC FOR SALE

Finally, finally - I've got a brand new Vic-20 for sale. It was bought at the beginning of October and has only been used for about 5 hours in total since then. It cost about £167.00 and I'll accept any reasonable offer over £140.00. If required, I will hand on the receipt to whoever buys it.

The cassette machine that was bought with it has already been sold, but I have an old cassette machine that I'm prepared to sell with it. It's in good condition and works fine and I'll accept £20.00 for it - making a total price of £160.00. I'll also "throw in" some blank cassettes at no extra charge. I'm afraid that the buyer will have to arrange collection, but I'm sure that we could come to some arrangement.

--oOo--

	Xerox 820	IBM Personal Computer	Apple III
Standard Memory	64K	64K	128K
Maximum Memory when fully configured*	64K	192K	256K
Expandability	No expansion slots	No extra expansion slots in fully configured* 192K system	4 extra expansion slots in fully configured 256K system*
Diskette Storage (per drive)	92K	160K	140K
Mass Storage (per drive)	–	–	5 megabyte Hard Disk
Display Graphics Capability	High resolution B/W	High resolution B/W or 4-color (color requires additional card)	High resolution B/W or 16-color

How Apple gives others the pip....

HOW COMMODORE PIPS APPLE.

FEATURES	COMMODORE	APPLE II+
Base Price	£299*	£499*
ADVANCED FEATURES		
Built-in user memory	64K	48K
Programmable	YES	YES
Real typewriter keyboard	YES (66 keys)	YES (52 keys)
Graphics characters (from keyboard)	YES	NO
Upper & lower case letters	YES	NO**
Function keys	YES	NO
Maximum 5¼" floppy disk capacity per drive	170 K.B. to 1 M.B.	143 K.B.
AUDIO FEATURES		
Sound Generator	YES	YES
Music Synthesizer	YES	NO
Hi-Fi Output	YES	NO
VIDEO OUTPUT		
Monitor Output	YES	YES
T.V. Output	YES	EXTRA
INPUT/OUTPUT FEATURES		
Cassette Port	YES	YES
Intelligent Peripherals	YES	YES
Serial Peripheral Bus	YES	NO
ADDITIONAL SOFTWARE FEATURES		
CP/M® Option (over 1000 packages)	YES	YES
External ROM cartridge slot	YES	NO

*EXC. VAT – DETAILS CORRECT AT TIME OF GOING TO PRESS
**UPPER ONLY
CP/M® IS A REGISTERED TRADEMARK OF DIGITAL RESEARCH, INC.

commodore

STRICTLY FOR BEGINNERS - 6

This month I am pleased to present to you an improved version of a small addition program I produced in the May issue. This has been sent to me by a reader in Sheffield, Mr. Simm. The program is called 'Addition List', and the listing is as follows:

```
 100 PRINT"<clr>ADDITION LIST"
 110 PRINT"##############":PRINT:REM SHIFTED # (4032)
 120 T=0:A=0
 130 INPUT"<dn>VALUE";N
 140 IF N=0 THEN GOTO 170
 150 T=T+N:A=A+1
 160 GOTO130
 170 PRINT"<dn>ENTRIES = ";A,"TOTAL = ";T
```

Do you see the improvement made by Mr. Simm ? The program now not only counts the amounts entered in the INPUT in line 130, but also counts how many inputs you make. Very good. Any other ideas on the programs you have seen so far?

DATA DELETION in Programs.

Suppose you have some data in an array of alphanumeric data called P$(X) (where X is the total number of names or whatever in the array).

One of these names now becomes superfluous, and you wish to delete it. This routine will do just that:

```
 450 REM ** DELETION **
 460 INPUT"NAME REQUIRED";N$:L=LEN(N$)
 470 FORI=1TOX
 480 IFLEFT$(P$(I),L)=N$THEN510
 490 NEXT
 500 PRINT"<cd><rvs>NO<off> <rvs>SUCH<off> <rvs>NAME<off>
     <rvs>ON<off> <rvs>RECORD<off>
 510 K=I:PRINT"<dn><rvs>"P$(I)" DELETED":FORI=KTOX
     :P$(I)=P$(I+1):NEXT
```

The explanation for this is simply that one searches for the name required in line 460, using the length (L) of the search name (N$) so that one need only enter enough

digits of the name to identify it from any other similar names in the array. e.g. If the names stored in P$ are; P$(1) = ADAMS; P$(2) = ADAMSON; P$(3) = ADSAM then inputting A will find ADAMS; inputting AD will also find ADAMS; inputting ADS will find ADSAM; to find ADAMSON it is necessary to input ADAMSO or the full name. If there are two ADAMS, then one would need to input their initials also, to distinguish them. If their initials were identical, then there is no way of distinguishing them with a simple program like this. But that is not too common an occurrence.

When a match is found, in line 480, the program branches to line 510. If no match is found, and the program reaches the end of the array, at which time 'I' will be equal to X+1, the program will continue to line 500 and print a suitable message on the screen.

Line 510 sets a variable (K) in this case, to the value of 'I' at the point of matching the name required with the appropriate name in the array. Then the name is printed on screen followed by the word DELETED. Then the array is set up again with each member of the array being renumbered 'downwards'. i.e. if the array was as described above, and P$(2) ADAMSON was deleted, K would be set to 2. P$(2) would then be made to equal P$(2+1), which is P$(3). Hence the new array would be: P$(1) = ADAMS; P$(2) = ADSAM. Thus the old P$(2) has been effectively deleted.

Here is a more complete program demonstrating the above:

```
 10 REM *** CREATE ARRAY P$( ) ***
 20 INPUT"<2dn>NO.OF NAMES IN ARRAY";X
 30 DIMP$(X+1):FORI=1TOX:PRINT"<rvs>"I"<off>";:
    INPUT"<dn>NAME";P$(I)
 40 IFP$(I)="ZZZ"THEN60
 50 NEXT
 60 REM *** LIST THE ARRAY ***
 70 FORI=1TOX:IFP$(I)="ZZZ"THEN90:PRINT"<rvs>"I"<off>"P$(I)
 80 NEXT
 90 REM *** DEMONSTRATE DELETION ***
100 INPUT"<dn>NAME TO DELETE";N$:L=LEN(N$)
```

```
110 FORI=1TOX
120 IFLEFT$(P$(I),L)=N$THEN150
130 NEXT
140 PRINT"<dn><rvs>NO SUCH NAME"
150 K=I:PRINT"<dn><rvs>"P$(I)" DELETED":FORI=KTOX:
    P$(I)=P$(I+1):NEXT
160 FORI=1TOX:IFP$(I)="ZZZ"THEN180
170 PRINT"<rvs><2dn>"I"<off>"P$(I):NEXT
180 PRINT"<dn>END OF DEMONSTRATION.
```

PEEK & POKE ON THE SCREEN

The words PEEK and POKE used to frighten me to death when I saw them in programs in the magazines. I couldn't fathom out what they did at all. Once again, they lost their fearsomeness when I started learning from my fellow enthusiasts at the USER GROUP meetings.

The PET screen is what is known as a Memory Mapped Screen. That is, each position on the screen has a particular place on an invisible map inside the PET's memory, which is labelled with a number. The top left hand corner of the screen, where the cursor goes when you press the CLR/HOME key, is position number 32768. The bottom right hand corner is number 33767.

Using these screen locations, as they are known, we can place bits of illumination in known parts of the screen. We can also erase them. This is useful (how's that for understatement) in games involving movements across the screen.

But games-type movements are a little bit complicated for beginners. So let us start with something simple.

POKE is the computing word for placing a value in a particular memory location. If that memory location is between 32768 and 33767 then the value POKE'd will appear on the screen. Try this:

```
POKE 32768,55
```

Lo and behold, a figure 7 appears at the top left hand corner of the screen! Now let us check this, by using the other function, PEEK. Try this:

```
?PEEK(32768)
```

This time we are asking PET to print on screen whatever value (ASCII value, that is) it finds at the top left hand corner of the screen. Provided you cleared the screen before you started the POKE command, the answer now will be 55. ASCII, incidentally, stands for American Standard Code on Information Interchange. And when I mentioned it above, I really meant CBM ASCII, because the makers of our beloved computer have altered standard ASCII to CBM ASCII, for their own purposes. Confused ? Well don't worry about it. There are available tables of ASCII values and characters for PET, so all you have to do is look them up. Unfortunately the CBM User Manual presents them in binary on Page A-13, which I do not pretend to understand. 'The PET Revealed', on the other hand, has excellent tables a few pages from the end. Or even easier, enter this program to see them on your screen.

```
 10 FOR J=32768 TO 32768 + 255
 20 POKE J,K: K=K+1
 30 NEXT
 40 POKE 59468,12: FOR J=1 TO 500:NEXT:POKE 59468,14
     :FORJ=1TO500:NEXT:GOTO40
```

[I hope you all know what POKE 59468,12 or POKE 59468,14 does. For those who do not know, POKE 59468,14 changes the writing on your PET screen into lower case and graphics characters, and POKEing the same location with 12 changes back to upper case characters (Capitals).] See page A-12 in the CBM User Manual for models 2001-16 to 32N. Or this program to see them and their 'reference numbers' (i.e. decimal value).

```
 100 ?"<clr>":FORI=33TO255:?"<rvs>"I"<off>";CHR$(I)
     :FORJ=1TO500 :NEXTJ,I:IFN<>0THEN300
 200 POKE59468,14:N=N+1:GOTO100
 300 POKE59468,12:END
```

Note: 0 to 32 are blanks, so I have excluded them. There are also some more blanks, but I've left them in because they appear in the middle of the list (128-160). Did you notice what CHR$(186) was? If you did not, it was a tick. I'll bet you didn't know PET had a tick in its character set, did you? (And if you did, you are not a beginner and should not be reading this!!) If you want the list to go through more quickly, change the value 500 (try 50) in line 100.

I have just returned from the ICPUG stand at the Personal Computer World Show in London. I hope that at least some of the people who came to the stand for information were satisfied with the replies they received, and also that some of them have actually attended a Regional Group meeting by now. This is still the best way to receive help and knowledge about computing. I spent a good part of my time there quizzing Stephen Rabagliati about Prestel for the PET. Before the ladder hit him on the head (in the local pub) his answers were quite logical! What a wonderful invention is Telesoftware. No more typing in programs at the keyboard, just download them from Prestel's ICPUG page (and others).

Don't forget, any ideas for this page, or any more problems, please contact me. [Ed's Note: much of the principles in this column apply to Vic users.]

--o0o--

STARTING FORTH

By Ron Geere

Beginning next issue I propose to start a regular column devoted to the language Forth. There appears to be justification for such a column from the cross-section of interest at the AGM, Versions of Forth are available for both CBM/PET and Vic-20 with a '64' version to follow shortly.

Forth is characterised by five major elements: dictionary, stack, interpreters, assembler and virtual memory. Although not one of these is unique to Forth, their interaction in Forth produces a synergistic effect that creates a system of unexpected power and flexibility.

The dictionary is a threaded list of variable-length items, each of which defines a word of the vocabulary. Two push-down stacks are maintained, a parameter stack and a return stack.

Forth is fundamentally an interpretive system, i.e. program execution is controlled by data items rather than by machine code. Two levels of interpretation exist in Forth, the outer, or text interpreter, parses text strings and looks them up in the dictionary. When found, it is usually interpreted by invoking the address interpreter, also known as the inner interpreter. This interprets strings of absolute addresses by executing the definition pointed to by each. This operation is fast, being only a few machine instructions.

Forth includes a resident assembler to generate specific machine instructions - a sort of equivalent to the SYS command. Finally, Forth has a number of buffers in memory which relate to fixed-length segments of disk space. If a block is modified in memory, it is automatically replaced on disk.

Forth is a language which, at first had little use, but had time to mature. In consequence few 'dialects' of Forth exist, unlike BASIC, and in consequence the programs are very portable between greatly differing machines.

Next issue I hope to publish some items from those of you that expressed an interest. In addition, I will be describing some more features of the language.

--o0o--

SOFTWARE LIBRARY

VIC-20 SOFTWARE NEWS.

We now have two disks of Vic-20 software. Members wishing a directory listing should send a stamped addressed envelope to me, you are reminded that the original list should be returned to me.

Software for the 3.5K Vic contains mostly games. What is lacking in the library is:-

1) assembler/disassembler
2) utilities
3) large games programes

Would members please note this deficiency and help by sending me programs for the Library.

PET/CBM SOFTWARE NEWS.

We are now a member of Commodore's workshop library and have all their software from this service. This consists of a total of 12 disks, mainly of the educational type programs.

We have received a contribution from Bob Chappell. The disk (number 15) containes games, educational, business and miscellaneous programs, most of which have appeared in 'Microcomputer Printout'. The editor of the magazine has given permission for ICPUG members to use the software.

8050 FORMAT DISKS.

Members please note that 8050 format disks containing software can now be obtained from:- Stephen Rabagliati, 46, Milton Dene, Woodhall Farm, Hemel Hempstead.

GENERAL.

When requesting software from the User Group library please bear in mind the following points:-
1) limit your choice to two disks or four programs on cassette per request.
2) adequate postage, packing and an addressed adhesive label should be included.

Members are reminded that anyone writing to me who wants a reply should include a stamped addressed envelope.

Bob Wood, 13, Bowland Crescent, Ward Green, Barnsley, Sth. Yorks S70 5JS.

--o0o--

EXTENDING THE COMMODORE ASSEMBLER.

By John Stout.

Using the extensions to the Commodore assembler (version V121579 for BASIC 4) described here writing in assembly language becomes a lot more like writing in a high-level language with a full complement of programming structures. For example the programming to calculate the length of a string (including the carriage return) stored in memory starting at location START can be written as:

```
 LDX #$00
 .REPEAT
 INX
 LDA START-1,X
 CMP #RETURN
 .UNTIL EQ
```

The five programming structures provided by the extended assembler are IF/THEN/ELSE, REPEAT/UNTIL, WHILE, LOOP/EXITIF/ENDLOOP and CASE/WHEN/ENDCASE. All the new structure words are preceded by a full stop as shown above and have the following syntax:

```
 .IFF <test>
 code to be performed if <test> is true
 .EIF
 .IFF <test1> THEN
 code to be performed if <test1> is true
 .ELSE [<test2>]
 code to be performed if <test1> is false
 .EIF
```

Anything enclosed in square brackets is optional. If [<test2>] is included the assembler can generate more efficient code, substituting a branch to the .EIF rather than a JMP. Note that if there is to be a .ELSE part the

```
 .REPEAT
 code repeatedly performed until <test> is true
 .UNTIL <test>
 .WHILE
 code to set up a test result
 .DOIF <test1>
 code repeatedly performed while <test1> is true
 .EWHILE [<test2>]
```

Again [<test2>] allows more efficient code to be produced.

```
.LOOP
code performed repeatedly until a .EXITIF succeeds
.EXITIF <test1>
code performed repeatedly until a .EXITIF succeeds
.EXITIF <test2>
.
.
.ELOOP [<testn>]
```

There can be as many .EXITIFs as required inside a efficient code to be generated.

```
.CASE <register>
.WHEN <value1>
code performed when <register>=<value1>
.EXCASE [<test1>]
.WHEN <value2>
code performed when <register>=<value2>
.EXCASE [<test2>]
.
.
.ECASE
```

There can be up to 10 alternatives (.WHEN/.EXCASE structures) inside a .CASE/.ECASE structure. [<test1>], [<test2>] etc produce more efficient code.

In addition to the above structures the statement jump (or a branch if followed by an optional test, e.g. two-character representations of the required tests, i.e. EQ, NE, CC, CS, PL, MI, VC and VS. <register> in the .CASE structure is either A, X or Y. The <value>s in the .WHEN statements are any valid memory reference for a CMP, CPX or CPY statement (CMP if <register> was A, CPX if <register> was X and CPY if <register> was Y). Thus if <register> was A valid <value>s are ($55),Y and START-1,X.

CODE GENERATION.

The code generated by the assembler for the structures will usually be the same as that generated by hand coding the structures. The only occasion when the assembler will actually generate the code to set up a test value (rather than test the value) is in the .CASE structure when simple CMP/CPX/CPY followed by BNE statements will be produced. In all other cases you can use your ingenuity to set up the test.

```
.IFF <test>                 Bopp(<test>) EIiiii
code                        code
.EIF                        EIiiii
.IFF <test1> THEN           Bopp(<test1>) ELiiii
code                        code
.ELSE [<test2>]             JMP EIjjjj if no <test2>
B<test2> EIjjjj if <test2>
code                        code
.EIF                        EIjjjj
```

Bopp(<test>) signifies a branch with the opposite sense to <test>, e.g. BNE if <test> were EQ. iiii and jjjj represent four-digit numbers starting at 0000 which are used to generate unique labels for the structures.

```
.REPEAT                     REiiii
code                        code
.UNTIL <test>               Bopp(<test>) REiiii
UNjjjj
.WHILE                      WHiiii
code                        code
.DOIF <test1>               Bopp(<test1>) EWjjjj
code                        code
.EWHILE [<test2>]           JMP WHiiii if no <test2>
B<test2> WHiiii if <test2>
EWjjjj
.LOOP                       LOiiii
code                        code
.EXITIF <test1>             B<test1> EPjjjj
code                        code
.EXITIF <test2>             B<test2> EPjjjj
```

```
code                                code
.ELOOP [<test3>]                    JMP LOiiii if no <test3>
B<test3> LOiiii if <test2>
EPjjjj
.CASE <register>
[code]
.WHEN <value1>          CAiiiO CMP/CPX/CPY <value1>
BNE CAiiii1
code                                code
.EXCASE [<test1>]                   JMP ECiii9 if no <test1>
B<test1> ECiii9 if <test1>
.WHEN <value2>          CAiiii CMP/CPX/CPY <value2>
BNE CAiii2
code                                code
.ECASE                              CAiii2
ECiii9
```

The .LEAVE statement will generate a JMP (if not followed by a <test>) or a B<test> (if followed by a <test>) to a label which labels the end of the innermost enclosing .IFF, depth (stack storage will run out before the limit is reached).

ERRORS.

There are four main errors that can occur when using the structures. If you have a wrong statement word, e.g. be reported. This error is also reported if you miss off a compulsory <test> or use a <test> which is not one of EQ, NE, CC, CS, PL, MI, VC or VS. The structures use a stack which builds down from top of memory towards the assembler generated symbol table which grows up. Each structure uses two bytes on this stack (except for CASE which uses four) so only a small amount of stack space will be used unless you nest structures to any great depth. If however you miss out one of the structure statements, e.g. .IFF, but leave in the rest of the statements, e.g. .ELSE and .EIF, a 'warning: stack underflow' message may be printed to the screen. You may also get a large number of **undefined symbol** and **duplicate symbol** errors in this case.

LISTING.

The listing produced by the extended assembler will show the structure statements on a line, listed as if they were comments, followed by the code they generate on following lines.

FUTURE ADDITIONS.

The major programming task has been done to convert the assembler into a macro assembler and this may be one of the additions made in the future. In addition some way of generating long branches, e.g.

```
.IFF <test> LONG              B<TEST> SIiiii
JMP EIjjjj
SIiiii
```

would be useful since without this **branch out of range** errors may be reported. Finally it would be useful to have some way of changing the formatting imposed on the listing by the assembler so that the program structure could be emphasised by indenting the sections of code inside structures.

PATCHING THE ASSEMBLER.

LOAD the assembler and then enter the extensions (1736 bytes) from $2171 to $2838. Now alter the following instructions:

Location	old contents	new contents
$042D	A0 21	A0 28
$042F	A2 71	A2 39

This alters the start of symbol table from $2171 to $2839

$043D	20 1A 06	20 37 22

This initialises the locations used by the extensions at the start of pass 1.

$05E2	20 1A 06	20 37 22

This initialises the locations used by the extensions at the start of pass 2.

$0910 4C 72 0F 4C 2D 23
This adds the structure statement words into the assembler.

$1F0E 20 CF FF 20 B3 22
This replaces the character input routine of the assembler with that in the extensions part of the assembler.

The extensions for BASIC4 are now in the ICPUG software library and can be obtained by members via Bob Wood. Note that the Commodore Assembler must be purchased via your Commodore dealer if you require it.

--oOo--

REVIEW

AUTO-1 A Club Software Review

It is club software review time again and this issue I turn my attention to the Merseyside area and Mark Atherton formerly of Merchant Taylors' School, Crosby in particular.

Most of you with disks and BASIC4 will know that pressing SHIFT RUN will automatically LOAD and RUN the first program on a disk. Unfortunately for any other program on the disk one still has to type DLOAD"PROGNAME". Now I like my program names to mean something so they can be quite long. Even though one can use pattern matching to save typing it can sometimes take a while to key in the names of programs I want to RUN.

Mark's AUTO-1 program was the answer to my prayers. What it does is to read the disk directory and display on the screen in an easy to read format all the file names together with a key associated with each file. All one does to LOAD and RUN a program is to RUN Mark's program and hit the appropriate key. If there should be more programs than screen space will allow one just presses the 'space' bar to view the next set of file names. All very simple and very, very useful.

I have been using the program with a BASIC4 'thin 40' PET and a DOS2A disk drive. Certainly it would need some modifications for BASIC2 but I am not sure how it would react with other BASIC4 PETs or different disk DOS. I have the program as the first program on all my disks. I can then RUN any program by 2 key presses- SHIFT RUN and the file key. Excellent Mark. Any body interested should contact David Jowett, 197, Victoria Rd. East, Thornton, Blackpool (Remember SAE).

STOP PRESS:

A version of AUTO-1 also exists which allows an option to just LOAD a program in addition to the normal LOAD and RUN. This is very useful when loading machine code to the top of memory. If the RUN is executed in this case AUTO would RUN itself!

****

Any regional groups with some good PET software freely available might like to send me a copy for review in this series. My address is as usual, 73, Minehead Way, Stevenage, Herts. SG1 2HZ. Let the rest of ICPUG know what you are doing...

****

Brian Grainger.

--oOo--

Just as my 8032 arrived, they announced the 700-series.

STORAGE TECHNOLOGY UPDATE

By Tom Cranstoun.

Once upon a time there was a standard for 'floppy' disks, a standard set by IBM (remember them) for 8" floppy disks. This was such a good idea that everyone jumped on the bandwagon and produced 'IBM'-compatible floppies. Then someone invented the 5 and a quarter inch drive. IBM weren't interested (at the time) and no standards were set. So computer manufacturers all produced their own disk formats for 5" floppies (Commodore doing more than most by providing three different formats!!). Times are changing and the latest developments are leading toward a 3.5" floppy disk; hopefully some sort of standards will arise. The Japanese appear to have different thoughts. The following article outlines current developments. Micro Floppies (hereafter called uFloppies) are going to provide the future in floppy disk design. Twelve major manufacturers have produced a specification:

1) The µFloppy will be small enough to fit in a shirt pocket.
2) It will be plug-compatible with existing products in the number of tracks per side (either 40 or 80) this enables existing software and drive controllers to be used.
3) The package will be in a hard shell to provide extra protection.
4) Reliability is supposed to be taking precedence over compactness.

The above spec has been agreed as the major points by all manufacturers: major differences occur at the important end - the media interface. Sony are to produce a device which rotates at 600 revolutions per minute. The americans are going for 300 revs. The capacity of these drives will be tremendous, it is in this area that you can readily see the advance in disk technology as well as the different approaches being taken. The American standard proposes one megabyte (that is one million characters) per side of the disk (compare this with the 4040 -170K or 8050 -500K). Whereas Toshiba are promising a drive with 3 megabytes per side. Hitachi currently reckon on 500K per disk. One thing is for sure, no-one, not even the manufacturers, are sure

about compatibilty in the future, which is a great shame as this would have been the time to review current methods and produce a cheaper more reliable mass market storage device.

We can be sure that 3.5" uFloppies will be around offering much more storage than the present 5.25" floppies but will have to wait for the 1" floppy for a set of standards!! I have seen a 3.5" floppy system working to a PET at the Computer Fair in Earls Court last Easter. It had a lot of promise but for the time being I'll stick to 4040s.

--oOo--

ROUND THE REGIONS

The highly successful Watford regional group recently had a demonstration of some business software presented by Harry Broomhall. Packages covered word processing, accounts and a 'sheet' modeller of an advanced specification. The evening was rounded off with a programmers clinic. The meeting on November 10th hosts Dr. David Annal, our resident expert on interfacing, and his talk is naturally on the 'ins & outs of interfacing'.

Robin Harvey and Alison Schofield are hoping to form an ICPUG regional group in North Gloucestershire, based on Cheltenham and starting in January 1983, when Alison will have time to act as group secretary. Interested persons should contact either R.C.Harvey, 30, Wimbourne Close, Coombe Glen, Cheltenham, Glos., tel: (0242) 27588 (home) or Mrs. A.Schofield, 78, Hesters Way Rd., Arle, Cheltenham, Glos., tel: (0242) 580789 (home).

An exhibition due to be held at the Queen's Hotel, Cheltenham, on Tuesday, 14th December from 12.00 noon to 10.00p.m. It is hoped to use this to launch the group, PET and Vic users both welcome.

After several set-backs in the continuity of meetings, the Hampshire region (region C) has been resurrected with the aid of a couple of Vic enthusiasts, Graham Hunt and Roger Chessell. Meetings are currently at Graham's house on the third Wednesday of the month, the address being 70, Reading Road, Farnborough, Hants. The editor remains as the regional representative to whom enquiries should be addressed. Since the venue is in the corner of the region, being one mile from Surrey and three from Berkshire, a cluster of attendees from Winchester propose that the region be reconfigured. This is logical since those from the depths of Hampshire are some 80 miles from the venue.

A number of 'self-help' groups, particularly of Vic users, have sprung up and not being affiliated to ICPUG they are missing out on access to facilities and information provided by the Group. If you know of any such groups and can find a spokesperson, do put them in touch with the regional Group co-ordinator (see inside front cover).

--oOo--

THINK ABOUT IT...

He who works a lot makes many mistakes; he who does not work does not make any mistakes; he who does not make any mistakes will be promoted and richly rewarded....

--oOo--

REVIEW

Beginning COMAL Teachers Copy £ 10.00
By Borge Christensen
Ellis Horwood, Market Cross House, Cooper St., Chichester.

This book on COMAL has been written by one of the founding fathers of the language. It is aimed at students programming, probably learning a computer language for the first time and to that market is a very good book. The teachers copy could also be used as a self instruction text.

The style of the book is to introduce concepts in a gradual manner with many examples and questions for the student interspersed in the text. The book assumes that the programs used in the course are held on a disk. As the programs are printed in the teachers copy there is no problem in preparing this disk.

The book has been written for the student using COMAL on a Commodore computer and although it does not say so, it also assumes a FAT40 or 8032 in certain examples.

All the basic elements of the language are introduced by examples starting with printing and listing, going on to input and reading of data and slowly introducing the COMAL structures. In passing through the examples such things as the MOD and DIV functions are introduced. Towards the end of the book the tempo speeds up and a lot is introduced at once.

My first impression of this book was that it was going to be terribly slow to read. However I found it interesting and the programs introduced were relevant rather than the usual artificial examples. There seem to be very few errors although confusion could be raised by what is NOT said. For example it says in the later exercises that data files can be on cassettes, but the programs are written as if they were on disk, with no instructions on conversion. My only criticism of this book is that not all of the language was introduced. REF parameters and FUNCtions are examples of this and the DIM statement was not very thorough.

To sum up a good text for learning COMAL on the CBM especially if one has had no introduction to programming before. However it does not introduce the full power of the language. Perhaps that is just as well with beginners.

B.D.G.

--o0o--

ANOTHER THOUGHT

No man is useless - he can always serve as a bad example.

--o0o--

DISK FILE - SECTOR 3

By Mike Todd.

Well, at last the DISK FILE is back again! This time I will explain the interface chips used by the Floppy Disk Controller and also how the FDC software reads and writes to the disk. The article assumes that you've read at least the previous DISK FILE and also that you've got the diagrams on pages 156/7 of the May Newsletter in front of you.

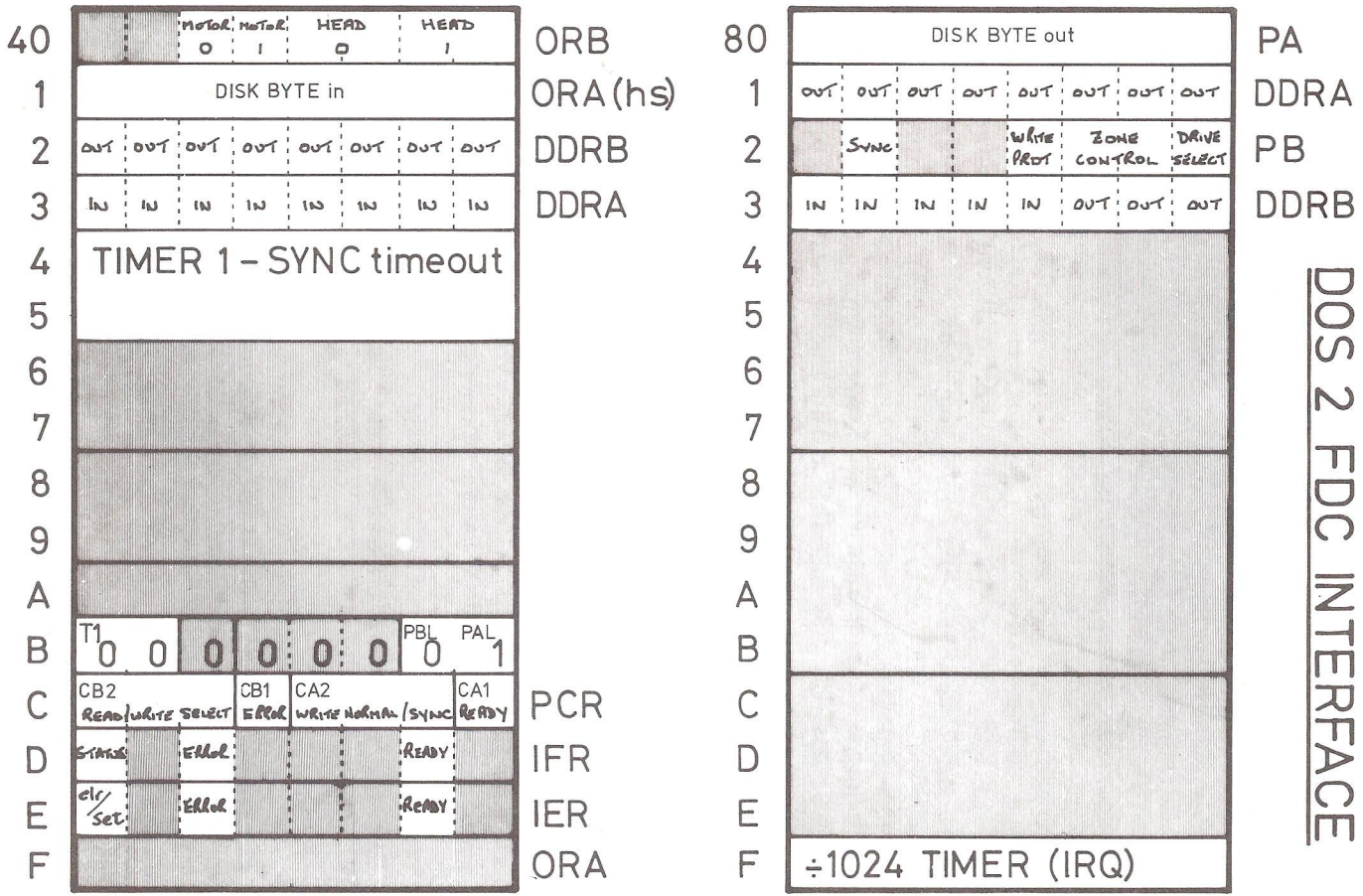
I will also repeat my warning that, what I'm talking about relates to DOS2.1 - although all the other DOS's are similar, they may not use exactly the same coding at the same place. I have also included some snippets of the actual code used in the FDC, together with the addresses at which it's located. This could be useful if you're trying to find your way around the FDC ROM, assuming that you've actually managed to get a dump of it!

I've also prepared memory maps of the interface chips used by the FDC. A 6522 at $40-$4F and a 6530 at $80-$8F.

$40 contains the main motor controls - bits 4/5 are used to turn the drive motors on and off (on=0, off=1) and bits 0-3 control the head stepper motor. By cycling the appropriate two bits through 00-11 or downwards from 11-00 the head can be made to step in one direction or the other. The drive needs a finite time to get up to speed and an interrupt routine handles the necessary flags to indicate that this has occurred. The stepper motors are a bit more complex and all I will say at this stage is that the interrupt routine steps each head one cycle on each interrupt according to a control register which determines both the direction and extent of the head's movement.

$41 is the byte received from the disk decoder ROM (see Diagram 2). This byte is latched into the register only when a complete byte has been read, setting the READY (CA1) flag at the same time.

DOS 2 FDC INTERFACE



Timer 1 of the 6522 is used as a time-out timer while searching for SYNC pulses, and timer 2 is not used at all.

The control register at $4C handles the R/W and MODE lines as well as determining the polarity of the ERROR and READY pulses. Bits 0 (=0) and 4 (=1) should always be preserved if this register is altered.

MODE=0 for normal writing (bits 321 = 110) and MODE=1 for writing a SYNC pulse (bits 321 = 111). To select WRITE mode, R/W=0 (bits 765 = 110) and for READ, R/W=1 (bits 765 = 111).

Although the interrupt line of the 6522 is not connected, use is still made of two of the interrupt flags for ERROR and READY, although the ERROR flag is normally disabled during a WRITE operation. The actual flags are detected using bit 7 of $4D which is 1 if either of these flags is set. The flags are cleared by a read of the appropriate data register.

In the 6530, register $80 is the byte to be written to the disk and is latched out of the register and into the write logic by the READY signal. Once this has been done, the next byte can be placed in the register.

The second I/O register of the 6530 is at $82 and has several functional bits. Bit 6 is normally a 1 until a SYNC byte is received (that is 10 or more consecutive 1's on the disk) when it goes to 0.

Bit 3 is used to detect if the write protect notch is covered up and is 1 if it's covered and a write operation is not allowed.

Bits 1-2 control the timing according to which zone the disk is operating in. Referring to the table on page 159 of the May Newsletter, you will see the zones listed and these two control bits correspond directly to the zone number.

The read and write electronics are shared between the two drives and bit 0 of $82 selects which of the two drives are required.

On the circuit diagram for the 8050, bit 4 of $82 is also available, although not actually connected. It is designated as ODD HEAD and presumably is designed to cope with the possibility of double sided disks in the future.

The only other register in the 6530 used by the FDC is the timer. Although there is only one timer in the 6530, it has several control registers, each having a different scaling factor for the count down. The FDC uses the 1/1024 count down which generates an interrupt when it reaches zero. The clock is 1MHz, which means that this counter will decrement every 1024/1000000th second, and the usual timing constant is 15 - giving an interrupt approximately 65 times per second. This is used to control the head stepping and motor "up-to-speed" delay.

Before reading or writing, it is assumed that the head is on the correct track on the disk, the drive has been selected, the motor is running at full speed and the correct zone has been selected.

Once this is done, the appropriae read or write mode must be selected in $4C. The coding used is as follows:

```
      LDA #$FC    ;%111 1 110 0 - READ mode
      STA $4C

or    LDA #$DE    ;%110 1 111 0 - WRITE SYNC mode
      STA $4C

or    LDA #$DC    ;%110 1 110 0 - WRITE NORMAL mode
      STA $4C
```

To read a byte, the READ mode is selected as above (in fact the FDC is normally left in READ mode) and the disk must be searched for a SYNC pulse which will mark the start of the

read operation. This is done using bit 6 of $82 and a loop is set up, waiting for a 0 in this bit.

However, if there were no SYNC pulse on this track (if there's no disk in the drive or the disk is unformatted) the loop would continue indefinitely and the FDC would hang up. To avoid this, a time out delay is set up and if no SYNC is received within the time limit the FDC will abort with a "NO SYNC" error. The actual coding used is as follows:

```
FF3F  LDA #$D0
FF41  STA  $45     ;set timer
FF43  LDA #$03     ;an irrelevant instruction

FF45  BIT  $45     ;check for time out
FF47  BPL  ERROR   ;error if timed out
FF49  BIT  $82     ;check for SYNC
FF4B  BVS  $FF45   ;loop if not present

FF4D  BIT  $40     ;reset ERROR flag
FF4F  BIT  $41     ;reset READY flag
```

Note the two BIT instructions which are used to clear the ERROR and READY flags.

Once the SYNC pulse has been found, the data byte doesn't start shifting until the sequence of 1's is finished and the first bit of data is being read (this occurs at point 3 in Figure 2). At this point, the first byte of data is read, and the READY flag is set when the byte is completed. The READY flag is checked and the read is performed:

```
FF51  BIT  $4D     ;is READY?
FF53  BPL  $FF51   ;loop until it is
FF55  LDA  $41     ;get the byte and reset READY flag
```

The code at this location is actually reading the first byte following the SYNC pulse which is a special identifier byte. As you may recall, the first byte following the SYNC pulse must have a 0 in the first bit position since it is

this which indicates the end of the SYNC pulse and initiates the data shifting process. This is achieved by arranging that the first byte is a special identifier byte (either $08 or $07) and the first bit will therefore always be 0.

Subsequent bytes are read using the same general idea and as many bytes as required can be read, although in practice this is normally limited to 256 bytes.

Writing is a little more difficult and the first check will be to see if the write protect notch is covered:

```
FDF4  LDA  $82
FDF6  BIT #$08     ;is write protect on?
FDF8  BEQ  $FDFD   ;continue to write if not
FDFA  JMP  $FF08   ;otherwise, jump to error routine
```

When formatting a disk under DOS1.2, the write mode is selected BEFORE checking for the write protect. If the write protect was then found to be on, the error routine failed to turn the write mode off! This meant that the erase portion of the head in question was left on and anything passing under the head would be erased.

Since the read mode didn't reset to read mode either, the disk was well and truly destroyed - even if the other drive was used. There are other quirks in DOS1.2 but this was probably the one which had the greatest potential for catastrophy.

To write a single byte to disk, the FDC uses the similar logic as for a read. That is, it waits for the READY flag to indicate that the byte that is currently in $80 has been taken out and is being written, and then places the new byte into $80. The READY flag must then be reset:

```
FF79  BIT  $4D     ;is READY?
FF7B  BPL  $FF79   ;loop until it is
FF7D  STX  $80     ;send next byte to disk
FF7F  BIT  $41     ;clear READY flag
```

Before writing data, the SYNC pulses must be written - at least ten consecutive 1's must be written and this is done by, first selecting WRITE SYNC mode, writing three SYNC bytes, restoring the normal WRITE mode and then writing the identifier byte:

```
FDFD  LDA #$10
FDFF  STA  $4E     ;disable the ERROR flag during write
.
.
.
FE11  LDA #$DE
FE13  STA  $4C     ;select WRITE SYNC mode
FE15  LDA #$DC     ;get ready to select normal WRITE mode
FE17  LDX #$FF     ;X=SYNC byte

FE19  JSR  $FF79   ;write one
FE1C  JSR  $FF79   ;and a second
FE1F  JSR  $FF79   ;and just to be sure, a third

FE22  BIT  $4D     ;is READY?
FE24  BPL  $FE22   ;wait for last SYNC byte to be taken
FE26  BIT  $41     ;reset READY

FE28  STA  $4C     ;select normal WRITE mode
FE2A  LDA #$07
FE2C  STA  $80     ;send identifier byte
.
.
.
FE32  BIT  $4D     ;is READY?
FE34  BPL  $FE32   ;loop until it is
FE36  BIT  $41     ;reset READY
```

Now the data bytes can be written as required and the WRITE mode is turned off immediately the data is written.

None of this actually explains how the data is actually organised on each track, which is something which I will cover next time.

I will also include a RAM usage list and details of the ROM routines together with a brief mention of how the Interface processor actually communicates with the FDC.

Finally, some time ago (page 121, Sept 1981) I described a technique for trying to recover from a 23 READ ERROR. At that time I said that I'd give details of how to recover from a checksum error in the BAM since an error in this block will prevent the disk from initialising and so result in ID mismatches when trying to recover the corrupted blocks.

The easiest way to do this is simply to format a new disk using the same name and ID as the corrupted disk and then initialise this. Then replace it in the same drive by the corrupted disk - the DOS still thinks that its got the same disk, which is now initialised.

Perform the U1 and U2 commands on track 18, sector 0 and this will correct the checksum error. Of course, there are likely to be corruptions on this block and so a sensible move would be to copy all files across to a fresh disk.

If there are checksum errors in the directory, it may be very difficult to copy all the files, but at least most of the disk should be recoverable.

Be careful not to write new files to a corrupted disk, especially if the BAM has been corrupted since it is possible that the write could erase some of the data already on the disk.

I have been asked if it is possible to recover from a 27 READ ERROR (checksum error in header). The answer is NO. The header is used to identify the block about to be read and is put on the disk by the initial formatting and stays there for the life of the disk - it is not possible to read it back and then re-write it. I can envisage techniques for recovering from this sort of error, but it would require some sophisticated machine code programming within the disk unit itself.

--ooo--

COMMODORE'S LATEST MACHINES

By J.I. Meardon

The first reviews of the Commodore 64 have now appeared (Popular Computing Weekly 2.8.82, Personal Computing Today - Sept. '82, and Your Computer - Oct. '82, and Computing Today - Nov. '82), as well as advertisements from Commodore and at least one dealer (Personal Computer World - Oct. '82 and Daily Telegraph 5.10.82). None of the reviews can be regarded as in-depth evaluations of the Commodore 64. They are, however, worth reading as general overviews of the 64's capabilities.

With the present highly competitive and volatile market situation, company plans can change almost overnight. However, at the moment it seems that the existing 4000 & 8000 CBMs, together with the Vic-20, will be joined by four new machines, or series of machines, - the Max, the Commodore 64, the 500-series, and the 700-series. Eventually the 4000 & 8000-series will probably disappear, but no cut-off date has been announced. The future of the Vic-20 remains an unknown quantity, but the recent launch of VICSOFT suggests that the machine will be with us for some time yet.

The new machines have certain features in common:

* 40 x 25 character screen display (700-series 80 x 25)
* 320 x 200 pixel hi-res graphics
* Full 16-colour capability (not 700-series)
* KERNAL jump table
* Identical graphics, games, and sound facilities - a consequence of using the same support chips for the central microprocessor.

The Common Chip Set.

Apart from the 700-series, where the 6567 is replaced by a special CRT control chip, all the machines have the new 6567 video chip and the 6581 Sound Interface Device (the so-called SID chip), together with the 6526 Complex Interface Adapter (CIA) - this latter replacing the well-known 6522 found in existing machines. An additional interface chip, the 6525, is found in the '64 and machines of the 500 & 700-series, supplemented on the last two by a 6551 RS232 controller.

The main features of the 6567 and the 6581 have already been discussed in the September issue of this Newsletter (pp 232 - 233) by Mike Todd.

The Commodore Max.

This is the smallest of the new machines. Originally it was called the Ultimax or Vic-10. It is essentially a games machine with conversion capability to a full colour 40-column microcomputer or a music synthesizer. The CPU chip fitted to the Max is the 6510. This chip is 6502-compatible, retaining the same machine codes but having certain additional features.

To convert the Max to a micro computer, a plug-in BASIC cartridge, which also provides 2.5K of RAM is needed. This cartridge, like all such cartridges for the Max, is fitted with an edge-connector whose pitch is smaller than that of the Vic-20 plug-in modules.

The Max BASIC appears to be a version of BASIC 2.0 without the facility for dimensional arrays and trig functions. When used as a computer, audible feedback of key depression is provided by a 'keyboard' sound generated by BASIC. Depression of the character generator is accompanied by a bell-like sound.

The keyboard of the Max is of the flexible membrane type; printed overlays being available for games, music, etc. Layout of the keyboard is standard QWERTY with no separate numeric pad.

A cassette port is provided which will interface to the standard CBM cassette player. Facilities are also provided for the connection of joysticks, paddles, etc.

The Commodore 64.

Using the same chip for the CPU as the Max, the Commodore 64 comes with 64K of on-board RAM. However, only 38K of this is accessible from BASIC, or 54K from machine

code. The 20K ROM comprises 8K BASIC + 8K KERNAL (both identical to that found in the Vic-20) plus a 4K character generator.

The graphics, games and sound capabilities are exactly the same as on the Max, but the keyboard is that of a Vic-20. I/O ports include: cassette interface; a serial interface; 8-bit user port and memory expansion/cartridge port, together with facilities for the connection of joysticks, paddles, etc.

New features are the ability to accept a second processor and a claim that the memory map can be reorganised to allow software written for other Commodore machines with 40 columns to be run.

The Commodore 500 & 700 Series.

Once again the same graphics games and sound facilities are found on these machines as on the '64 and the Max. The CPU, however, is a 6509. This gives the possibility of memory expansion in excess of 750 Kilobytes.

Ports provided include IEEE-488 (enabling existing Commodore peripherals to be used), RS232C (for specialised printers and communications applications), an 8-bit user port (for scientific applications), and a cassette port which doubles for tape interfacing and security devices. A second processor slot opens up the possibility of operating the machines under other systems such as CP/M, whilst a cartridge slot is provided to allow ROM-based software to be used.

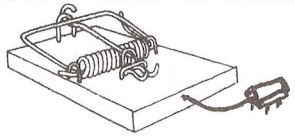
Although differing slightly in physical appearance, the keyboards of both series of machines have a full QWERTY layout plus 10 programmable function keys (accessible from BASIC or machine code), the usual four cursor control keys, and a separate calculator key-pad. On the 700-series, the keyboard can be detached from the main housing.

Machines in the 500-series need a separate monitor (or TV) whilst the 700-series of machines have an integral monochrome monitor (P39 phosphor) with long retentivity. This monitor, which can be tilted and swivelled, is fitted

with an anti-glare screen. To control the monitor a 6845 chip is used, this chip is the same as the one found in the 8032, but the built-in facilities for hi-res graphics may be utilized.

On-board RAM varies from 64K (500-series only) to 128K & 256K.

On the 700-series twin disk drive units are built into the main processor cabinet beneath the screen. These drives are linked directly to the main p.c.b. using DMA (Direct Memory Access) which results in enhanced disk access speeds. The DMA facility may also be used for fast access to large capacity storage devices - such as Winchester disk drives.



New Peripherals.

The new peripherals are unashamedly up-market and are priced accordingly.

Disk drives being added to the existing range are the 8250 (a dual drive floppy disk unit), the 9060 (a 5Mb hard disk drive) and the 9090 (a 7.5Mb hard disk drive). All three new drives are, like the existing drives, intelligent IEEE-488 devices.

The printer range is being enhanced by the addition of the 8300P (a daisy wheel printer based on the Diablo 630).

The 8250 drive, which looks both internally and externally very similar to the 8050 Micropolis drive, offers twice the capacity of the 8050 on a double-sided, double-density 5-1/4 inch diskette. The diskette is arranged with 77-tracks per side, switching between sides being transparent to the user. The drive is 8050 compatible - the initial illegal track and sector message generated when using 8050 disks should be ignored - but relative record files cannot be directly interchanged without the use of a special program. Some of its features are the same as the 8050 (e.g. max directory entry 224, bytes/sector 256). The diskette has an unformatted capacity of 1Mb per drive (950Kb after formatting).

The 9060/9090 drives use Tandon hard disk mechanisms (9060 - 2 platters, 4 heads; 9090 - 3 platters, 6 heads) and recognise all standard BASIC 2.0 and 4.0 commands. They will run all existing software which utilizes only one drive. Both drives dynamically allocate the block availability map and the directory areas.

The 9060 has an unformatted capacity of 6.38Mb (5.01 after formatting), whilst the 9090 has capacities of 9.57Mb (unformatted) and 7.52Mb (formatted).

The new printer (8300P) is a letter quality bi-directional daisy wheel printer designed for use with word-processors. It is fitted with a 320-character buffer. Operating at a print speed of 40 characters per second, print wheels (metal or plastic) are available for pitches of 10, 12 & 15 characters per inch - the latter offering the possibility of proportional spacing. Friction paper feed is standard, but tractor feed is available as an alternative if required. A single-sheet feeder is scheduled for later development. Unlike the Diablo 630, the 8300P is converted to use the IEEE-488 interface (the Diablo normally uses the RS232C) and cannot auto-emphasize or underline, except under software control.

--o0o--

HELP REQUIRED

Are you a 9000-series user ? Help is needed in getting a copy of a Waterloo tutorial disk to run some APL program files. Chapter 9 of the Micro-APL manual states "There are four types of files APL sequential, BARE sequential, relative and program.". There then follows a definition of the first three, which is fine, the "A program file is a special kind of BARE-sequential file normally used to contain programs and workspaces. As such, it is not usually accessed with the techniques discussed in this chapter ! I am having difficulty finding out where program file access routines are discussed. Any information would be welcome, contact A. W. Fisken, 23, Warwick Road, Altrincham, Cheshire, WA15 9NP. Tel: 061-928 4958.

--o0o--

REALISATION OF SWITCHING FUNCTIONS USING MULTIPLEXERS

Here is an interesting program for electronics buffs. You can easily multiplex many switching functions that would otherwise require several gates and associated wiring. To operate such a multiplexer, though, you must know how to configure it to obtain each desired output. The BASIC program meets this need, accepting the minterms from a KARNAUGH MAP and printing the data input configuration required for the desired function.

The program assumes that logic variable A is the data input; it labels the remainder as B,C,D,E.....H for address line selection. Suppose for example that the function required has four inputs and is defined as :

$f(A,B,C,D)=ABCD+\overline{B}\overline{C}+\overline{A}\overline{C}\overline{D}+\overline{A}\overline{B}\overline{D}$

First of all map the function:

CD \ AB	00	01	11	10
00	1	1	0	1
01	1	0	0	1
11	1	0	1	0
10	0	0	0	0

Figure One.

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Figure Two.

(For the non-technical: A Karnaugh Map shows the switch configuration for each desired output, and is an aid to producing a minimal logic circuit). Now you can enter the minterms using the Sigma convention (figure 2 shows numbering for 2 x 2 map).

The Program:

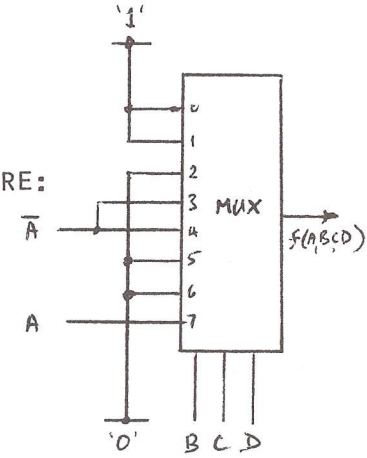
```
 100 REM
 110 REM * MULTIPLEXER REALISATION OF SWITCHING FUNCTION *
 120 REM
 130 REM
```

```
140 DIM S(65),I(65)
150 A$="ABCDEFGH"
160 PRINT"<clr><2dn>REALISATION OF SWITCHING FUNCTIONS
    WITH MULTIPLEXERS "
170 PRINT"HOW MANY INPUT VARIABLES ARE THERE : ";
180 INPUT N
190 IF N<2 THEN 170
200 IF N>6 THEN 170
210 PRINT"ENTER THE FUNCTION AS THE MINTERMS"
220 PRINT"ASSOCIATED WITH THE SUM OF PRODUCTS"
225 PRINT"FORM OF THE FUNCTION"
230 PRINT"ENTER -1 WHEN FINISHED"
240 REM
250 REM INITIALISE VECTOR S TO 0
260 REM
270 FOR K =1 TO 65:S(K)=0:NEXT
280 REM
290 REM PLACE MINTERMS IN VECTOR S
300 REM
310 FOR K=1 TO 2↑N
320 INPUT "MINTERM ";B
330 REM
340 IF B<0 THEN 400
350 IF B>(2↑N-1) THEN PRINT"ERROR NO SUCH MINTERM ! ":
    GOTO 320
360 REM
370 S(B)=1
380 NEXT
390 REM
400 REM DETERMINE MUX INPUTS
410 REM
420 FOR K=0 TO 2↑(N-1)-1
430 IF S(K)= 1 THEN 490
440 REM
450 REM TO DECODE I(K)
460 REM
470 I(K)=0:IF S(2↑(N-1)+K)=1 THEN I(K)=2
480 GOTO 500
490 I(K)=3:IF S(2↑(N-1)+K)=1 THEN I(K)=1
500 NEXT
510 REM
```

```
520 REM OUTPUT RESULTS
530 REM
540 PRINT"THE DATA SELECTOR INPUTS TO THE MUX ARE:"
550 FOR K=2 TO N:PRINTMID$(A$,K,1),
560 NEXT:PRINT
580 PRINT"MUX DATA INPUTS ARE :"
590 FOR K=0 TO 2↑(N-1)-1
600 PRINT"I(";K;")=";
670 IF I(K)=0 THEN PRINT"GROUND"
680 IF I(K)=1 THEN PRINT"LOGIC 1
690 IF I(K)=2 THEN PRINT"A"
700 IF I(K)=3 THEN PRINT"A NOT"
710 NEXT
720 END
```

Sample Run for Map in figure One.

```
 REALISATION OF SWITCHING FUNCTIONS WITH MULTIPLEXERS
 HOW MANY INPUT VARIABLES ARE THERE : ?4
 ENTER THE FUNCTION AS THE MINTERMS
 ASSOCIATED WITH THE SUM OF PRODUCTS
 FORM OF THE FUNCTION
 ENTER -1 WHEN FINISHED
 MINTERM ? 0
 MINTERM ? 1
 MINTERM ? 3
 MINTERM ? 4
 MINTERM ? 8
 MINTERM ? 9
 MINTERM ? 15
 MINTERM ? -1
 THE DATA SELECTOR INPUTS TO THE MUX ARE:
 B C D MUX DATA INPUTS ARE :
 I( 0 )=LOGIC 1
 I( 1 )=LOGIC 1
 I( 2 )=GROUND
 I( 3 )=A NOT
 I( 4 )=A NOT
 I( 5 )=GROUND
 I( 6 )=GROUND
 I( 7 )=A
```



Circuit diagram of multiplexer connections.

TECHNICAL TIPS

Disk BUG - 2031 Drive.

A fault when using BASIC 4.0 DIRECTORY or CATALOG commands in conjunction with the 2031 single disk drive has been reported by Commodore. A corrupted directory listing will occasionally occur. If this happens take another directory listing to ensure data integrity. An alternative method is to use the 'wedge' command >$0 or BASIC 2.0's LOAD"$0"<ret> LIST<ret>, as these appear to be OK!

BASIC 2.0 and 2031 drives.

There is an error in BASIC 2.0's IEEE handshaking routine, detailed in CBM PROFESSIONAL COMPUTER GUIDE (a successor to PET/CBM PERSONAL COMPUTER GUIDE) page 362, this states that the only difficulty will be with non-Commodore devices. Unfortunately it affects the 2031 drive; occasionally characters are lost during GET# and INPUT# commands, ST gets screwed up. There are two ways to fix this bug:

1. Upgrade to BASIC 4.0; this is OK!
2. Change the offending ROM;

ROMs are available from Skyles Electric Works including 28-pin (i.e. small keyboard PET) - A UK source may be SUPERSOFT of 10-14, Canning Road, Wealdstone, Harrow, HA3 7SJ.

Bad news for 2031 owners.

The 2031 may appear to be a single 4040 drive with all the features of the 4040, indeed the manual may lead you to believe this. Unfortunately the 2031 is a 4040 'sawn in half'. It has half the RAM of a 4040 - therefore half the channel buffers, which is a restriction on the number of files that can be opened on the disk. For instance a 4040 can easily have two Relative Files opened at once; the 2031 will struggle. The maximum number of sequential files open at once is similarly halved. This will probably not affect

many 2031 owners but you should be aware that not all PET 4040/using-one-drive programs will work on the 2031.

More on 2031 & 1540 (Vic drive).

Both the 2031 and 1540 drives issue an IEEE reset signal when switched on thus resetting any other device (except the computer) on the bus. It is recommended that you turn on the computer first, then the disk drive then any other peripherals. If you wish to switch on or off the drives whilst other devices are connected then disconnect disks first. This will ensure that any softcoded secondary addresses, etc., are still valid.

PS for 2031 or 1540 owners.

The save-with-replace bug on 2040, 3040, 4040 drives has never been cured, it is liable to be present on the 1540 and 2031 drives as well. Never use the save-with-replace (e.g. SAVE"@0:PROGRAM",8) as it is possible to corrupt data and pointers on the disk. You should always SCRATCH the old programs first then use a resave.

SUPERPETS.

Many of the SuperPETs/9000-series (this is the 8032 with access to a 6809 processor at the flip of a switch) sold in the UK have not been fitted with a retrofit kit to enable the 6502 to access ROM slots UD11 and UD12. To check your SUPERPET look at the number of switches on the lower right hand side of the case. If you have four then everything is all right. If there are only two then contact your dealer who should arrange to have the kit installed.

Did you know that the SUPERPET has the best implementation of APL available on any 8-bit micro ? (neither did I; but John Collins of Commodore's Technical Department assures us that it has). To support the use of the SuperPET in APL a communication software package, written in APL and 6809 assembler, called KEYCOMM should now be available from your dealer. KEYCOMM enables the

transfer of APL programs between the SuperPET and a mainframe (in either direction). These programs can be saved to disk for future running. Alternatively one may develop programs on the SuperPET before running on the mainframe.

Are you confused about ROM/RAM in the SuperPET/8096 machines; I know I am.

There is an extra 64K of RAM in both of these beasts, each of which can run standard 8032 software BUT NOT one another's special software. The 8096 has four banks, each of 16K, two being alternates from $8000 to $BFFF and the other two being alternates from $C000 to $FFFF. I wonder how long it will take some company to produce a 64 emulator for the 8096!! The SuperPET has sixteen banks each of 4K, which are alternates from $9000 to $9FFF - with only one bank being operative at a time. Waterloo languages are loaded in these 4K blocks.

8096 STUFF.

OK so you've got that wonderful big 96K PET, you've got lots of nice machine code software to run in the 96K but can you access it from BASIC ? The answer is usually NO unless you are a machine code wizard. Now there is a product to help !

LOS which stands for Loadable Operating System runs on an 8096 and provides 32K for BASIC programs and 32K for variable storage. LOS is also loaded into RAM. In addition to this larger memory space for BASIC, LOS also adds PRINT USING, ON/ERROR/GOTO, an improved LIST command and TOOLKIT-like commands such as AUTO.

Chaining of programs is supported (as if 96K of RAM were not enough!!). LOS has an RRP of £ 100.00 and really expands BASIC's horizons.

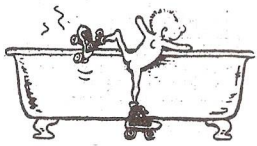
T.C.

--o0o--

SOFTWARE HEADERS

The following text gives the standard header information for ICPUG software:

```
1 REM"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
2 REM"PROG INFO   - DIRECT PRINT
3 REM"VERSION     - 2.4
4 REM"TYPE        - UTILITY
5 REM"SOURCE      - UNKNOWN
6 REM"AUTHOR      - UNKNOWN
7 REM"DATE WRIT.  - ?
8 REM"AMEND.      - NONE    [DEFAULT]
9 REM"MACHINE REQ - ANY PET [DEFAULT]
10 REM"SIZE       - SMALL   [DEFAULT]
11 REM"PERIPHERALS- DISK
12 REM"LANGUAGE   - BASIC   [DEFAULT]
13 REM"LOAD ADDRS - 1024    [DEFAULT]
14 REM"STATUS     - PUBLIC  [DEFAULT]
15 REM"DESCRIPTION-
16 REM" READS DISK DIRECTORIES
17 REM" AND PRINTS THEM ON A PRINTER
18 REM"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
19 REM"LETTER A OPTIONS FOR MACHINE :-
20 REM" 2 - 2000 MACHINE ( BASIC 1 )
21 REM" 3 - 3000 MACHINE ( BASIC 2 )
22 REM" 4 - 4000 MACHINE ( BASIC 4 )
23 REM" 8 - 8000 MACHINE ( BASIC 4 )
24 REM" A - 3000 MACHINE UPWARDS
25 REM" B - 3000 & 4000 40 COLS ONLY
26 REM" C - 4000 SERIES UPWARDS
27 REM" U - UNIVERSAL
28 REM"LETTER B OPTIONS FOR STORAGE :-
29 REM" C - CASSETTE NEEDED
30 REM" D - DISK NEEDED
31 REM" 0 - NOTHING REQUIRED
32 REM"LETTER C P = PRINTER REQUIRED
33 REM"         0 = NO PRINTER REQUIRED
34 REM"LETTER D D = DOCUMENTATION AVAILABLE
35 REM"         0 = NO DOCUMENTATION
36 REM DIRECTORY PRINT V-2.4 13-08-81 ARM
```



--o0o--

17) LOOP is a new structure introduced on 1.02 and the COMAL board. A set of instructions enclosed by LOOP and ENDLOOP will execute continuously. Obviously there must be a way out of the loop. This is by an EXIT command (or possibly EXITIF command, as this may be implemented instead in the final version).

```
LOOP
 A$=KEY$
 IF A$="A" THEN EXIT (or EXITIF A$="A")
ENDLOOP
```

will wait until 'A' is pressed.

18) LIST will support the option:

```
LIST PROCNAME
```

this will only list the procedure PROCNAME. Pressing the 'space' key will pause listing until pressed again. If LISTing to a file, version 1.02 will expect a drive number as part of the file name. These options are not supported by 1.01. LISTed files in 1.01,1.02 and the board COMAL will be sequential files rather than PRG files as in 0.11.

19) NULL is as for 0.12 described last Newsletter.

20) OBJLOAD "FILENAME" will load the file FILENAME into top of memory and lower COMAL pointers accordingly. One can load from tape by adding a unit number (,1).

21) OPTION 9*4096 will recognise the COMAL-compatible machine code procedures written in ROM starting at $9000. OPTION can be used to access any start address, so machine code procedures can be written, loaded with OBJLOAD and recognised via OPTION. Unfortunately I have not seen any instructions on how to implement machine code procedures to use with this powerful facility.

22) PRINT USING is fully supported by these COMAL versions.

23) RENUM is extended:

```
RENUM 500; 1000,5
```

will renumber lines from line 500 to the end of the program starting line 1000 in steps of 5. This extension is not implemented in 1.01.

9) ESC was described for 0.12 last Newsletter.

10) EXIT is described under the LOOP command. It is not supported by 1.01.

11) FUNC was described for 0.12 last Newsletter. It is not supported by 1.01.

12) GET$(1,6) will get 6 characters from the logical file number 1 which has been previously opened for READ. One can GET characters from the keyboard by using a file number of 0 (which need not be opened). Note that at least one character must be returned. Use KEY$(q.v.) if no key press is a possibility. I am unclear as to whether this command is supported by 1.01. GET$ certainly exists in 1.01 but it may act like KEY$.

13) GLOBAL is provided in 1.01 only and allows a CLOSED procedure to use variables from outside. To do this they are defined as GLOBAL inside the procedure.

```
PROC TEST CLOSED
GLOBAL VAR1,VAR2
.....
```

will allow the procedure to use VAR1 and VAR2 which are defined outside the procedure.

14) IMPORT is the equivalent of GLOBAL in 1.02 and the COMAL board.

15) INTERRUPT PROCNAME will execute the procedure PROCNAME if a signal is sensed on the SRQ line of the IEEE bus.

16) KEY$ will return the last key typed. If no key is pressed CHR$(0) will result, NOT the null string. This is not supported by 1.01 (unless GET$ is the equivalent).

COMAL FOR THE 8096

In this article I want to talk about the additional COMAL commands available in the 8096 versions of COMAL. Unless stated otherwise they apply to versions 1.01, 1.02, and the COMAL board version. Version 1.02 represents the full COMAL implementation.

1) It is now possible to say PRINT AT 5,16: SALARY
This will print the value of SALARY starting at row 5, column 16. This can also be combined with USING as e.g.
```
PRINT AT 5,16: USING "####.##": SALARY
```
The AT command is not available in 1.01.

2) BASIC is identical to 0.12 in that a warm reset is carried out. It assumes a BASIC4 machine so would not be valid with a COMAL board in a new ROM PET.

3) The CAT command has been extended to a full specification as follows:

```
CAT [<drive>][,<unit>] [<pattern>]
```

An example is CAT 0,9 "COM*" which will display all files beginning COM from drive 0 of disk unit 9. This extension is not available in 1.01. Hitting the space bar will pause the listing.

4) CURSOR row,col moves the cursor to the given row and column.

5) DEBUG will jump to monitor, but the routine has a bug in it so you are not advised to use it.

6) DELETE "0:FILENAME" will scratch the file FILENAME from drive 0. The drive number must be given for the command to work. This command is also supported by 0.12.

7) ENDFUNC was described for 0.12 last Newsletter. It is not available in 1.01.

8) ENDLOOP is described under the LOOP command. It is not supported by 1.01.

24) RESTORE LABEL will restore the DATA pointer to the first DATA item after the line labelled LABEL.

25) RETURN is as defined for 0.12 in the last Newsletter.

26) SAFE will make a program unlistable. I am not sure if this is implemented in 1.02. In my opinion it shouldn't be implemented at all !

27) LOAD, ENTER, LIST, SAVE to tape are supported by these COMAL versions.

28) SETEXEC is as described for 0.12 last Newsletter.

29) SETTIME 123 will set the jiffy clock to 123. This is only available on 1.01.

30) SPC$ is supported by these COMAL versions and acts like the BASIC equivalent.

31) STATUS$ is similar to DS$ in BASIC4. In version 0.11 or 0.12 STATUS acts like PRINT STATUS$.

32) STOP "MESSAGE" will print the text 'MESSAGE' when the STOP command is executed. This is not supported by 1.01.

33) STR$, the BASIC function, is supported by these COMAL versions.

34) TIME holds the value of the jiffy clock. The 1.01 SETTIME is mimicked in other versions by TIME 123.

35) TO can be used for readability in the LIST command on 1.02 or the COMAL board:

```
LIST 10-266 TO "FILENAME" is identical to
LIST 10-266,"FILENAME"
```

36) TRAP is as described for 0.12 last Newsletter.

37) VAL and VERIFY are exactly as the BASIC equivalents.

38) READ FILE, WRITE FILE, INPUT and PRINT to random access files are supported.

39) COMAL uses file numbers 1 and 255 for its own use (1 for disk, 255 for printer) so do NOT use these numbers in your programs.

As you can see from the above, these versions of COMAL introduce all the missing items from the earlier software versions. They also provide some very useful extensions and further enhance the suitability of COMAL over BASIC. While the COMAL board is not cheap it certainly will provide a much better programming language than the original BASIC while still allowing BASIC to be used when required. By the time you read this I should have version 1.02 for 8096 computers available on 4040 or 8050 disks. Send a disk and return postage to my usual address if you want a copy - 73, Minehead Way, Stevenage, Herts. SG1 2HZ.

Brian Grainger.

--oOo--

PI WITH THE BUSINESS KEYBOARD

Users of machines with the business keyboard, as on 8032 models, may find the lack of the pi-function a disadvantage. Actually the function isn't missing, it's just that the symbol does not appear on the business keyboard.

The following is the technique that I use to develop programs that are to contain pi: first clear the screen, then enter:

```
    <dn>POKE32788,94          (the POKE code for pi)
  <hcsr>0 REM <cr>            (enter line 0)
```

now line 0 of the program contains pi, so that when pi is required in a line simply LISTO and screen edit the line to, for example, 150 Y = R * SIN(π * X)

At the end of program development line 0 could of course be deleted. Finally pi can be produced in output by PRINT CHR$(255).

R.D.G.

--oOo--

SHOP WINDOW

Vic-20 users struggling to find a cheap printer may care to consider the Amber 2400 model aimed at the home computer market. It features a range of serial and parallel input capabilities as standard to interface with as many different computer makes as possible. The printer is unusual in having four print solenoids, oscillating from side to side, and each covering a quarter of the paper width. The paper then advances one dot height to form the next part of the character. The printer can handle continuous graphics and costs £ 69.95. Full details are available from Amber Controls Ltd., Andover, Hants.

The Tandem expandable expansion system for the Vic-20 costs £ 34.95 and gives 1 + 3 expansion slots for Vic cartidges. No additional power supply is required. Contact Stonechip Electronics, Unit4, Hoskins Place, Watchetts Road, Camberley, Surrey. Tel: (0276) 681131.

In the early PET days, a high-resolution graphics package was produced by IJJ Design Ltd., and is still going strong. Currently versions are produced for 3-, 4- and 8000-series models, including 8096. Shortly to be produced is a pixel-resolution light-pen. Sales and marketing are now handled by PMS (Instruments) Ltd., 107/109, King Street, Maidenhead, Berks, SL6 1DP. Tel: (0628) 76688.

The Personnel Manager is a software package to assist in the day-to-day management of a personnel department. The system features multi-level password protection from access and can report on head count, turnover, recruitment and salary analysis. The standard system (8096, 8023 and 8050) including hardware, software, communications, desk and sundries comes to £ 6250 + VAT from Missing Link Computers Ltd., Abacus House, 53-55, Ballards Lane, London, N3. Tel: 01-349 4711. Various options are also available.

Also available from the above is a Replacement Window System to handle ordering, material utilisation, pricing, et al.

CP/Maker is a circuit board add-on and CP/M software add-on which increases RAM to 96K on 3-, 4- and 8000-series machines, adds a Z-80A microprocessor that runs concurrently with the 6502, can emulate the Hazeltine 1500 VDU, among other features. Price is £497. UK distributors are Tamsys Ltd., 12a, Sheet Street, Windor, Berks, SL4 1BG. Tel: Windsor 56747.

--o0o--

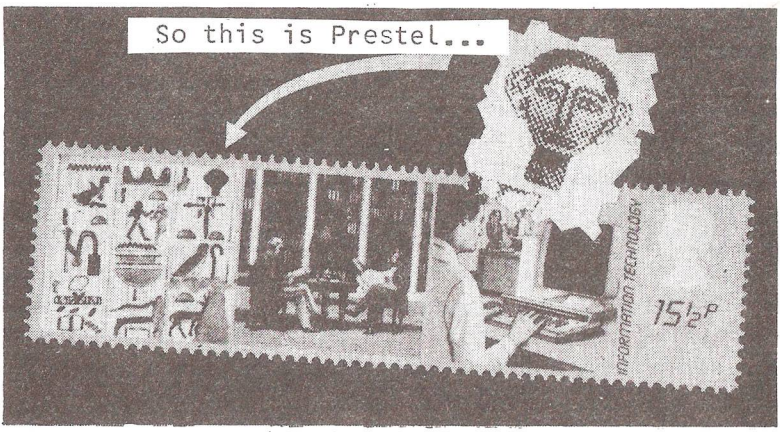
OFFICERS ELECTED AT THE AGM.

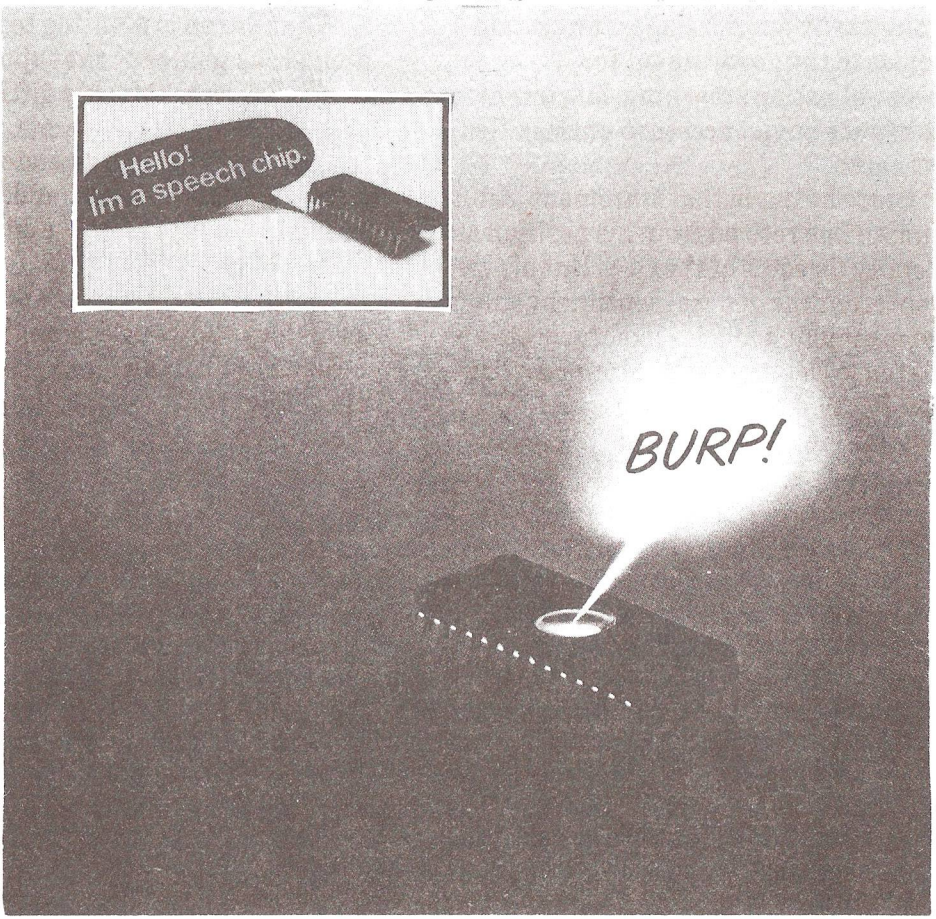
The following officers were elected at the A.G.M.:

Chairman:	Mick Ryan.
Treasurer:	Joseph Gabbott.
Membership Secretary:	Jack Cohen.
Regional Co-ordinator:	Terry Devereux.
Editor:	Ron Geere.
Vic Co-ordinator:	Mike Todd.
Publicity Officer:	David Annal.
Assistant Publicity Officer:	Rod Eva.
Software Librarian:	Bob Wood.
Software Organiser:	Carl Millin.
Discounts Officer:	John Bickerstaff.
Minutes Secretary:	Alan Birks.
Hardware Projects:	Fred Offler.
Software Advisers:	Harry Broomhall, Tom Cranstoun, and Ray West.
Comal Representative:	Brian Grainger.
Prestel Representative:	Stephen Rabagliati.
Exhibition Co-ordinator:	Stephen Rabagliati.
Assistant Editor:	Tom Cranstoun.
Technical Queries Secretary:	Jim Tierney.

The editor wishes to acknowledge the assistance proffered by many 'behind the scenes' members who do not hold an official position, but without whose help many tasks would remain uncompleted.

--o0o--


So this is Prestel...
INFORMATION TECHNOLOGY
15½P


Hello!
I'm a speech chip.
BURP!

SOME READERS PROBLEMS

By Mike Todd.

I receive a huge number of letters from members (at one time I got over 45 in one week!) asking for advice and help with problems they're having, and as far as possible I try to reply to them all personally. Unfortunately, with all the work that I am doing I have acquired a significant backlog and to those who have not received a reply I apologise.

It struck me that some of the problems that people are encountering, and that require some research on my part, may already have been solved by other members so I intend in this and the next issue to list some of these problems and if anyone has solved them, please let me know and I will forward your replies immediately to the originator of the query.

I considered giving contact names and addresses but this may not be fair on those who would rather not have their names and addresses published, but in future columns I will publish names and addresses with the hope that those with solutions can get in touch directly, unless of course I am instructed otherwise!

Although only an experiment, if you do want to have your problems aired in this way, don't hesitate to drop me a line and let me know if you're happy for your name and address to be published. It may be that this sort of column is very popular and could become a regular feature.

VIC - LOAD ERRORS

One member has had problems while typing programs from a magazine. He sensibly SAVEs it at various stages during the typing, rewinding it to the beginning each time, but has found that the program has become corrupted, sometimes beyond all recognition. This is something I've not come across before and it may be due to tape reading errors which have not been reported by the "LOAD ERROR" message.

Perhaps the simplest answer would be to VERIFY the tape every time it is SAVEd - I know it is time consuming, but not as time consuming as having to retype the whole program! Has anyone come across anything like this, or can offer a solution?

VIC - SUPER EXPANDER & PROGRAMMERS AID

There appears to be a problem reported by one member when using the Commodore Super Expander and Programmers Aid cartridges at the same time. If he uses the RJOY function on the Expander and then tries to FIND something with the Programmer's Aid, it always stops at the line containing RJOY and then hangs up.

He wants to know if there's anything can be done about this, other than removing all the RJOYs before using FIND.

I'm not in a position at the moment to try both these cartridges simultaneously but I suspect it may be due to the fact that RJOY is converted to a new, and perhaps as far as the Programmer's Aid is concerned, unrecognisable token - in fact the new token is 220 for RJOY. It may be this which upsets the FIND operation - or is it just that there is some sinister interaction between the two cartridges, perhaps through common memory usage, which is causing the problem?

VIC - ROBOTS & INTERFACING

One member wants to do some "robot-arm" and similar type controlling with the Vic - the easiest way would probably be the Vic-REL relay cartridge which has 6 relay contacts rated up to 24v at 10W. He would like to know a suitable source of 12volt motors and also some idea of how to use stepper motors, either with Vic-REL or some other way.

In fact, I'm sure that there are many who have attempted this sort of project before and could give a newcomer some sound advice and/or assistance.

4032 (12") and SIMPLY-WRITE

One member has the Simply-Write word processor which he uses with a FAT-40 and 4022 printer. Although he can set the programmable character on the 4022 to generate a pound sign, but he cannot find a way of accessing it from his program.

I suspect the problem is one of generating CHR$(254) (ASCII $FE) which is the printer's programmable character. Somehow, Simply-write's output routine needs intercepting and whenever some arbitrary character (perhaps "$" or "#") is printed, it is instead substituted by ASCII $FE.

If anyone has done this with the Simply-write wordprocessor, or is prepared to have a go, I'd like to hear from them.

8050 DISKS

Many will know of the bug in the 8050 disk drives (p211) which prevents a successful "COPY D1 TO D0" being executed. Commodore admit to this and suggest that the program "COPY DISK FILES" (available in the ICPUG software library) be used instead. I suspect that there is no obvious solution to this, but has anyone succeded in cracking it - perhaps with an explanation as to why it happens?

VIC (& OTHER COMMODORE PRODUCTS!)

I am a radio amateur and spend a lot of time listening around the short wave bands. I also use the Vic as a morse code and teleprinter decoder. Unfortunately, the interference generated by the Vic (also the 3032 and 3040 disk drive) often obliterates the bands even at a great distance from the receiver and aerial.

I would be very grateful to know if anyone has suffered the same - and better still, what their solution was.

--oOo--

COMMODORE COLUMN

Christmas comes but once a year is a saying which I am glad to say is true. The potential micro-buyer would probably wish it were a year-round event. Micros, in one shape or another, are the 'in' present this year and manufacturers are lining up tempting dishes. Goodies lined up from Commodore for this Christmas include: The release of the 64 (brought forward from the original date of January 1983 to quantity delivery in November '82) which can now be seen at your local dealer, but not yet bought by the general public. If you get one this year consider yourself lucky, Commodore expect to be sold out until beginning '83, judging by the response to their initial advertising.

Disk drives may find their way into your Christmas stocking; VIC-1540 drives (single, 4040-compatible drives) have been reduced in price by 100.00 pounds to 299.99 including VAT. This is a truly amazing price for an intelligent drive - the same drive will work on the 64 (with yet another DOS upgrade ROM to be fitted first). Commodore promise a 1541 drive which will work on both VIC-20 and 64. I wonder if this new price will be reflected in the 2031 PET/CBM single disk drive (currently £ 395+VAT) one can hope! As an aside, is anyone interested in a Serial VIC bus (sometimes, wrongly, called a SERIAL IEEE BUS) for the PET thus allowing us ageing PET owners to obtain 'cheaper' peripherals. If you are interested in this project, which will be software only, could you contact Tom Cranstoun. If you have already produced it donate a copy to the ICPUG library.

Commodore have told us that the VIC-1010 expansion board is now available at a price of £ 119.95 inc VAT. (Is this good news or bad!!). 38 new packages are planned for Vic-20, the first few now available; BBC MASTERMIND and QUIZMASTER games among them. Offerings for business users will follow next; software for General Practitioners & Civil engineers; Electronic spreadsheet program (something-CALC no doubt); word processing and database management. Watch this space for news as it arrives.

Commodore have launched a VICSOFT software club for Vic-20 owners. This is not really a Christmas ploy it is intended to be part of Vic for a long time. This is a mail order service intended to complement the big chain stores who are happy to sell low price computers but unable to offer the full software and peripheral range. Vic users who joined before the end of September received a dust/cover for their Vics (personally if you were that keen to join I don't see how your Vic will gather any dust). Moves are afoot to combine VICSOFT and ICPUG membership; more details next issue. Also see the Vic Column for further information.

Commodore have recently set up a speech technology division in the States which, it is hoped, will develop a variety of products based on voice input/output to be used with its existing range. Commodore aim to get 10% of the expected $500 million dollar speech market, the majority use is expected to be education, followed by entertainment.

US NEWS: Vic currently has 30% of home computer market in US, followed by Atari 27% and TI 17%. Average price of Vic-20 is $229, although TI have slashed yet another $100 off their TI99/4A, with a short term rebate scheme making this machine $239 dollars - a serious contender for Vic. Atari is offering up to $60 refund on software packages (at $10 a time) ATARI is not reducing the price of their machines as 'they have sold out this years stock'. It looks like a micro war in the states to match our BBC/Vic/Sinclair/Acorn on this side of the Atlantic.

The Commodore 500 is now in the UK! (only in the firm's Engineering Dept at the moment.) It is expected to be available in late October 82. The Commodore 700 is in an advanced state of pre-production testing.

Integral disk drives are on time (this means all delays will be Commodore's fault as they manufacture everything but disk drives). A launch is expected in November.

T.C.

--oOo--

DEBUGGING AND DOSSING AROUND

By Brian Grainger

Since writing my disk to tape SAVE routines and disk rename program I have had some letters pointing out they do not work! Sometimes the problems are simple, such as incorrect typing and using the programs on equipment for which they were not written! The answer to the first problem is simple. Always check the listing first. On the second problem I do try to specify with what hardware programs will work but maybe some clarification is needed with regard to disks.

I tend to specify disk formats in the form DOS1, 2A or 2C. I do this rather than DOS1, 2.1 or 2.5 because one can read the 1,2A or 2C by displaying the disk directory and noting the top right of the the first line. DOS1 was used on 2040 and 3040 disk units. DOS2A is used on 4040 disk units. DOS2C is used on 8050 disk units. I have no idea what is used on the single disk drive or the VIC disk so I give no guarantee that any of my programs will work with these units.

Despite the above notes there ARE bugs in the programs, mainly because I fell for the old trick of assuming because a standard manual command works in DOS2A it ought to work in the same way on DOS1. As I no longer have DOS1 I do have a problem in testing ! Anyway let's up the bugs.

First of all looking at Universal Disk Rename (Vol.4 No.3 P.107) line 240 should read

```
240 IFV$=""THENV$="NOTHING":GOTO260
```

Also line 230 should be deleted to conform with the Editors warning on disk ID changing. I now realise a problem exists and I had not seen it as the correct conditions for the problem had never occurred in use.

Secondly let us turn to disk to tape saving (Vol.4 No.4 P.204). Replace line 260 with

```
260 OPEN15,8,15,"IO":OPEN4,8,4,"#"+"0":Z$=CHR$(0)
```

Replace line 2330 with:

```
 2330 DO$=CHR$(A(165)+CHR$(A(166)):IFDO$=CHR$(32)+
      CHR$(160)THENDO$="1 "
 2340 RETURN
```

Now replace lines 2400-2410 with:

```
           2400 PRINT#15,"B-P";4;1
           2410 PRINT#15,"M-R"CHR$(0)CHR$(17)
           2420 GET#15,A$:A(0)=ASC(A$+Z$)
           2430 FORB=1TO255:GET#4,A$:GOSUB2000
           2440 A$=A$+Z$:A(B)=ASC(A$):NEXTB:RETURN
```

Finally one should replace line 2560 to be on the safe side (in other words I do not know why but it works if you do!).

```
     2560 NEXTB:IFS<>255THENT=18:GOTO2500
```

The above changes are because of a bug with DOS1 in the B-P command which has not been well publicised. Even Raeto West's book did not cover it. Apparently trying to move B-P to the 0 position and reading the 0th byte by GET# will only allow access if the last character pointer is NOT 255. A pity therefore that the U1 command which is used instead of B-R because of a bug forces the last character position to 255! The solution is to read the byte in positon 0 by a memory access command to the disk buffer.

I would now like to make a plea. I have seen the U1 command in various sources being used in the following way:

```
PRINT#15,"U1";CH;DR;TR;SE
PRINT#15,"U1",CH,DR,TR,SE
PRINT#15,"U1:"CHR$(CH)CHR$(DR)CHR$(TR)CHR$(SE)
PRINT#15,"U1:CH,DR,TR,SE"
```

These variations apply in a similar way to the block commands as well. Now under DOS2A all variations seem to work for the B-P command although I cannot think how the last two can be equivalent. Would somebody (Mike Todd?) please tell me what will work on the various DOSs and if possible give me a single command which will work on any DOS!

To finish this article on disk problems I would like to mention a problem when using the UNIT TO UNIT copying program that Commodore produced. It doesn't work when copying large (approx. 100 block) files. The solution is simple. Use Jim Butterfields COPY/ALL which should be available in the ICPUG library.

--o0o--

WRITING FOR THE NEWSLETTER

Contributors to the Newsletter can save the editor considerable typing time if items submitted for publication are in a form that can be read by the computer. If your item is more than say, half a page, send it on cassette (returnable) in the form of a program thus:

```
10 OPEN1,3
20 PRINT#1,"The text of your article here."
30 PRINT#1,"More text, etc. using UPPER and lower case"
40 CLOSE1
```

Enclose with your cassette a hard-copy listing of your program or text in case it fails to load. Keep a back-up copy yourself, and include your name and address on the cassette so that it can be returned to the rightful owner. Should you prefer to use a diskette, then unless the article is somewhat lengthy, 8050 formatted disks should not be used. Do ensure disks are adequately protected for transit through the post.

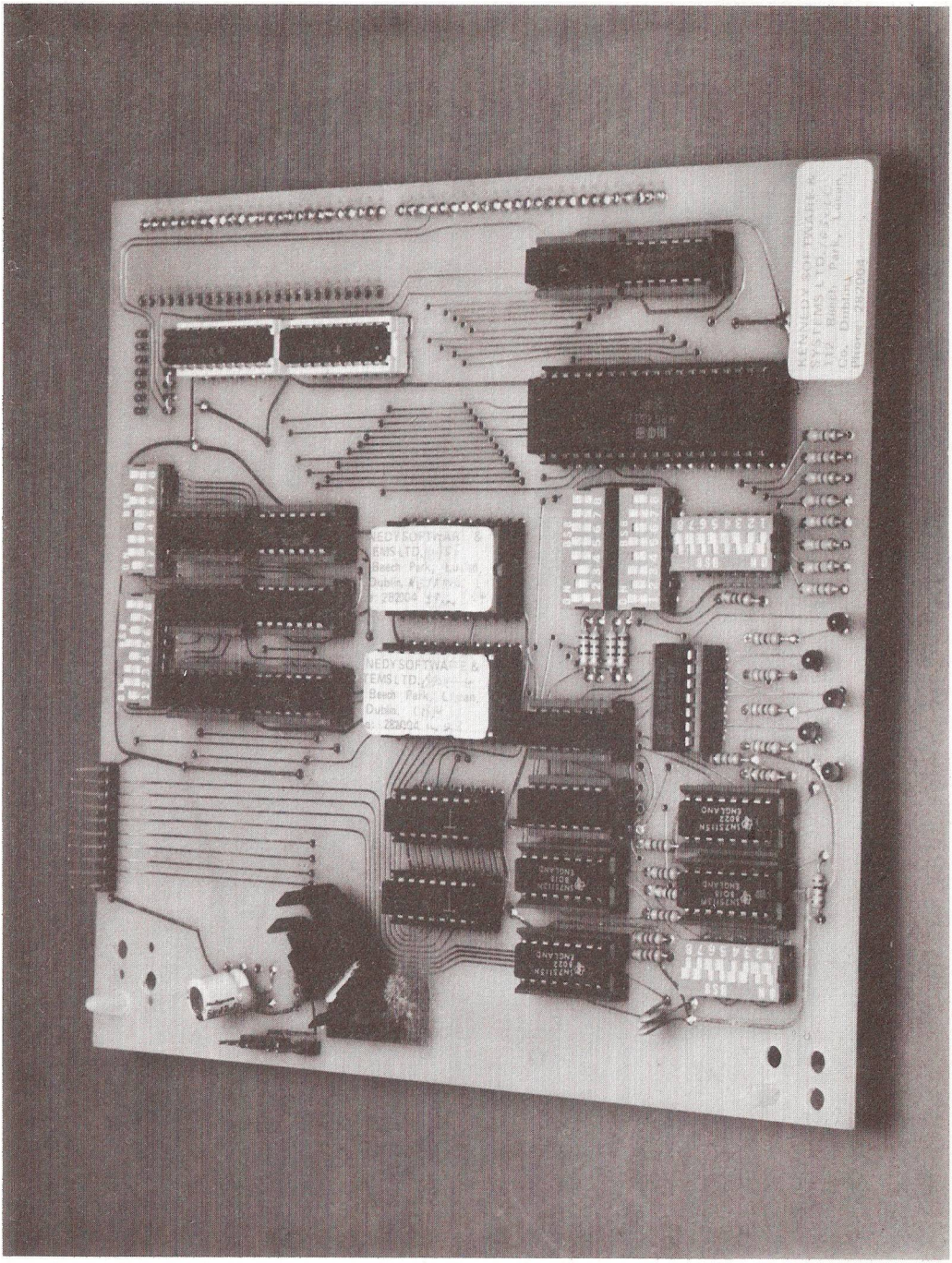
If you are writing using a word-processor, text can be accepted on Papermate, Superscript, Wordpro and Wordcraft formats, but to save much detective work, do please state which you have used. ASCII sequential files can also be accomodated.

Formatting should be kept to a minimum since it differs for each word processor and it will probably be changed anyway. Do not include cursor controls in ASCII text strings, they may stop your printer producing graphics, but they can produce havoc with ASCII printers !

Certain regular contributors have special arrangements and for them the above notes do not apply. But for all, please note the language BASIC is upper case (Pascal is not), flat round-shaped items are discs, but the floppy variety are, by convention, disks with a 'k'. Please use k= kilo = 1000, and 'K' for Kbyte = 1024. Note that the last date for machine-readable material is the 2nd week of even months, otherwise, the first week.

--o0o--

The Commodore 720 Computer



The Keynet Hardware

The stack 40/80 column card for the Commodore VIC-20

Printed and distributed by Richardson Printing Ltd., Unit 23, Colville Road Works, Colville Road, Oulton Broad, Lowestoft, Suffolk NR33 9QS. *Telephone: (0502) 67029*