$2.95
U.S. FUNDS

DECEMBER 1982

COMMANDER

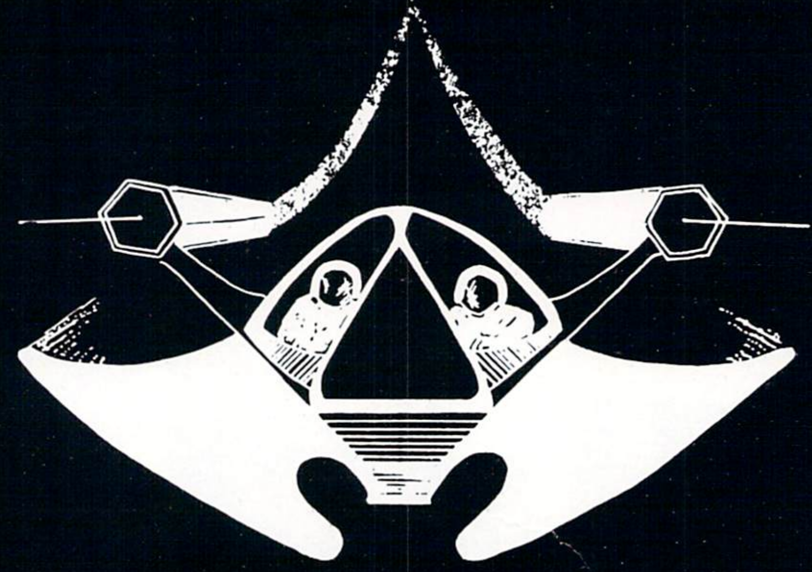
The Monthly Journal for Commodore Computer Users

Game Contest–Deadline Feb. 1

All About CB2 Sound

Peek & Poke

PREMIER ISSUE



First call for Clubs and Newsletters Directory

To be included in the first edition of the Commander Clubs and Newsletters Directory, your club or publication must supply the following information:

1. *name of organization or publication*
2. *mailing address*
3. *contact person and telephone number*
4. *name of newsletter or publication*
5. *special interests*

Send your information to Clubs and Newsletters Directory, Commander, P.O. Box 98827, Tacoma, Washington 98498.

Commander - The Monthly Journal for Commodore Computer Users is published monthly by Micro Systems Specialties, P.O. Box 98827, Tacoma, WA. 98498. Domestic Subscriptions, 12 issues, $22.00. Second Class Postage pending at Seattle, WA 98134 and additional mailing offices. Postmaster: Send address changes to: **Commander - The Monthly Journal for Commodore Computer Users, P.O. Box 98827, Tacoma, WA 98498.**

Table of Contents

STAFF

Publisher
THOMAS L. ROSENBAUM

Editor
ALICIA A. LINDEN

Assistant Editor
TERILYN M. FLOYD

Art Director
JANE RAMSEY

Typesetting
K.C. PRINTING

Consultant
EDWIN SUND

Printed By
GRANGE PRINTING

COMMANDER is published monthly by:
MICRO SYSTEMS SPECIALTIES, P.O. Box 98827, Tacoma, Washington 98498

Subscription Rates	*Per Year*
U.S.	$22.00
Canadian, Mexican	$26.00
Surface Rates	$37.00
Air Mail	$54.00

For back issues, subscriptions, change of address or other information, write to:
COMMANDER
P.O. Box 98827
Tacoma, Washington 98498
(206) 565-6816

Letters to the Editor

To the Editor of Commander Magazine:

Congratulations and thank you for taking the initiative to create a magazine for us the silent minority (soon to become the not so silent majority - I hope).

There are many good publications on the market today but it seems that they are affiliated with either some manufacturer or special interest group. Now, thanks to your magazine COMMANDER, we have a chance to have an unbiased medium for Commodore computer users.

I wish you the best of success in this new enterprise as your success will be to our advantage also.

EDWIN SUND
Tacoma, Washington

Dear Editor,

It's great the Commodore users have finally encouraged the generation of a magazine devoted to those of us who need accurate information.

While being a skeptic, I am sure in time we will see if our prayers have been answered.

I am hoping to see a full spectrum of articles and software for the full range of computers and perepheral devices available. As it is we are stuck with half baked news letters and magazines published by manufact-urers that don't support after market equipment that make our computers worth using.

I am glad you are partaking in this venture and hope that you succeed in producing the best Commodore magazine available.

Sincerely,
Michael Aichlmayr
Lewiston, Idaho

Dear Commander Magazine,

I am very pleased and excited to hear about your new magazine. I have been searching for months now to find a publication devoted to the Commodore line of computers.

It will be extremely refreshing to read a magazine that will "tell it like it is" about software, hardware and special applications that are available for Commodore users like myself.

I sincerely hope (for all of us) that you continue to publish your wonderful magazine. Good luck and thank-you Commander!

John P. Gabbard
Dallas, Oregon

In My Opinion

In my opinion the VIC 20 is one of the nicest little machines on the market today. So why is it being under rated, under utilized and under promoted?

In this column I'd like to explore some of these items, which are, strictly, my own opinions and a philosopher once said that "opinions are like navels, everyone has one and they are all different." If your "navel" differs from mine I'd be glad to hear from you and to respond.

Now, why do I think the VIC is under utilized? Look at any computing magazine and examine the ads closely. 90% of everything that is available for the VIC 20 is a game of some sort or another, is that any way to treat a computer? Most of the other applications that are not games will run on the "unexpanded VIC," when are the producers of software going to start giving us some really significant applications? Applications that require a little more than 3.5k of RAM.

Have you ever tried to find anything on disk? What a laugh, the few items that are available on disk are hardly worth the effort to find, there again, when are we going to be able to utilize the peripheral items that are available?

I do have a few positive opinions. I think Scott Adams is a genius, devious as the dickens but a genius nontheless. If you have not yet tried a Scott Adams adventure game then I'd recommend that you do so as soon as possible—when you have a lot of time and don't mind losing sleep.

It is also my opinion that some of the new cartridge games are very worthwhile as far as games go. They will keep the kids occupied for hours and keep you away from your computer when you could be doing something really useful. The best games I've seen so far are GORF and OMEGA RACE.

There **are** some items available that require memory expansion. The few I've stumbled across were games. I am not against games, don't get me wrong, it is just that I think there should be something other than games to use the VIC for. I didn't buy my VIC just to use as a toy, had I wanted that there are a couple of very well advertised "computer systems" available that are nothing more than game machines.

Another complaint that I have is that there is so little information available to a specific group of people, I refer to people that know a little too much to be considered beginners but don't have anywhere near enough knowledge to be programming in machine language or some other exotic method of computing. People like me...People that would like to know simple things like how to trade programs with someone else that has a MODEM hooked to his VIC (I hear it can be done but I don't know how to do it), or simple procedures like changing some information on a disk without re-writing the whole disk. Where does information like that come from? In my opinion it is not easily understand-able when you do find it.

Editorial

Welcome to the exciting world of Commodore personal microcomputing! The decade of the 1980's will see a technological explosion similar to the industrial revolution of a century ago. This explosion will be fueled by the very inexpensive personal computer — today you can purchase 100,000 VIC-20s for what the first computers cost only thirty years ago. The power of even a VIC-20 was not available before the late 1940s. Soon every home, office and school room will have a computer in it and the real challenge of the 80's will be upon us - to provide the education and software to utilize all of these computers.

Commander magazine is dedicated to providing the Commodore computer user with all of the information he will need to make informed decisions when it comes to purchasing hardware and software. Many articles and departments will be aimed at providing the latest information on how to use your computer. We will make every effort to provide a quality mix of articles on business, educational and recreational subjects and tutorials on Basic, assembly and other languages. In order to provide the best possible service to you, the reader, we will need feedback on how you feel about the job we are doing. Please do not hesitate to write if you have any suggestions, complaints and/or praise.

The editorial premise of Commander can be simply stated as "just". In an effort to avoid platonic discussions of justice, suffice it to say that it will be difficult to provide justice to all segments of our readership but that is what we will try to do. Commander cannot, however, attempt to serve as a policeman to the industry, but we will not accept advertising from companies which do not deal fairly with their customers nor will Commander support clubs or any organization which trades, swaps or "libraries" non public domain software.

It is our belief that one of the most important factors in the success of any computer is the amount of software available. Past history has shown that the most and the best software will be developed by small, independent companies. For these companies to survive, customers must *BUY* their software and this will not happen if pirated copies are spread around. If all copies are legally obtained, the software companies can lower their prices and more software will be produced.

Commander will be reviewing as much hardware and software as possible in order to keep you as informed as possible. Patronizing our advertizers will provide you with a quality product at a reasonable price. Should you ever not find this to be true, please let us know at once.

Commander is the glue which is going to cement the user, author and manufacturer together and turn Commodore personal computing into the nation's favorite pasttime. Join us and be in on all the fun.

— COVER BY —

Computertowne is proud to be Pierce County's First Full Service Exclusively Commodore Computer dealer. We have a representative stock of the full line of Commodore products as well as selected items from 3rd market sources for use with the Commodore Computers. In addition our in house partnership with Allied Business Machines and Tacoma Cash Register make us naturals to serve every business machine need.

Remember Computertowne for Sales, Service and Support

1215 Center, Tacoma, Washington 98409

(206) 272-2271

News Releases

New Company Formed To Market Encryption Device in US

San Diego, CA — Patricia Doering & Associates, a San Diego based communications company, announced formation today of a new computer distribution firm called DISTRIBUTION UNLTD. Under the direction of Patricia Doering, the new company will begin active marketing of a new encryption device manufactured in England called SECURE. According to Ms. Doering, SECURE will initially be sold directly to end users in the United States with negotiations underway with several major U.S. distributors and dealers. The full marketing campaign is expected to break in January, 1983 with emphasis upon direct mail, leading computer publications' advertising and public relations activities.

A program encryption kit for the 3000, 4000 and 8000 series COMMODORE computers, SECURE is endorsed by Commodore and produces 256 encryptions of single programs at random. "It's an unbreakable device," Doering says, "and we'll market at approximately half the price of other such devices."

Information about SECURE and the new company can be obtained by contacting: Patricia Doering & Assoc. or Distribution Unltd., P.O. Box 81702, San Diego, CA 92138-1702. (714) 299-3718.

Microcomputer Products

A new 40-page catalog from ELECTRONIC SPECIALISTS presents their line of MICROCOMPUTER interference control products. Protective devices, Line Voltage Regulators and AC Power Interrupters are also included.

Descriptive sections are included outlining particular problems and suggested solutions. Typical applications and uses are highlighted. Request Catalog 821.

ELECTRONIC SPECIALISTS, INC., 171 SO. MAIN ST., BOX 389 NATICK, MASSACHUSETTS 01760. Phone: (617) 655-1532.

EXCHANGE

EXCHANGE — read and write IBM standard floppy disk files on a Commodore PET/CBM with a new system from Microtech. The EXCHANGE System consists of an eight inch floppy disk drive, controller, and a special software package called FILEX.

EXCHANGE can be used to transfer data files and information between large computers and the CBM. Remote data entry and processing facilities can be set up using this system. Enter data and record a disk file that can be read by many computers. Use the CBM to read data from a large computer and do remote processing.

EXCHANGE utilizes the PEDISK II Model 877 Floppy Disk. It is available in single or dual drive versions. The FILEX software is provided on ROM and consists of five routines to initialize, open input files, open output files, get records, put records, and close files. All information is transferred using BASIC variables. All files are written and read using the IBM 'Diskette 1' interchange standard. IBM directory information is created also. A special utility is available that allows files created by the COPY-WRITER word processor to be recorded as IBM type files.

The EXCHANGE System costs $1295.00 in a single drive version and $1795.00 in a dual drive version. The FILEX software is available separately at $250.00 for those who have PEDISK II 8' systems. Contact CGRS Microtech, P.O. Box 102, Langhorne, Pa. 19047. (215) 757-0284.

New Products

The Commander

THE COMMANDER is a 4K ROM for use with a Commodore system 2000, 4000, 8000 series, AND 4.0 or 4.1 Basic. It contains exclusive programmable commands. These powerful commands, under program control or direct mode, contain an enhanced COMMON function which retains all variables and arrays. All programs contained in THE COMMANDER are copyrighted 1982 by Commander Systems, Inc.

Listed below are comands, which until now were only available on large systems.

Common:
THE COMMANDER has an enhanced COMMON operation with INSERT, APPEND, DELETE, AND RE-DIMENSION. All variables and arrays are retained.

Insert:
Inserts a disk program, or subroutine, into the beginning, or between specific line numbers of a running program, without losing variables or arrays. Program execution will continue at any line number, even one just inserted.

Append:
Appends another program or subroutine to the end of the running program, and continues execution without losing variables.

Delete:
Deletes any portion of a running program under program control, and continues execution. All deleted memory is reclaimed for system use, and all variables and arrays are retained.

Com Literals:
Four EXTRA commands to COMMON literal strings.

Printusing and Image:
Formats a floating point variable into a string. Includes selection and placement of the dollar sign "$", commas, decimal places, total length, trailing 0's, and leading spaces.

Convert:
Converts Commodore character set to standard upper/lowercase for printing to an ASCII printer.

Frame:
This command prints a message to the screen (centered on 40 or 80 Columns), within a reverse video frame to inform the user of system status.

Overlay:
This command loads and runs any length program, and automatically adjusts to the length of the new program. All unused memory is reclaimed.

Enhanced Get:
As its name implies, an ENHANCED keyboard input routine with HUNDREDS of programmable variations.

Re-Dimension:
Allows dynamic re-dimension of arrays, while program is running, without losing variables or any array data.

Return Clear:
Deletes all GOSUB's and FOR...NEXT loops from the operating system. Allows an exit without a RETURN or NEXT.

Computed Goto:
Any Basic math may be used to compute a GOTO. Same as GOTO GT.

Window:
Clears any number of screen lines, or a Window anywhere on the screen (40 or 80 columns).

Mat Print#:
(Speed data) Writes an entire array to the disk, FAST.

Mat Input#:
(Speed data) Inputs an entire array from the disk, FAST.

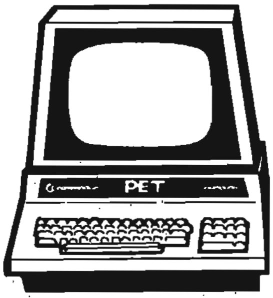
Mat Init (" "):
Nulls (" ") an entire string array, FAST.

Mat Zer:
Clears an entire array to ZERo (0), FAST.

String:
Inputs a string from any disk file up to 255 characters in length. Will input leading spaces, or any ASCII character including commas.

THE COMMANDER ©: $70.00
Includes manual and Demo diskette. Specify ROM expansion socket $A000 or $9000, and 4040 or 8050 demo diskette.

Commander Systems, Inc.
4505 Jackson Street
Hollywood, FL 33021
Phone (305) 962-5183



PET / CBM

All About CB2 Sound

by Louis L. Sander
Pittsburgh, Pennsylvania

Every PET/CBM owner has heard of "CB2 sound", which was developed by early PET users as a way to give their machines a voice. Information about this interesting but "unofficial" Commodore feature has appeared in such scattered places that few PET/CBM owners *really* understand it. This article presents a full explanation of CB2 sound, including the theory behind it, "how to" information on hooking it up, its relationship to "music", and a comprehensive demonstration program for the sound enthusiast.

How CB2 Sound Is Made

CB2 is the name of a control line on the input/output chips inside the PET/CBM. The CB2 line from one of these chips is brought out to the User Port, where the knowledgeable programmer can use it to put his machine through some extraordinary paces. The less-knowledgeable person can use CB2 to make some interesting sounds, by switching the voltage on the line alternately high and low, then using this changing voltage as an input to an audio amplifier. The resulting tones have a clear and haunting quality which makes them pleasing to the ear.

To create our sound, we set PET/CBM up to send a repeating series of pulses out to CB2, then we tell it what pulses to send and how fast to repeat them. All this is done by POKEing three memory locations. POKE 59467,16 enables the pulse sending mode. A POKE to 59466 establishes the pattern of pulses, and one to 59464 determines the repetition rate. To disable CB2 sound, we POKE zeroes into these locations.

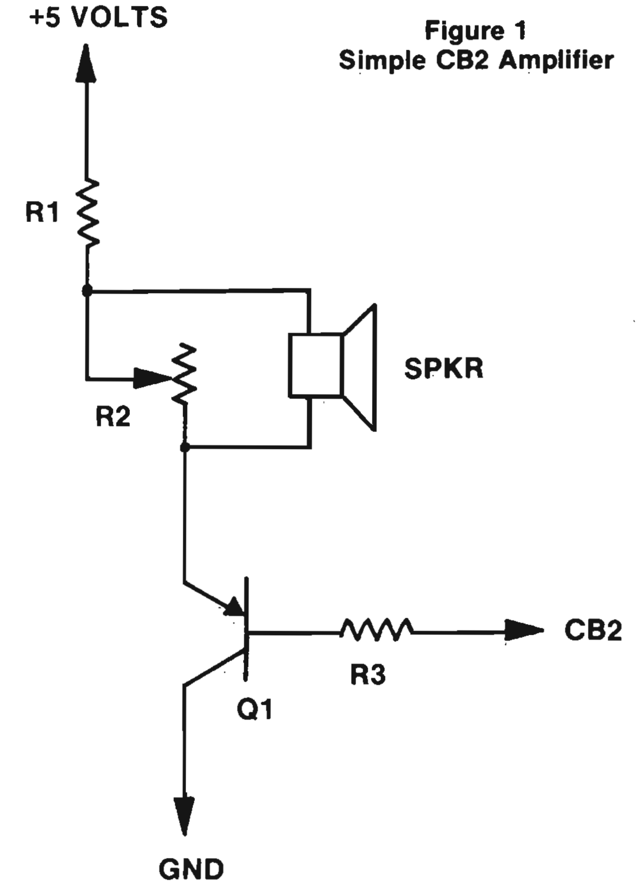


Figure 1
Simple CB2 Amplifier

PARTS LIST

Part numbers are from 1983 Radio Shack Catalog. Total Cost $6.35

R1 - 27 ohm, ½ watt (271-006)
R2 - 100 ohm speaker control (40-550)
R3 - 470 ohm, ½ watt (271-019)
Q1 - MPS3638 or equivalent (276-2032)
SPKR - 2¼" replacement type (40-246)

To better understand PET/CBM's sound generating process, let's look closely at POKE 59466,85. The binary representation of 85 is 01010101, and when this pattern is sent out to the CB2 line, bit-by-bit from left to right, it comes out as lo-hi-lo-hi-lo-hi-lo-hi. (Lo represents a low voltage level on the CB2 line, and hi represents a high voltage level. Typically, lo=0 volts, and hi=5 volts). If this pattern is repeated indefinitely, the result will be a perfect square wave, of the form 0101010101010101... *ad infinitum.* If it is input to an amplifier, we will hear it as a tone, because the amplifier responds to each of the voltage transitions from lo to hi or vice-versa. If we send our pattern out to CB2 at 500 bits per second, the tone will have a frequency of 250 cycles per second, or 250 Hertz. (1 cycle = 1 lo plus 1 hi).

If we were to POKE 59466,51, the pattern would be 00110011, or lo-hi-lo-hi. (Things do not return to zero between the l's). If we sent this pattern to our amplifier at the same rate as before, the new tone would be at half the frequency of the old, because there are now only four voltage transitions per pattern, where before there were eight. (Besides at the obvious times, transitions occur when the final 1 of one pattern is replaced by the initial 0 of its repeated twin, or when a final 0 is replaced by an initial 1.) If we POKE 59466,17, the pattern is 00010001, and there are still four transitions per pattern, but the tone is no longer a square wave — it will sound somewhat different than the other one, even though its fundamental frequency is the same.

Counting all the numbers between 0 and 255, there are 256 different values that we can POKE into 59466. But many of them produce identical patterns, and therefore identical sounds. POKEs of 1, 2, 4, 8, 16, 32, 64, and 128, for example, all produce two transitions per pattern (one lo-hi and one hi-lo), and when these patterns are repeated indefinitely, the ratios of hi to lo time in each are exactly the same. For every practical purpose, these eight POKEs all produce the same sound.

A POKE to location 59464 determines how quickly each bit of our pattern is sent out to CB2. PET/CBM sends out another bit each

Figure 2

```
100 REM  ** PRINTS MUSICAL SCALE INFO **
102 REM     FOR 'ALL ABOUT CB2 SOUND'
103 REM
104 REM
105 REM
106 REM
110 REM
120 REM ** CONTAINS MANY EXTRA LINES **
1000 P3=50003:P4=59464:P6=59466:P7=5946
7:P8=59468
1010 KD=5:KE=515:KU=12:SD=136:SE=133:SH
=516:SK=537:REM ** OLD ROMS **
1020 IFPEEK(P3)=1THENKE=151:SD=49:SE=46
:SH=152:SK=144:REM ** 3.0 ROMS **
1030 POKEP4,0:POKEP6,0:POKEP7,0
1040 IFPEEK(50003)THENKE=151:SH=152
1210 REM *** SETUP PRESET TONES
1220 IFPEEK(825)<>OTHENFORI=825TO951:PO
KEI,0:NEXT
1230 DATA15,15,51,51,85,85,85,85,85,248
,198,248,198,248,198,123,48,23
1240 FORI=1TO9:READSR:POKE904+I,SR:NEXT
1250 FORI=1TO9:READRA:POKE841+I,RA:NEXT
1260 PRINT"PRESS ANY KEY TO PRINT SCALE
 INFO"
1280 GETA$:IFA$=""THEN1280
1290 IFA$=CHR$(3)THEN1880
1300 GOTO2180
1310 IFA$="M"THEN2180
1320 POKEP7,16:SR=85:D1=2:RA=248
1780 REM *** SILENCE THE TONE
1790 POKEP4,0:POKEP6,0:POKEP7,0
1800 PRINT"{CLEAR DOWN REV}FREQUENCY (H
Z){OFF} ="FR
1810 PRINT:PRINT"{REV}TONE #{OFF}"TN"OF
 514"
1820 PRINT"{DOWN REV}POKES{OFF} 59467,
16 : 59466,"SR": 59464,"RA
1830 PRINT"{5DOWN}    {REV}PRESS ANY KEY
 TO RESTART THE TONE."
1840 GETA$:IFA$=CHR$(3)THEN1880
1850 IFA$=""THEN1840
1860 GOTO1340
1870 REM *** RESET GRAPHICS, CB2, & STO
P             KEY, THEN STOP
1880 PRINT"{CLEAR}":POKEP8,12:POKEP7,0:
POKEP6,0:POKESK,SE:END
1890 POKESK,SD:GOTO1320
2170 REM *** MUSICAL INFO
2180 REMTARGET
2300 RESTORE:FORI=1TO18:READA$:NEXT
2355 OPEN4,4:CMD4
2356 PRINTCHR$(14)CHR$(12)CHR$(8)
2370 PRINTSPC(18)"CLOSEST   PET"
2380 PRINT"NOTE  STD FREQ   PET TONE   T
ONE#"
```

time an internal timer counts down from PEEK(59464) to zero; on reaching zero, the timer starts again at PEEK(59464). So if 59464 contains a 127, the bits from 59466 will be sent out twice as fast as if it contains a 254. The difference will be heard as a difference in pitch, with the first tone being higher-pitched than the second.

With 256 different POKEs for 59464, and 256 for 59466, there are 65536 different combinations possible for controlling CB2. But because of the numerous duplicate patterns, many of them produce identical sounds. Sorting through the possibilities reveals 17 completely unique waveforms. Multiplying this by the possible repetition frequencies gives over 4000 different CB2 tones. Considering that you can step through them under computer control, producing sirens, whistles, etc., the number of possibilities is infinite. With our demonstration program, you can explore the individual tones to your heart's content.

Connecting An Amplifier

Adding an amplifier is a simple matter for anyone familiar with elementary electronic work, but making a mistake can destroy some of the chips in your machine. If you've done work like this before, this job will require nothing more than meticulous care. But if you're a complete novice, you should get the assistance of your dealer's service department, or of someone with electronics construction experience.

The first step is to buy a connector for your User Port. What you want is a 12-position, 24-contact printed circuit board edge connector with 0.156 inch contact spacing. They are made by the thousands for industrial use, but many electronic stores don't carry them, and you won't find them at your local Radio Shack. Most Commodore dealers should have them, or should be able to suggest a source of supply.

Your Commodore User Manual shows the User Port's location, in the chapter on Interfaces and Lines. It is the middle connector of the three on the back of the machine. The CB2 connection is pin M of the port, and

```
2385 PRINT
2390 FORI=1TO100:READNO$,SF$,PF$,PN$
2392 IFI<>32THEN2400
2394 PRINT:PRINT:PRINTSPC(18)"CLOSEST
 PET"
2396 PRINT"NOTE  STD FREQ   PET TONE  T
ONE#":PRINT
2400 PRINT" "NO$SPC(5-LEN(NO$));
2401 IFVAL(SF$)<1000THENPRINT" ";
2402 PRINTSF$"   ";
2403 IFVAL(SF$)<1000THENPRINT" ";
2404 PRINTPF$SPC(6-LEN(PN$));
2406 PRINTPN$
2410 IFNO$="C"THENREADSR,RA:REM***POKEP
7,16:POKEP4,RA:POKEP6,SR
2425 IFPN$="484"THENI=100
2430 NEXT
2432 PRINT:PRINT:PRINT:PRINT
2435 PRINT#4:CLOSE4
2440 IFPN$<>"484"THEN2310
2450 PRINT"{DOWN}   END OF MUSICAL SCAL
E INFO{DOWN}"
2460 END
2500 REM *** TONE INFO
2510 DATAB,246.941,247.036,5,"C",261.62
5,261.506,19,15,237
2520 DATAC#,277.182,277.778,33,D,293.66
4,293.427,45
2530 DATAD#,311.126,310.945,57,E,329.62
7,328.947,68
2540 DATAF,349.228,349.162,79,F#,369.99
4,369.822,89
2550 DATAG,391.995,390.625,98,G#,415.30
4,416.667,108
2560 DATAA,440.000,440.141,116,A#,466.1
63,466.418,124
2570 DATAB,493.883,494.071,134,"C",523.
251,523.012,148,51,237
2580 DATAC#,554.365,555.556,162,D,587.3
29,586.854,174
2590 DATAD#,622.253,621.890,186,E,659.2
55,657.895,197
2600 DATAF,698.456,698.324,208,F#,739.9
88,739.645,218
2610 DATAG,783.991,786.164,228,G#,830.6
09,833.333,237
2620 DATAA,880.000,880.282,245,A#,932.3
27,932.836,253
2630 DATAB,987.766,988.142,263,"C",1046
.502,1046.025,277,85,237
2640 DATAC#,1108.730,1111.111,291,D,117
4.059,1173.709,303
2650 DATAD#,1244.507,1243.781,315,E,131
8.510,1315.789,326
2660 DATAF,1396.912,1396.648,337,F#,147
9.976,1479.290,347
```

the accompanying GROUND connection is pin N. Viewed from the rear of the computer, these are the two rightmost contacts on the bottom row of the connector. They are nicely illustrated in the User manual, too.

Don't make any connections to your machine until you are absolutely certain you have found the proper pins. Don't plug your connector in upside down, and don't make the mistake of plugging it into the IEEE port, which takes an identical connector. You can avoid these problems by buying polarizing pins for your connector, and matching them up with the slots between contacts on the PC board at the User Port.

Once you have a connector and know where CB2 and GROUND are, you need a suitable amplifier to hook them to. Your choices are to build a small amplifier, to buy one, or to use an amplifier you already own. Figure 1 shows an amplifier that can be built for only a few dollars, from parts available at any Radio Shack. If you prefer to buy an amplifier, the Radio Shack #277-1008, for $11.95, works very well. Cable assembly #42-2435 will attach it to your CB2 connector, and a 9 volt battery will last for weeks, unless you forget to turn the amplifier off when you aren't using it. To use your stereo amplifier with CB2, connect its input to CB2 through a 50K - 500K series resistor. If you are unsure of how to do this, call on an expert for assistance.

Using The Tone Generator Program

The accompanying TONE GENERATOR program is designed to work on any PET or CBM, and has been tested on most models. It provides a comprehensive set of tools for working with CB2 sound, and is well worth the effort spent in keying it in. Do that now, then follow the instructions below, which will familiarize you with all its features.

1. If you have a CBM 8032, 8096 or Super PET, remove the first REM statement from line 530; if you have a "Fat 40" machine, (40 columns, 12 inch screen), remove the first REM from line 540. Taking these steps will establish the proper values of TU, TD, WU, and WD for your machine.

```
2670 DATAG,1567.982,1572.327,357,G#,166
1.218,1666.667,366
2680 DATAA,1760.000,1760.563,374,A#,186
4.654,1865.672,382
2690 DATAB,1975.532,1968.504,389,"C",20
93.004,2100.840,397,85,117
2700 DATAC#,2217.460,2212.389,403,D,234
4.318,2336.449,409
2710 DATAD#,2489.014,2500.000,416,E,263
7.020,2631.579,421
2720 DATAF,2793.824,2808.989,427,F#,295
9.952,2976.190,432
2730 DATAG,3135.964,3125.000,436,G#,332
2.436,3333.333,441
2740 DATAA,3520.000,3521.127,445,A#,372
9.308,3731.343,449
2750 DATAB,3951.064,3968.254,453,"C",41
86.008,4166.667,456,85,58
2760 DATAC#,4434.920,4464.286,460,D,469
8.636,4716.981,463
2770 DATAD#,4978.028,5000.000,466,E,527
4.040,5319.149,469
2780 DATAF,5587.648,5555.556,471,F#,591
9.904,5952.381,474
2790 DATAG,6270.928,6250.000,476,G#,664
4.872,6578.947,478
2800 DATAA,7040.000,6944.444,480,A#,745
8.616,7352.941,482
2810 DATAB,7902.128,7812.500,484

READY.

     *** TONE GENERATOR 6.0 ***

100 REM
110 REM
120 REM
130 REM
140 REM
150 REM
160 REM
170 POKE59468,12:FORI=825TO951:POKEI,0:
NEXT:GOTO360
180 REM ** POKE & PRINT FREQ INFO
190 POKEP7,16:POKEP4,RA:POKEP6,SR:D=1:W
N=WN(SR)
200 IFSR=85THEND=2:TN=514-RA
210 IFSR=51THEND=4:TN=385-RA
220 IFSR=15THEND=8:TN=256-RA
230 FR=500000/((RA+2)*D)
240 PRINT:PRINTB$TAB(29)"              ":PR
INTTAB(20)"{DOWN}    "
250 IFD=1THENPRINTB$TAB(29)"N/A":PRINTT
AB(20)"{DOWN}N/A          ":GOTO280
260 PRINTB$TAB(28)FR
270 PRINTB$"{2DOWN}"TAB(19)TNTAB(24)"OF
 514"
```

2. RUN the program and follow the brief instructions. When you hear a tone and see its screen display, move on to instruction #3.
3. Press SPACE a few times, to see how it is used to start and stop the tone.
4. Stop the tone and look closely at the screen display. It shows the waveform of the tone you are hearing, its frequency, and an arbitrary "tone number" which helps to identify it for future recall. The screen also shows the three POKEs which are made to produce this particular tone.
5. Start the tone again, and use the GREATER THAN and LESS THAN keys to change its frequency. Observe that holding either key down causes the tone to keep changing, one tone number at a time. SHIFTing either key moves it faster.
6. As the tone changes, notice that the screen display changes along with it. Usually only the frequency, tone #, and the POKE to 59464 change, but at certain points 59466 and the waveform change, as well. But the waveform is always a square wave — that is, the times of hi and lo are always equal. Observe that the low-numbered tones are much closer together in frequency than the higher ones.
7. Now use your new knowledge to return to Tone #137, a 500 hz square wave.
8. Use the SQUARE BRACKET keys to change the shape of the waveforms, and notice the difference in the quality of the sound. Observe that only 59466 is changing, and that for non-square waves the Tone Number and Frequency are not defined. Once you get a non-square waveform with an interesting sound, you can change its pitch with the GREATER THAN and LESS THAN keys, which will change only the POKE to 59464.
9. With the tone sounding, press each of the numeric keys between 1 and 9. Each one of them will call up an interesting pre-programmed sound.

```
280 PRINTTAB(24)"{DOWN}     {3LEFT}"SRTAB
(36)"     {4LEFT}"RA
290 IFSR=WFTHENRETURN
300 PRINT"{HOME 6DOWN}     ";:WF=SR
310 FORI=1TO4:BV=128:FORK=1TO8:IFWFANDB
VTHENPRINT"{REV} {OFF}";:GOTO330
320 PRINT" ";
330 BV=BV/2:NEXT:NEXT
340 RETURN
350 REM ** INSTRUCTIONS
360 PRINT"{CLEAR 3DOWN} TONE GENERATOR
- BY LOUIS F. SANDER"
370 PRINT"{3DOWN}PET CAN GENERATE 514 D
IFFERENT SQUARE"
380 PRINT"WAVE TONES BETWEEN 243 AND 12
5000 HZ.,"
390 PRINT"AS WELL AS THOUSANDS OF COMPL
EX TONES."
400 PRINT"{DOWN}THIS PROGRAM LETS YOU H
EAR EVERY ONE"
410 PRINT"OF THEM, AND SHOWS YOU THE WA
VEFORM"
420 PRINT"AND PROPER POKES FOR EACH."
430 PRINT"{DOWN}TO HEAR THE TONES, YOU
MUST CONNECT"
440 PRINT"AN AUDIO AMPLIFIER TO THE CB2
 LINE, AS"
450 PRINT"DESCRIBED IN MANY PET PUBLICA
TIONS."
460 PRINT"{5DOWN}     PRESS ANY KEY TO S
TART . . ."
470 REM ** INITIALIZE
480 B$="{HOME 11DOWN}":P3=50003:P4=5946
4:P6=59466:P7=59467
490 KE=515:SD=136:SE=133:SH=516:SK=537:
REM ** ORIGINAL ROMS
500 IFPEEK(P3)=1THENKE=151:SD=49:SE=46:
SH=152:SK=144:REM ** UPGRADE ROMS
510 IFPEEK(P3)=160THENKE=151:SD=88:SE=8
5:SH=152:SK=144:REM ** 4.0 ROMS
520 TD=5:TU=12:WD=7:WU=14:REM ** CONTRO
L KEYS FOR ALL MACHINES WITH 9" SCREENS
530 REM TD=44:TU=46:WD=219:WU=221:REM C
ONTROL KEYS FOR 8032
540 REM TD=44:TU=46:WD=91:WU=93:REM CTR
L KEYS FOR 40 COL W/12" SCRN (UNTESTED)
550 DIMSR(17),WN(85):FORI=1TO17:READSR(
I):WN(SR(I))=I:NEXT
560 DATA 1,3,5,7,9,11,15,17,19,21,23,27
,37,43,45,51,85
570 IFPEEK(825)<>0THENFORI=825TO951:POK
EI,0:NEXT
580 FORI=1TO9:READRA:POKE841+I,RA:NEXT:
FORI=1TO9:READSR:POKE904+I,SR:NEXT
590 DATA 248,248,248,248,248,123,48,23,
18,15,5,19,51,85,85,85,85,85
600 POKEP6,0:POKEP7,0:POKESK,SD:SR=51:R
A=248:REM ** 500HZ
```

10. Use SPACE to turn the tone off, then press each of the number keys again. Surprise! — your computer is now a musical instrument. When you press SPACE again, both the tone and the screen display will match the last "note" that you played on the number keys.

11. Now start the tone and use the BRACKETS and GREATER THAN/LESS THAN keys to produce a note you like. While it is sounding, press SHIFTED A. Then change the tone and press SHIFTED S. Hit UNshifted A and S in succession, and observe that THEY are now programmed to produce the tones you selected by shifting them.

12. Turn off the tone, and see that UNshifted A and S now have the same "organ key" characteristic as the number keys you tested in step 10.

The feature you used in steps 11 & 12 works with any key that normally prints a character, except, of course, the four that are used to change tones. You can re-program a number key, or any key, just by pressing it and SHIFT together, while sounding the tone you want it to produce. This feature is useful when you want to store and recall several specific tones, such as when you make a computer organ. Using it, I once made my PET into a programmed signal generator for adjusting modems, and it was worth its weight in gold.

All About CB2 Sound - Figures and Programs

Figure 1 is a schematic of a CB2 amplifier, with parts list. (attached)

Figure 2 is a listing of musical scale information. (attached) It can be reproduced on your printer by RUNning the SCALE PRINTER program.

```
610 GETA$:IFA$=""THEN610
620 IFA$=CHR$(3)THEN1120
630 REM ** PRINT SCREEN
640 PRINT"{CLEAR}  TONE GENERATOR - BY
LOUIS F. SANDER"
650 PRINT"{2DOWN REV}WAVEFORM{OFF}:"
660 PRINTTAB(28)"{3DOWN}^^^^^^^^"
670 PRINTTAB(13)"BITS IN 59466{OFF}: {R
EV}76543210"
680 PRINT"{2DOWN REV}SQUARE WAVE FREQUE
NCY (HZ){OFF}:"
690 PRINT"{DOWN REV}SQUARE WAVE TONE #{
OFF}:"
700 PRINT"{DOWN REV}POKES{OFF} 59467, 1
6 : 59466,     : 59464,"
710 REM ** SCAN KEYBOARD
720 GOSUB190
730 GETA$:I=PEEK(KE):IFA$=""THEN830
740 IFI=TDORI=TUORI=WDORI=WUTHEN830
750 IFA$=" "THEN1010
760 IFA$=CHR$(3)THEN1120
770 REM ** TONE 'MEMORY'
780 A=ASC(A$):IFA=188ORA=190ORA=219ORA=
221THEN720
790 IFA>160ANDA<224THENPOKEA+665,RA:POK
EA+728,SR:POKEP6,0:GOTO720
800 IFA>32ANDA<96ANDPEEK(856+A)>0THENRA
=PEEK(793+A):SR=PEEK(856+A):GOTO720
810 GOTO730
820 REM ** CHANGE TONE OR WAVEFORM
830 J=PEEK(SH)
840 IFI=TUTHENRA=RA-(1+9*J):GOTO890
850 IFI=TDTHENRA=RA+(1+9*J):GOTO890
860 IFI=WDTHENWN=WN-1:GOTO960
870 IFI=WUTHENWN=WN+1:GOTO960
880 GOTO730
890 IFSR=15ANDRA=<126ANDI=TUTHENSR=51:R
A=RA+129
900 IFSR=51ANDRA=<126ANDI=TUTHENSR=85:R
A=RA+129
910 IFSR=85ANDRA>255ANDI=TDTHENSR=51:RA
=RA-129
920 IFSR=51ANDRA>255ANDI=TDTHENSR=15:RA
=RA-129
930 IFRA<0THENRA=255:IFSR=85THENSR=15
940 IFRA>255THENRA=0:IFSR=15THENSR=85
950 GOTO720
960 IFWN<1THENWN=17
970 IFWN>17THENWN=1
980 SR=SR(WN)
990 GOTO720
1000 REM ** SILENCE THE TONE
1010 POKEP6,0:POKEP7,0:SI=1
1020 GETA$:IFA$=""THEN1020
1030 J=PEEK(KE)
1040 IFA$=CHR$(3)THEN1120
1050 IFA$=" "THENSI=0:GOTO720
1060 A=ASC(A$):I=PEEK(856+A)
```

Pet And The Musical Scale

When you use your computer for playing music, the subject of "scales" is bound to come up. It's a mysterious subject for the non-musician, but we can shed light on it here. Scales involve the intervals between tones, or the ratio of one frequency to another. Starting with one arbitrary frequency, it is possible to construct several different scales that "sound right" to most ears. Since the starting frequency can be anything the scale-maker chooses, there are more possible scales than there are scale-makers.

Musical experts have settled on one standard scale, to which they try to tune most of their instruments. The PET/CBM tones don't match this scale exactly, but many of them come close. Figure 2 lists 61 standard notes, along with the number and frequency of the closest PET/CBM tone. You can see that for the low notes, PET comes quite close to the standard, but that the higher we go, the further PET departs from perfection. To get an idea of what "close" means to musicians, consider this -- it is customary to tune living-room pianos to within a fraction of a Hertz of the standard tones. Our computer seldom gets that close, but its "music" will be acceptable to most ears.

Sound Effects

Once you've discovered the wide world of CB2 tones, you can play them one by one to your heart's content. It's fun to see how high a frequency your ears can hear. Most amplifiers and speakers won't give much output above 15000 hz or so, but it's interesting to see what happens in the range between 10000 and 15000, and to discover the ear's directive characteristics in that range.

But to take full advantage of CB2 sound, you'll want to work with sound effects that put out a whole series of tones under program control. The accompanying programs called STAR SOUNDS and WOLF WHISTLE illustrate some of the techniques which are used in creating sound effects. Many sound effects have been listed in PET/CBM

```
1070 IFA>32ANDA<96ANDI>OTHENRA=PEEK(793
+A):SR=I:GOTO1090
1080 GOTO1020
1090 IFJ<>255THENPOKEP7,16:POKEP4,RA:PO
KEP6,SR:WAITKE,255,J
1100 GOTO1010
1110 REM ** RESET CB2/STOP KEY & QUIT
1120 PRINT"{CLEAR}":POKEP7,0:POKEP6,0:P
OKESK,SE:END

READY.
```

```
100 REM *** STAR SOUNDS #1 ***
110 REM
120 POKE 59467,16
130 POKE 59466,15
140  FOR A=1 TO 20
150   FOR B=1 TO 3
160    FOR C=255 TO 100 STEP -A
170    POKE 59464,C
180    NEXT C
190   NEXT B
200  NEXT A
210 FOR I=1 TO 1000 : NEXT
220 POKE 59467,0
230 END
```

```
100 REM *** STAR SOUNDS #2 ***
110 REM
120 POKE 59467,16
130 POKE 59466,15
140 A=255
150  FOR B=A-4 TO A
160  POKE 59464,B
170  NEXT B
180 IF A<5 THEN POKE 59467,0 : END
190 A=A-1 : GOTO 150
```

```
100 REM *** WOLF WHISTLE ***
110 REM
120 POKE 59467,16
130 POKE 59466,15
140 FOR A=180 TO 20 STEP -2
150 POKE 59464,A
160 NEXT A
170 FOR B=200 TO 100 STEP -2
180 POKE 59464,B
190 NEXT B
200 FOR C=100 TO 250 STEP 2
210 POKE 59464,C
220 NEXT C
230 POKE 59467,0
240 END
```

publications, and I encourage you to find them and experiment. CURSOR #7, from February, 1979, had a program called SOUND!, which included about twenty sound effects of all sorts, some of them VERY good. The old PET User Notes and PET Gazette had a number of nice ones, too.

Cautions, Troubles, and Fine Points

When making the POKEs to start your sound, you MUST make POKE 59467,16 the FIRST one. Only after that has been done will the others have any effect. And you only have to make that one once.

If you cannot get CB2 sound, try unplugging your cassette recorder; sometimes that will fix things. If you still have no luck, check your amplifier. Finally, try replacing the 6522 chip inside your machine. I had one once that worked perfectly in every other respect, but its CB2 line was totally dead.

Once a CB2 tone is started, it continues until the proper POKEs are made to stop it, regardless of what else your computer is doing. This can be most disconcerting when your machine crashes in the middle of some spectacular operation, and the sound effect keeps on running at full blast. Also, ENabling CB2 sound DISables the two cassette ports, in a way that is not immediately obvious. (The motor starts and stops as usual, but no information is transferred). So it is vital to know how to disable CB2 sound, and to DO it at the proper times.

I have found that POKE 59467,0, just by itself, is enough to stop the sound and to enable the cassette functions, although I haven't tested it under every circumstance. POKEing zeroes into all three memory locations is guaranteed to reset everything properly.

Well, readers, you now qualify as CB2 experts. Maybe you don't know ALL about CB2 sound, but you know everything I've been able to find in my four-foot shelf of PET/CBM literature, and in many hours of POKEing around. I know you can use the knowledge for some sound purpose.

No-Cost Memory Expansion and Its Uses

by Roy Busdiecker
Woodbridge, Virginia

Sound too good to be true? Actually, the extra memory is already installed (which is why there's no cost), but probably 99% of all PET/CBM owners don't even know it's there!

Is there a catch? Yes, in fact there are several! First, there's not very much additional memory available. In the 4016 and 4032 (and the older 2001 series) computers, there are a grand total of twenty-four bytes available...NOT kilobytes, just bytes! The 8032 and SuperPET are a little more generous, in providing an extra 48 bytes. A second drawback is that you can't run BASIC programs in this free bonanza...but it does give a protected location for peek/poke storage of variables, or for short machine-language routines. It is protected storage, in the sense that it does not get wiped out when you do a RUN, or a cold or warm start of BASIC (if that last part sounds Greek, forget it...it's nice, but may not be important to you).

Where is this mysterious windfall memory? Figuratively speaking, it's tucked right around the edge of your screen! CBM/PET displays come in two versions. The original was 40 columns wide by 25 rows high, displaying a total of 1000 characters. The newer 8000 series has a display 80 characters in width, still 25 lines high, for a total of 2000 characters. Each character position on the screen is supported by one byte of computer memory, separate from the RAM and ROM used for and by programs. If you consult a "memory map" for your computer, you'll see that "screen memory" starts at location 32768 ($8000, in hexadecimal).

While screen sizes come in 1000 and 2000 byte sizes, computer memories come in powers of two. The powers of two which are closest in value to those screen sizes are 1024 (=2 10) and 2048 (=2 11). It's a simple matter of economics...it's cheaper to furnish the extra 24 or 48 bytes and never use them, than to piece together "exact fit" memories from smaller "chips".

Those extra 24 or 48 bytes constitute the "memory expansion" ...it's free because it's there, and it's expansion because it's not advertised and you probably didn't know you had it!

Where Is It?

Now that you know it's there, the next logical question is "how do I go about using it?"

In order to use the memory, we first need to locate it. Take a minute to look at figure 1, before you read on.

Recall that screen memory starts at location 32768. If you have a forty-column computer (2000 or 4000 series), the screen uses 1000 bytes of the memory, up through 33767. The extra memory starts at location 33768, and runs through 33791. That much is fairly simple...and you could stop at that point, if you desired.

An interesting phenomenon, created by a design shortcut, makes it look like those same 1024 bytes of memory are located in two different places. To prove it, try the following experiment. First, clear your screen, then type in the following command, and press RETURN.

POKE 32768,1

Instantly, the letter "A" will appear in the upper left-hand corner of your screen. If it's not there, it probably means that you either typed in the wrong number, or else you typed the command near the bottom of the screen and it scrolled upward off the screen when your computer replied,. "READY".

Now move the cursor back up to the line you just used to create the "A", and change it to the following.

POKE 33792,2

When you press RETURN, the "A" will change to a "B". Why? Well, 2048 bytes were set aside as the screen memory area, but only 1024 were used. Since nothing else was in the 1024 bytes following the screen memory, the designers did not bother to check which of those two 1024-byte blocks was being addressed (it's possible to reduce circuitry a little with techniques like this).

Each location from 30768 through 33791, then, has an "image" location which will act just like the original for PEEKs and POKEs. To find the image location for any of the "real" screen memory positions, just add 1024 to its address. Notice that in our experiment, the second POKE value we used was 33792, which is 32768 + 1024.

Similar, But Different

Because the 8032 has 80 columns, it uses 2000 bytes out of 2048 which are installed. Since only the same 2048 bytes were reserved for screen memory, the 8032 does not display the "image location" effect.

You can verify this for yourself by repeating our experiment on the 8032. Our assumption this time would be that the image, if it existed, would have to be addressed 2048 higher than a given screen location. Start with the same command used last time.

POKE 32768,1

The same "A" will appear at top-left of the screen. Now enter

POKE 34816,3

and "C" would replace "A" if there were an image. It does not, so there is not. If you type in the command which produced the "B" on the forty-column screen, that letter will show up near the right edge of the screen, halfway down (1024 positions away from the top left corner, at 80 positions per row).

Figure 2 summarizes the pertinent address locations for the 8032.

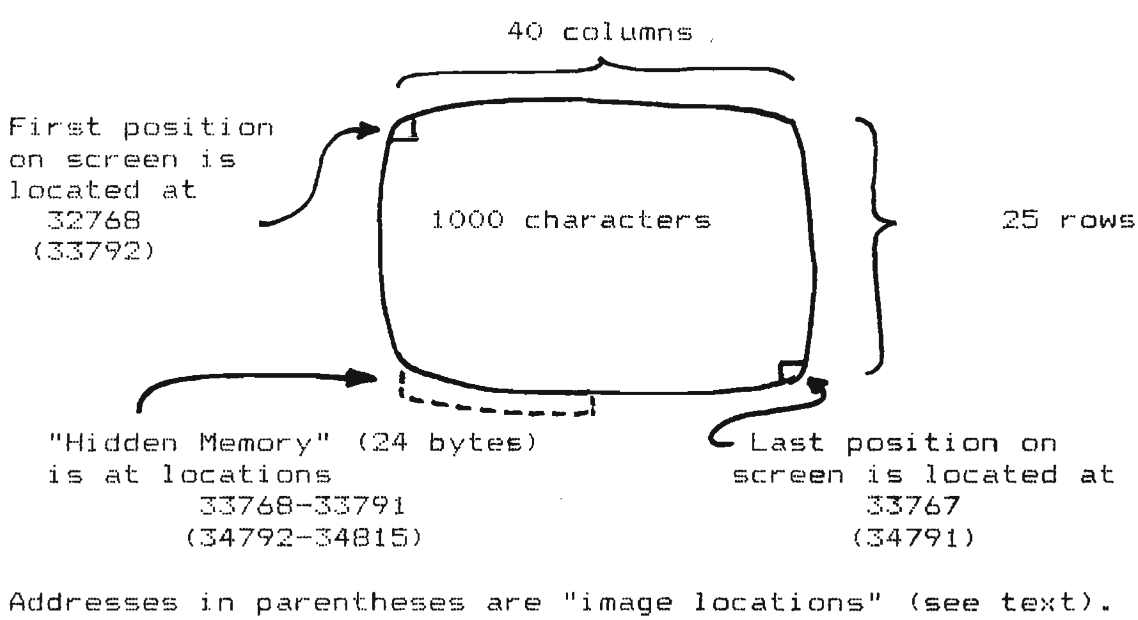


Figure 1. Forty-column screen memory.

"Universal" Hidden Memory

It's a nice feature (but getting harder and harder to do!) to make programs run on any version of PET/CBM.

If you compare the hidden memory locations for the two screen sizes (Figure 1 and 2), you'll see that the last 24 bytes of the 8032's hidden memory are the same as the "image" of the 40-column hidden memory (34792—34815).

Using addresses in the common range for your PEEK's and POKE's guarantees that they will work, and stay hidden, on either screen size. Score another point for program compatibility!

What's It Good For

The preceeding examples bring to mind one interesting use for this memory. Let's say you want to design a program that will "look right" on either 40 or 80 column screens. It's not hard to make adjustments to your PRINT statements to do this, but you need a test to find out what size screen the program is running on.

One approach is to use the image locations, but position them "off-screen" on one of the computers. For example, POKE 33768,1 will place an "A" just off the 40 column screen. On an 80 column screen, it would be at the center. Now if you POKE 34792,2 it will appear in the "hidden area" of the 8032, but will REPLACE the "hidden A" on the 40 column screen.

The following simple program illustrates the technique.

```
5 A$="40 columns wide"
10 PRINT "This screen is";
20 POKE 33768,1
30 POKE 34792,2
40 IF PEEK (33768)=1 THEN A$="80
   columns wide"
50 PRINT A$
```

Non-Reset Memory

The program below illustrates another interesting use of the hidden memory.

```
10 MEM=34792
20 IF PEEK (MEM)=1 THEN 60
30 POKE MEM,1
40 PRINT"THIS IS THE FIRST RUN"
50 STOP
60 PRINT"THIS PROGRAM HAS
   BEEN RUN BEFORE"
```

When you run the program, it will tell you whether or not it has been run previously during this session (since it was loaded). With a little extra effort, you could make it count the number of times it has been run. It is not possible to do the same with BASIC variables, because they are all cleared when you do a RUN...and the program would always think it was the first time.

In one of my own programs ("Disk Librarian", available from Micro Software Systems, P.O. Box 1442, Woodbridge VA 22193), the free expansion is used to hold the values of year, month, and day, which are used in conjunction with the TIME variable built into the PET/CBM operating system.

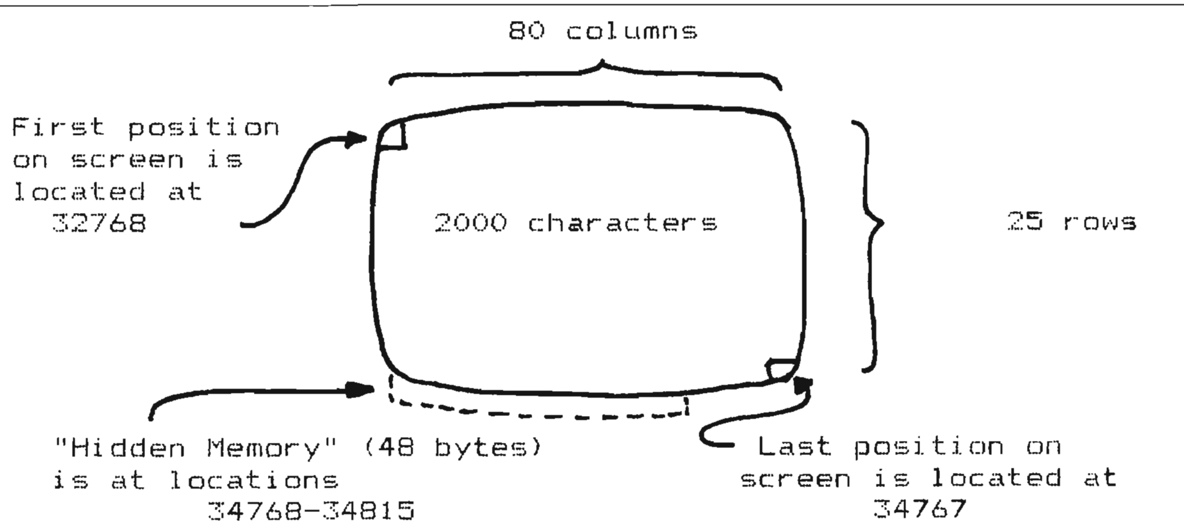
As long as you don't turn the computer off, values poked in the hidden memory will stay there even if you type NEW, or press a reset switch (e.g., UNCRASHER from Virginia Micro Systems, 13646 Jefferson Davis Highway, Woodbridge VA 22191).

Programs, Too!

In addition to storing values via PEEK and POKE, the extra memory can also be used to store short machine language programs. They can't be in BASIC because the PET/CBM does not look for BASIC programs in that area. They must be short in order to fit the available space. Twenty-four bytes allows only the most compact programs...but the forty-eight available on 8032's is large enough to be useful.

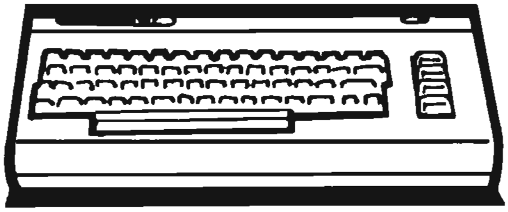
Other Uses?

Undoubtedly, some clever readers will think of other good applications for this special memory. I'll be watching future issues to see additional uses.



The eighty-column screen does not have "image locations" (see text).

Figure 2. Eighty-column screen memory.



VIC 20

A Beauty of a Utility Rom

by Tim Parker
Ontario, Canada

United Microware Industries Inc. of California introduced a ROM (read only memory) chip earlier this year that is quickly becoming indispensable for VIC-20 owners. It adds most of the functions that Commodore left out. These functions (which are available on most of their larger computers) are the utility functions that make programmer's lives much easier.

The Basic Utility Program (BUTI for short) is a machine language ROM that supplies seventeen new instructions to the interpreter available in the VIC-20. It is available in several formats, as chips that plug onto a board, or a cartridge that plugs into the expansion socket on the back of the VIC-20, or an expansion chassis. Chips are available as either one 2532 or two 2716 ROM (EPROM) units. (UMI also sells boards that will take the chips, as do several other manufacturers.)

The list of commands that is added by the BUTI is given in the accompanying table, along with a brief description of their purpose.

The well written forty-five page manual that accompanies the ROM has been designed for rapid reference for checking syntax and other details of each command (although when in use frequently, the HELP command of the ROM is usually more than adequate for reference). The added commands are listed alphabetically after a brief introduction and description, and installation instructions. The syntax that the manual uses is straightforward, and is well explained prior to the function list.

Some of the commands deserve a more thorough explanation than is available in the table, as they are of genuine value to programmers. (All commands can be abbreviated to a three letter short form for speed of programming.)

The APPEND command will add a program from media to the end of the current program in memory. (Note that insertion of a program into a specified slot in a current program in RAM cannot be done directly, although subsequent instructions can accomplish this purpose.) Duplication of line numbers should be avoided, as the BUTI does not check for repetition. The added program should have line numbers higher than those currently in memory.

AUTOmatic line numbering is rapid and easy. The increment and starting line number can be specified, or left to assume a default value. (The default is an increment of tens, starting at ten.) After leaving AUTO (by simply typing RETURN), it can be entered again with the same conditions; BUTI "remembers" the increment and last line number.

DELETE can be given a range of lines just as the LIST command. For example, DELETE 500- will delete all numbers above 500, while DELETE -50 deletes up to line 50. DELETE 500-1000 deletes all lines between 500 and 1000.

DUMP is a function software designers will not want to be without. It allows a list of all variables in use (or any particular type, if so specified by a one character limitor such as DUMP % for all integer variables) to be printed with their values. When used in program design, it allows a rapid check of which variable names are used, and what their current value is set at.

EDIT and FIND are string manipulators. They allow location or replacement of these variables within a program. (The length of the string name is limited to nine characters, which is not as restrictive as may originally sound.)

The KILL command disables the BUTI, and no calls can be made to it until it is enabled by cold boot of the computer, or a SYS call.

STEP and TRACE are debugging tools that are invaluable. STEP allows stepping through a program one statement at a time, each step executed by the press of the RETURN key. TRACE displays the continuing series of lines executed, and results of that execution. The TRACE function is slowed by pressing the shift key, and halted with the RUN/STOP key. Both functions can be turned off with the command OFF. Also, both TRACE and STEP (and OFF) can be executed by a SYS call to the processor from within a program, allowing selective study of a program.

The other commands are self explanatory as given in the table. In the end assessment, is the BUTI worth the $34.95 charged by UMI? The answer is an unqualified "yes". If you do any amount of program development, the ROM will save time, effort, and a lot of hair-pulling. Cudos are definitely to be extended to UMI for marketing the package, and especially to Steve Penners for designing the ROM.

Table of New Instructions and Their Use

APPEND
Adds a program from media to end of program in memory

AUTO
Automatic line numbering

DELETE
Erases a block of program lines

DUMP
Displays variables and their values

EDIT
Replaces defined string with a new string

FIND
Locates occurances of a given string

HELP
Shows BUTI commands and their syntax

KILL
Disables the BUTI without affecting the VIC-20

OFF
Disables the TRACE and STEP commands

RENUMBER
Renumbers blocks of programs and references

REPEAT
Turns on (or off) automatic key repeat

STEP
Single step execution of a program

TRACE
Displays line number being executed in program

UNNEW
Cancels NEW command (restores memory)

VIC
Configures VIC-20 memory size

#
Gives hexadecimal and binary equivalents

$
Gives decimal and binary equivalents

VICMON
A Machine Language Monitor for the VIC 20

169, 1, 141, 0, 30, 0. These numbers look meaningless, although they do in fact constitute a machine language program to place an "A" at the top of the screen. It's so much easier to write the program as LDA #$01, STA $7680, BRK, and the VICMON cartridge (manufactured by Commodore) allows you to do just that.

The most important task of a monitor-assembler is to translate mnemonics (such as "LDA") into binary numbers, and to place them into memory. The VICMON can, of course, do this flawlessly, saving the programmer hours of tedious and exasperating work. There is more to an assembler than that, however, and in this review I will mention the major good and bad features of VICMON.

Start off with the good. VICMON gives the user all the normal screen editing facilities found on the VIC 20: erasing characters, inserting characters, moving the cursor around, and what not. You can even see a whole program without resorting to successive LIST commands — keep pressing the DOWN CURSOR key and successive lines of the program appear on the screen; press the UP CURSOR key and you will scroll up through your program.

The monitor eases the task of writing programs in other ways as well. Branch statements in machine language often require that the target of the branch be specified as an offset from the current value of the program counter — not a simple thing to calculate. VICMON lets you specify an *address* for a relative branch, and itself calculates the proper offset. As I mention below, however, a good monitor could do even better.

You can save a program by using the "S" command, and load it with the "L" command; no need to rely on PEEKS and POKES. Unfortunately, there is no VERIFY function which would let you know whether the program was correctly stored.

VICMON demands little memory. It comes on a ROM plug-in module, and so itself takes none of your valuable RAM. VICMON does, however, like to use 256 bytes for its own purposes so as not to upset the contents of page one in memory.

Writing a program is easy. Debugging it is the difficult part. VICMON includes a disassembler and has commands that display the contents of memory, search for a particular pattern of bytes in memory, and move the contents of one area of memory to another. Its trace facilities allow you to execute one line of code at a time, or to stop execution (and then resume it) at any point.

VICMON's major disadvantage is that it is a line, and not a full, assembler. You cannot use symbolic names. Thus, you cannot write a statement such as BCC LABEL, and somewhere else use LABEL to define a line of code. You must specify an address in all branch instructions; this becomes especially bothersome if you must branch forward, haven't yet written the line of code to which you will branch, and therefore don't know its address. Similarly, the operands of instructions must be in hexadecimal. You cannot use variable names. Even worse, you can't use the computer's arithmetic ability while writing a program. Thus, say you want to store the contents of the accumulator into location 7680+15 (the fourteenth column on the first row of the screen). You can't simply write STA 7680+15; instead, using your head, a conversion table, or some calculator, you must convert 7680+15=7695 into hexadecimal and write STA $1EOF.

The module comes with a small seventeen page manual which is written in Commodorese or some other variant of English. Consider the following lines: "The other method works only on code located in ROM. The other method works in both RAM and ROM." So which is it, RAM, ROM, or both? With a little effort the reader can figure this out, but why should he have to suffer?

Yet these flaws are not fatal, and I find VICMON to be a very useful, easy to use utility. I wouldn't even think of writing a machine language program without it.

Ravings of a Madman

by Tim Parker
Ontario, Canada

"But why a VIC-20?" was the overwhelming response when I bought my machine a year and a half ago. "It's a toy! It has only a twenty-odd character screen, and a memory that's smaller than my calculator!"

Naturally, that response was from those friends I have that use "toys" like Amdahls, IBMs, and PDPs. So I gave up trying to explain. Why climb Mt. Everest? For the challenge!

That was one reason for my choice. Since I first learned to program in BASIC (was it really thirteen years ago?) I have used computers, big and small. Most computing was on the big, as a necessary tool in my research. But when the VIC-20 was announced, I knew I'd want to take a look at it.

True, the screen was small, but that was ideally suited for several purposes, including games. The memory was restricted at the time (memory expansion was not available commercially), but that sharpened my programming skills. And the VIC had a few gimmicks my CP/M system hadn't. Such as a sound generator. And easily addressable screen graphics. I haven't regretted buying it for one second.

Of course, the uses I put my VIC to are not the same as most others. For many, the VIC is their only computer. While it may have been purchased as an alternative to the video game systems, it also serves for any computational purposes the owner wishes. For me, the VIC has been relegated to game design, and the occasional programming development for an article. Most other work is done on my bigger, more versatile system, including writing these columns. The VIC isn't ignored. It's used almost every week for something, and I'm in the process of designing an interface that will connect it to scientific equipment, as an alternative to the far more costly Apple. It has my heartiest approval, and I've recommended it to first time buyers many times.

The introduction of the Commodore and B128 has made the choice even more interesting. Although I have yet to get my greedy mitts on a BX128 for any length of time, the 64 certainly lives up to its promises. (Oh, that sound synthesizer!)

The purpose of this tirade? Most VIC owners that I have met have been almost apologetic about their machine. They seem to feel that as it is so inexpensive, and lacking in the "flash" that Apple and Atari bestow on their machines, that it is an inferior product. Fear not, brave programmers! The VIC is a computer in name and purpose. I have developed a routine for handling hecklers (usually Apple owners) that may be of some help: I load a tape that plays a Back Invention (NO. 8) in three parts through the sound chip, and generates a full color geometric pattern on the screen simultaneously. I then point out that I paid less for the VIC and its tape deck, memory, and expansions altogether, than they paid for a single disk drive. They usually quieten down after that. If not, I get my dog to bite them.

In the last few months quite a bit of attention has been focused on a mathematical curiosity known as Ulam's Conjecture. Most of the interest has been through a group called PPC. (PPC is a non-profit group dedicated to Personal Programmable Calculators/Computers, such as the Hewlett-Packard HP41, Texas Instruments TI59 and TI88, as well as less elaborate machines. Their monthly magazine PPC Journal is a goldmine of interesting programming quirks, techniques, routines and information. For details, write PPC, 2545 W. Camden Place, Santa Ana, CA 92704. Tell them Tim (8944) sent you! End of plug.)

Ulam's Conjecture (Stanislaw Ulam, who is credited with it, didn't originate the thing: its actual origin seems unknown) can be stated in the following way:

For any positive integer N, if even divide by two, if odd multiply by three and add any positive odd integer A, and apply the same rules to the result, will reach a terminal value of a multiple of A.

Translated, assume the value of A to be 1. Take any positive integer, for example 5, and if the number is even, divide by two. If the number is odd, multiply by three and add A, which here is 1. As the first number is 5, the number generated next will be three times five plus one, or 16. The rules are then repeated. As 16 is even, we divide by two, and get 8. This is even, so we divide by two and get 4, and so on until the resultant number is one, which will be the number arrived at on every subsequent series of operations. The Conjecture states that the terminal value will be a multiple of A, and here is one times A, or one.

This Conjecture has been tested by computers for literally millions of integers, and holds true in every case. (An interesting side note is the number of steps required to get to the terminal value, which is something of a curiosity itself.) What makes the Conjecture so interesting to mathematicians and computer people is that it has not been proved! No mathematically sound proof (or disproof) has been offered, and in fact, one gentleman in England is offering a thousand pounds to the originator of such a proof or disproof.

If so inclined, readers are encouraged to write their own programs to solve Ulam's Conjecture for a number. A BASIC program to do this is simple and short, and the results are worth the effort, if only to astound friends. (For more details, read Martin Gardner's June 1972 column in *Scientific American,* or Douglas Hofstadter's *Godel, Escher, Bach,* page 400.)

I occasionally get asked about machine language programming for the Commodore computers. Generally, the questions deal with the advantages and disadvantages of machine language, and more often, the best method to learn such programming.

The first part is easy to answer. Machine language deals with the instruction codes for the microprocessor directly (without having to be "translated" from a language such as BASIC), which for machines such as the VIC-20, Apple, and others, is the 6502. Computers such as the Commodore, and the new B series use a microprocessor that is patterned on the 6502, but with extra instructions.

The advantage to machine language programming is speed. A language such as BASIC requires time for the program itself to decipher the BASIC statements, and address the 6502 in its native language. Obviously, if the microprocessor can be addressed directly, it saves the deciphering time. The difference in speed is very noticable. Game design in BASIC is limited by the program execution time; it takes so many seconds to follow a GOSUB, do whatever is required there, and show the necessary information on the screen. In machine language, the time is usually not a factor. In fact, most machine language games require a time delay loop to enable the action to be slowed down to a pace the player can keep up with!

The disadvantage of machine language, however, is a factor that scares many programmers away. It requires learning a whole new language, that doesn't have the inbuilt ease of use of BASIC. The time required to master machine language programming is quite substantial, and constant practice is usually needed to keep in top programming form.

Luckily for the 6502 user, the instruction set for the microprocessor is not too elaborate. (Compared to other chips, such as the Z80 or Z8000, with many more codes to be memorized.) And this brings us to the latter part of the original question: how to get into machine language. The easiest way is by working with someone who knows the techniques, and is willing to spare the time to demonstrate the process. Failing that, there are a few excellent books available that will ease the neophyte 6502 programmer in with a minimum of hassle.

Probably the best book of them all is *Programming the 6502* by Rodnay Zaks, published by Sybex (No. C202). This book should be available at almost all well stocked computer stores, and many nonspecialist bookstores. The price is reasonable, and the book will quickly become well thumbed as a reference guide. Zaks (who has written books on almost every phase of computer operations) takes the user through an introduction to the archiecture (i.e. the design) of the 6502, and deals with the instruction set in a well developed, fully explained series of chapters.

There are a few other books in the 6502 series by Sybex that continue on by adding applications and programming details. But get *Programming the 6502* first, and read it through. It should help eliminate the aversion to machine language most people seem to develop.

RAM/ROM on the VIC for $2.00

For the price of a switch and three pieces of wire you can modify your RAM cartridges and make them switch selectable as RAM or ROM. Before a discussion on how to accomplish this feat some basic comparisons and understanding might be useful.

RAM - short for Random Access Memory is the type of memory contained in the 3K, 8K and 16K cartridges which are inserted into the VIC to expand the available memory. This type of memory is volatile which means it can be changed at any time (preferably when you want it to be) and when the power is turned off the contents of memory are lost.

ROM - short for Read Only Memory is the type of memory contained in Game cartridges and in the Vic Kernal. This memory is non-volatile which means that it can't be changed and will not be lost even if power it turned off.

The ability to change memory contents in RAM is controlled by a signal called read/write. If the signal is 5 volts or on then the RAM chips are enabled in the read only mode and data can be transferred from the RAM chips to the computer processor. If the signal is 0 volts or off then RAM chips are enabled in the write only mode and data can be transferred from the computer processor to the RAM chips thus changing its contents. This method is referred to as write active low.

Now you know as much about memory chips as I do. I used to know more but my memory is volatile too!

To change RAM into ROM is quite easy. A steady 5 volt power supply on the read/write pin will do it. I did this by:

1. Open the memory cartridge,
2. Cut the trace leading from cartridge contact #17 (top edge)
3. Solder a wire from the cut trace on the contact side to the outside edge of a single pole double throw switch (spdt)
4. Solder a wire from the cut trace on the chip side to the center pole of the spdt switch.
5. Solder a wire from cartridge contact #21 (top edge) to the other outside edge of the spdt switch. This is the 5 volt power source.
6. Notch out the top edge of the plastic case so that the switch fits into it.
7. Put the case back together and you are finished.

With the switch in one position the cartridge will act normally and in the other position it will act like ROM. IMPORTANT NOTE: if power is turned off you will still lose the memory contents. If you have wired in a reset button into your system resetting the system will not destroy the contents of the pseudo ROM.

Why do it? I wanted to develop games which would not run in RAM as they would self destruct if run in RAM. You may want to use it to protect a program from being destroyed on a system reset or for various other reasons.

Big Programs In Your VIC/PET/—64

by Ron Gunn
Livermore, California

Would you like to convert a really big program to run in your PET, Commodore-64, or even what you thought was your little VIC-20? Has your really big storage program run out of room? Don't you really want to finally add those comprehensive instructions to your otherwise crowded program?

If you do then read on, because if you have one of the PET family of machines it is possible to easily put very large programs on your computer, or conversely to leave lots of data space while still preserving all of those editing, instructing, and printing functions. This bit of magic is possible because Commodore has designed their machines to accept overlays; the substitution of one program for another with ALL OF THE OPERATING DATA INTACT.

Commodore Feature

This capability does not exist in most of the other popular microcomputers, so you won't read about this in the general microcomputing literature. It is too specific for even the 6502 magazines. That may be why you haven't heard much about it. It is given with the Commodore line of machines, however, because of the way data are assigned space in available memory.

If you use the simple rules explained here, you will be able to run and use much larger programs than can be fit on any personal TRS-80 or

Apple; and you will do it conveniently and simply. Imagine being able to call in 15K of instructions, a complex 15K program to display, another complex 10K program to print, another complex 12K program to edit, and so on; all on the same 32K machine loaded with 16K of data!

This method works with disk or cassette programs or data. A single 5-minute load of data from cassette can be manipulated with a series of minute long cassette programs, saving an enormous amount of time while providing almost unlimited program versatility. You do not have to input data at all once loaded in. Simply call in the desired overlay, and proceed.

In "The Psychology of Computer Programming", Gerald M. Weinberg said: "Asking for efficiency and adaptability in the same program is like asking for a beautiful and modest wife....we'll probably have to settle for one or the other". You can now get both. Use overlays to seperate the efficient parts of your programs from the 'nice to have' parts. Use different overlays for a regular size printout and a legal size printout. Bring in one OR the other when a printout is needed.

The Rules

Here are the rules for these overlaid programs, all summarized so you can look them up as you use them. We will look at examples of statements used to go from program to program, and will explain all of these rules in detail later in this article.

1. The main program, used to call up the others must be the first loaded into the machine and RUN.

2. This first program must be the largest program of the group.

3. All assigned string variables, like A$= "This is assigned", must be reassigned in any called-in program in which they are used.

4. The beginning statements of the called-in program must be conditioned to accept a warm start with variables already in place.

5. Any FN function used must be redefined in each program where it is used.

That's about it. Using the ideas explored here, you can flit from program to program, using and manipulating the same data base as you go.

Loading

Now let's play with these concepts. They are of no use to you if they don't help you to create programs that do the things you want to do better, faster, and more completely. Let's assume the existence of a set of programs that keep track of Little League scores. The main program is the largest, and it has a menu with the following options:

(T)	team calcs
(I)	indiv calcs
(PT)	print team
(PI)	print indiv
(E)	edit

If you select team calcs, the program will direct flow to the following line:

```
3250 LOAD "TEAM CALCS*", 8
```

This will load TEAM CALCS from disk. Change the 8 to a 1 if you wish to load from tape #1. Program flow will always start from the beginning of the program, and variables collected during the run of the main program would be there when it starts. Upon the completion of the TEAM CALCS program, this line would be found:

```
9950 LOAD "MAIN PROG*", 8
```

The main program will now come back in, and you will be ready for another choice from the menu. The newly calculated data are available to use. The line numbers above can be anything that fits your program, of course.

Your disk calls in the new program by name. The asterisk is a way of using programs that are being developed and numbered. Line 9950 above, for instance, would load a program named MAIN PROGRAM-32F. Cassette will load any program if told to load a null. LOAD " ", 1 would load the next program played on the cassette. LOAD "MAIN", 1 would load only a program starting with MAIN. Any other program would be passed by.

What and Why

What is happening is that the overlaid program is read in where the original program was, but instead of the variables starting at the top of the new program, the pointers from the original program are left, and the variables continue to occupy the same place in memory that they did before the change.

This is the reason that the first program must be the largest. When the pointers are set for variables at the top of the largest program, none of the other programs can reach that top and overwrite them, causing a crash.

The method is smooth. Using a disk, you get a slight sensation of delay when the menu calls another program, but the new screen display comes up within seconds and you are there. The advantage to cassette users is that memory is almost indefinitely expanded, though time is required for the program exchanges. The trip back to the main program is always the longest as it must by definition BE the longest.

Each program is a single working entity, so there is no relationship between the line numbers of one program and the next. You can use a different series of line numbers, or the same ones over again. I have found no difference in years of use of overlays.

String Handling

We have to understand and define string variables in two different ways when using this method of memory expansion. There are variables that are input (from the keyboard, for instance) and then there are string variables that are assigned with a program statement using the equals sign: (A$="I will appear when you print A$").

When you input a string variable, it is put away in a safe place with the other little variables and we don't have to worry our heads about it.

When it is assigned, however, a different thing happens and we do have to worry about it.

The PET (VIC, —64), in order to conserve space and not have identical string variables hanging out to dry all over the place, simply remembers where in the text of the program the assigned string variable is, and then goes back there whenever it has to have that information. If you do away with that program, however, then that information isn't there to look up any more. If you are going to use an assigned string in any program that you call in, you have to tell the computer where it is now. That means a new assign statement.

There are two cases where you can ignore this step. First, if you aren't going to use the variable in a particular overlaid program, you don't have to bother with it at all. When the main program comes back in, the assigned string information will be right back where it was, and the PET will never know it was gone.

The second case is where you do your assigns at the end of the main program. I have found (aha!) that if you do not overlay programs up to the point where the assign is lurking, then it can still be referred, as it is still there. If you do this, you have to be careful not to get too carried away with the size of your other programs. If one gets large enough to overwrite the assigns some day, your previously useful assigned strings will suddenly start to look like random pieces of program statements.

Warm Starts

Each overlaid program starts at its beginning when loaded in. It is already running. Let's go back to our Little League program set and say that the 'team print' program can run by itself; that is it can input data and print as well as overlay and print. When it is run by itself, the operator is immediately asked the question "DATA DATE".

When this program is run as an overlay, the data are already there, you don't want to give a date. You want to skip over the "DATA DATE" question when the program is run as an overlay.

If T is the number of teams in the league, then whenever data are loaded, T will have a value. A simple IF T THEN GOTO statement will skip into the overlay smoothly and get you directly to the print menu. The program can now be used by itself by reading in data, or can warm start as an overlay with the data already there.

DEF FN

Since use of this user-defineable function depends on going back to see what the definition of the function is, this must be re-defined in each overlay in which it is used, just like the requirement for re-assigning strings. If you follow the same rules outlined above for assigned strings, your defined functions will be there when you need them.

Overlaid Programs

It is really surprising what you can get away with in an overlaid program. As long as it is properly called up by the larger main program, you can even do all of your DIMensioning in a 'start up' program. Variables can be manipulated, and floating point, integer, and string arrays can be operated on just as if you were in the main program. The only real restriction is in assigned string variables, whether in an array or not. If they are defined from the keyboard or a data read, they are OK. If they are assigned with the = operator, then they do not survive the overlay.

Try this out and you'll always be able to use it when you need it. Francis Bacon said: "Knowledge is power". In this case that's computer power.

```
PROGRAM 1.

1 REM"01234567890123456789012345678901 2345"
2IF PEEK(1031)=48 GOTO N

10SYS(1056):GOTO M

N DIM..........
M [FIRST LINE OF PROGRAM]

999GOSUB 1000:SYS(1037):END
1000REM-----MACHINE LANGUAGE LOAD
1010DATA162,6,181,123,157,6,4,202,208
1020DATA248,165,128,133,124,165,129
1030DATA133,125,96,162,6,189,6,4,149
1040DATA123,202,208,248,96
1050FOR I=1037 TO 1066
1060READ XX:POKE I,XX
1070NEXT:RETURN
```

PROGRAM 2.

```
040D A2 06              LDX    #06    :
040F B5 7B      STORE   LDA,X  7B     : LOOP MOVES 6 POINTER
0411 9D 06 04           STA,X  0406   :     BYTES FROM LOCATIONS
0414 CA                 DEX           :     $7C - $81 TO LOCATIONS
0415 DO F8              BNE    STORE  :     $0407 - $040C
0417 A5 80              LDA    80     :  MOVE TWO
0419 85 7C              STA    7C     :      BYTE POINTER
041B A5 81              LDA    81     :      FROM $80/$81
041D 85 7D              STA    7D     :      TO $7C/$7D
041F 60                 RTS           :
0420 A2 06              LDX    #06    :
0422 BD 06 04   LOAD    LDA,X  0406   :  LOOP MOVES 6 POINTER
0425 95 7B              STA,X  7B     :      BYTES FROM LOCATIONS
0427 CA                 DEX           :      $0407 - 040C BACK TO
0428 DO F8              BNE    LOAD   :      LOCATIONS $7C - $81
042A 60                 RTS           :
```

PROGRAM 3.

```
1 REM"012345678901234567890123456789012345"
2IF PEEK(1031)=48 GOTO 20
REM
REM----THIS PROGRAM RECORDS SCORES
REM    AND COMPUTES AVERAGES FOR
REM    FOUR PLAYERS FOR FIVE WEEKS
REM
10SYS(1056):GOTO 26
20DIM SC(5,4,3),ST(4),AV(4)
25PL$(1)="JOHN"
26PL$(2)="TOM"
27PL$(3)="DICK"
28PL$(4)="HARRY"
30WK=WK+1
40IF WK=5 THEN STOP
50PRINT"{CLEAR}WEEK NUMBER";WK
60FOR P=1 TO 4
70PRINT"{2DOWN}INPUT SCORES FOR ";PL$(P)
80FOR G=1 TO 3
90PRINT"GAME";G
100NPUT L
110SC(WK,P,G)=L
120ST(P)=ST(P)+L
130NEXT G
140AV(P)=ST(P)/(WK*3)
150PRINT"{DOWN}";PL$(P);"'S AVERAGE";AV(P)
160NEXT P
998PRINT"{3DOWN}REMEMBER TO SAVE
999GOSUB 1000:SYS(1037):END
1000REM-----MACHINE LANGUAGE LOAD
1010DATA162,6,181,123,157,6,4,202,208
1020DATA248,165,128,133,124,165,129
1030DATA133,125,96,162,6,189,6,4,149
1040DATA123,202,208,248,96
1050FOR I=1037 TO 1066
1060READ XX:POKE I,XX
1070NEXT:RETURN
```

Enterprise

by Tim Parker
Ontario, Canada

One of the most popular types of computer games is the genre patterned on the television series **Star Trek**. Versions exist for almost every type of programmable machine, ranging from hand held calculators to super computers. These games generally fit perfectly into the small home computer field, as the home computer can employ graphics and sound, along with cursor manipulations that are seldom found on larger systems. The trade-off, of course, is memory.

The VIC series provides an interesting challenge for a programmer who wishes to creat a TREK game. The small screen size of the VIC-20 is ideally suited to a game of this sort, allowing graphic block positioning. The 64, with its larger screen (and memory), can make use of the extra space for more involved displays. Although a version can be written to fit in an unexpanded VIC-20 with a 3.5k of memory, the limitations are quite severe. However, with an extra 3k of memory, a fairly good program can be achieved.

This article deals with the game ENTERPRISE, which fits nicely in the 6.5k of an expanded VIC-20. By changing a few memory screen pointers, the program will also run with more memory, or using the 64. By trimming out the "flash", a pared down unexpanded version can be used. (It can be modified to run on PETs, and other computers by changing color controls, and in some cases, cursor control characters. Memory location pointers will also require changing, but this is primarily a text based game, so conversion will be fairly simple.)

In order to utilize memory in the most efficient manner, and to enable fast program execution, screen displays are generated using POKE commands. The primary saving here is in not having to use a matrix for the screen display. (A 10x10 matrix can consume over half a kilobyte of memory.)

By cutting down on the number of text statements, further memory saving is achieved. While the game may lack some of the features of the 40k (and up) Trek versions for computers such as the Apple, it includes many of the extras that are most interesting.

Generation of most of the game factors is controlled by random number generators. This is initialed by reference to the built in clock function.

Color is used to separate the screen into logical sections, and to add a little bit of variety to the display. Sound programming in this game was constructed for the VIC-20's sound registers. These can be converted to the 64's more involved sound device by reference to the manual. Alternatively, sound can be omitted, although the saving in terms of memory space is minimal.

When RUN, the screen display is shown in the upper left. A matrix grid is labelled by the numbers 0 to 9 on the top and left axes. These correspond to the X and Y coordinates respectively. The video display uses an asterisk to represent a star, a "K" for the Klingons, and an "E" for the Enterprise. If a starbase exists, it is represented by a "B". (If extra memory is available, custom characters can be generated for these items.)

The upper right shows the Enterprise's status at all times. The Universe of the game is a 3x3 matrix, giving nine quadrants. The current quadrant is shown at the top of the status display.

Below the quadrant, the ship's functions are all represented by efficiency ratings from 0 to 99. They are ENGY (energy), SHLD (shield), COMP (computer), SCAN (scanner), PHSR (phaser), TORP (photon torpedoes), IMPL (impulse engines) and WARP (warp engines). After any action, these ratings are updated.

On each turn, there are three primary commands that can be executed: COMPUTER, MOVE or FIRE. MOVE allows either impulse movement (within the current quadrant) or warp movement (to another quadrant) as long as the respective ratings are not zero.

FIRE will target either photon torpedoes or phasers on a specified coordinate. As the firing is computer controlled, there is little dependance on distance to target within a quadrant.

COMPUTER allows several different functions to be carried out. A scanner will give the number of enemy ships and starbases remaining in the universe. Self destruct does exactly that. (To be used only in the face of overwhelming odds, or by masochists.) The SET command allows the ratings of any of the status functions to be raised or lowered. (Except computer and scanner—these are considered to be unfixable.) If a function is lowered, the excess units are diverted to ENGY (think of them as auxiliary batteries) up to a maximum of 99. Note that any energy above 99 is lost! If a function is raised, the required units are diverted from ENGY, to a minimum of zero.

Strategy is a matter of personal preferences. At each quadrant where there are enemy ships, they will take shots at the Enterprise on any FIRE or MOVE command. (As computer commands are considered to be carried out almost simultaneously, the enemy does not fire at you when the computer is used.)

There is the possibility of the enemy calling in reinforcements if the battle goes poorly for them, or if they sense victory.

All hits by the enemy are deducted from the shield rating. If this drops too low, a "red alert" warning is issued. At this point, reinforcement of the shields is highly advisable. If the shield rating drops to zero, enemy damage is deducted from ENGY, with the further possibility of internal damage to the other functions. If the computer drops to zero, or both shields and energy are zero, then the game is over, and the enemy is triumphant.

Computer rating drops when a weapon is fired, as it acts as a targeting device. The scanner drops when it is used. (If the scanner rating is zero, the scanner is inoperational.) If weapons or engines fall to zero, they are useless until the computer is used to increase their rating.

Impulse engines are of little use in combat, as distance to enemy is not a factor in hit probability. The impulse engines are primarily used to manoeuvre to starbases for refuelling. A wise captain uses the impulse engines as a battery backup for energy and shield.

If a starbase is landed on, the ratings of the ship all increase to maximum, except for the computer. Using a starbase to refuel will affect your performance rating at the end of the game, when a numerical score will be given that reflects the number of enemy destroyed, and the Enterprise's status at game conclusion.

The program is designed in a modular method, allowing quick modification and reference to any sections. Initiallization and enemy distribution is established in the first few lines. The control loop is lines 500-700, which direct further branching to any relevant subroutines.

The program is broken down as follows:

70-90 Klingon distribution and base location
300-500 POKEing of K, E and B onto grid
500-700 Control loop
1000 Refuelling

Most of the variable's functions are easily identified from their context. Line length has been limited in most cases to one statement per line to simplify programming and debugging. The major exception is the IF...THEN statement. All statements after the IF...THEN are executed if the conditional is true, allowing the avoidance of multiple IFs or subloops. For byte savers, the number of statements per line can be increased, thereby saving several bytes for each line number omitted.

The difficulty of the game can be changed in line 90, where KT refers to the total number of Klingons. Increasing the maximum from 11 will increase the difficulty. The hit probability is given in line 3480. Decreasing the number in the FNA (X) statement will decrease the difficulty.

As the variables are in this listing, the game is relatively easy to win once a strategy has been determined as effective. There is however always the possibility of a few damaging hits from the Klingons that totally ruin shields. That's when things get very interesting!

When using ENTERPRISE on a VIC-20 with more memory, the screen pokes have to be changed. Line 60 defines the location of the memory locations for these screen pokes. As set with the value 7724, the program will run on unaltered and 3k expanded VICs. (The memory map actually starts at 7680; the extra 44 positions the cursor at the start of the display grid. Color starts at 37888.) With 8k or more, substitute the value 4140. (The screen memory starts at 4096, and the color memory at 37888.)

2000-2999 Movement (Warp and Impulse)
3000-3999 Fire control
4000-4999 Computer functions
5000-5999 Red Alert loop
6000-6999 Introductory loop
7000-7599 Klingon fire loop
7600-7999 Reinforcement loop
8000-8999 Victory loop
9000-9999 Destruction loop
10000-10999 Status and screen grid
11000-11999 Cursor positioning
12000-12999 Coordinate loop

With extra memory, the program can be expanded in ways that are limited only by the programmer's imagination. The 3x3 universe can be expanded, as can the quadrants. Different enemy classes can be incorporated, and a more sophisticated targeting and hit algorithm can be used. The "fog of war" can be imitated by adding limited intelligence as the scanner and computer ratings drop. Mutinies, shuttlecraft, drones and many other features can add to the complication of the game.

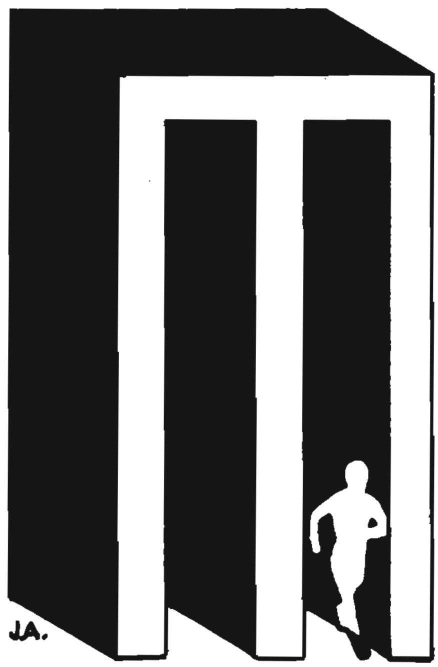
On the output side, the sound can be altered (especially with the 64's exceptional sound generators), and use can be made of multiple screens. Animation is possible for the ambitious.

Although several commercial versions of Trek games are now available, this program was designed from scratch to conform to the author's concepts. Comparison with the commercial products has shown areas where it is deficient, and where it excells. Placing the game in a magazine such as this ensures that people will have a choice, and do not have to pay inflated software prices. Enjoy, and may the Klingons all be cowards!

Game—CONTEST

Escape MCP™

Introduced in late summer of this year by Comm*Data Computer House Inc., ESCAPE utilizes the full capacity of the VIC-20. Written in full machine code, requiring no additional memory and distributed on tape with a joystick option, ESCAPE MCP offers a simple challenge with no simple solution. In the program you have been de-atomized and teleported into the logic circuits of your computer. With you is the sinister MAIN CONTROL PROGRAM which wants to capture and destroy you. As you make your escape through the circuits, you discover that MCP chases after you, right through the walls! Try and try again as MCP taunts you with music and unbelieveable changing circuits. But you'll make it. After all you have speed and intelligence on your side! Besides your pursuer is Evil but still only a program. Make it through nine levels of logic and ESCAPE MCP. $14.95



The Game Contest is a continuing feature of Commander magazine aimed at providing entertainment for and promoting competition among our readers.

The Comm-Data company has graciously provided us with a great game and some super prizes for our Premier Contest.

Don't be the last one on your block to buy Escape MCP and beat the maze.

DEADLINE FOR ENTRIES: 1 FEBRUARY 1983

Escape MCP may be purchased from COMM-DATA or anyone of its fine dealers.

P.O. Box 325
Milford, Michigan 48042
1-313-685-0113

Terms for the Premier Issue Contest

First prize will be awarded to the first person who successfully passes through all nine levels of Escape MCP. The winning entry must contain a photograph of the final screen of the game, an Escape MCP package front, proof of purchase slip and the correct name of the final tune played.

Entries must be mailed to COMMANDER , Escape Contest, P.O.Box 98827, Tacoma, Washington 98498. All entries must be mailed, as postmarks are required to determine the earliest winning entry. In the event of a tie duplicate prizes will be awarded. Employees of Comm*Data and their families may not participate. First prize will consist of Comm*Data VIC-20 Software, valued at $200.00. Second prize will consist of Comm*Data VIC-20 Software, valued at $100.00. Third prize will consist of Comm*Data VIC-20 Software, valued at $50.00.

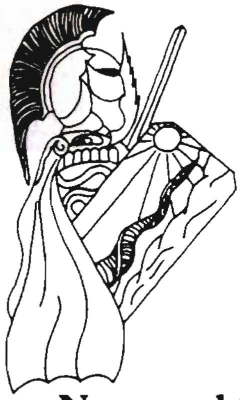
The contest will run until a first prize is awarded. Comm*Data will notify Commander Magazine of the winner(s) and provide copy and photographs for a follow-up story.

Hints from the Commander

Zenith televisions and VIC's can be made compatible by typing in " POKE 3684, 133 " and pressing return. Zenith owners who wish to run Comm*Data's software must type in the poke **over** the " 3583 BYTES FREE" line that appears when the VIC is first turned on. The screen should look like this:

```
****CBM BASIC V2****
POKE 3684, 133
READY
LOAD
```

SUBSCRIBE TO



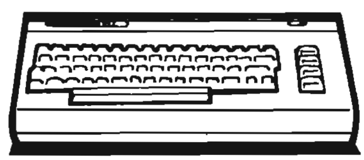
Commander

Now, and take advantage of our Charter Subscriber Discount of $4 OFF

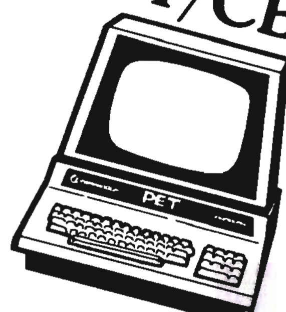
THE MONTHLY JOURNAL FOR COMMODORE COMPUTER USERS

SUPER PET

VIC - 20



PET/CBM



64

"COMMANDER will be dedicated to communicating the fun of, as well as the latest information about the **COMMODORE COMPUTERS.**"

EACH MONTH COMMANDER WILL HAVE:

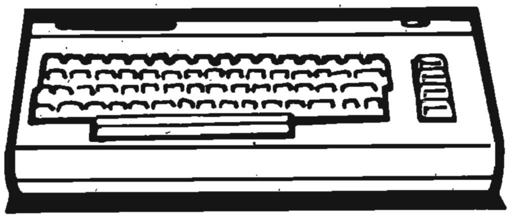
-the latest information and news releases
-software for education, business and fun
-reviews on hardware and software
-program listings
-application (how-to) articles
-a contest and MUCH, MUCH MORE!!

DON'T MISS OUT

on the most informative magazine dedicated to the COMMODORE COMPUTERS

HAVE YOU GOT WHAT IT TAKES TO BE A

Commander?



64

Peek & Poke

by George R. Gaukel
Tacoma, Washington

Peek & Poke # 1

The 'SYS' Command and Cursor Control

The C-64 USER'S GUIDE states that the 'SYS' command does not allow parameter passing. This is true only if you consider a basic variable a parameter. The 'SYS' command is structured to allow pre-loading the processor chip registers prior to executing the specified call and to save the registers upon return.

There are four register storage locations used by the 'SYS' command.

#780 ($030C) (A) Accumlator
#781 ($030D) (X) X Register
#782 ($030E) (Y) Y Register
#783 ($030F) (P) Processor Status

These locations can be poked with the desired value prior to a 'SYS' call. When a return to Basic is accomplished, by a 'RTS', the processor registers are saved in these locations.

This processor interface with Basic is very useful because it allows the programer direct access to some of the resident ROM routines which require the processor registers to contain the subroutine variables. The integer values (0-255) for the processor registers may be stored in Basic variables or arrays.

The routine at address #65520 ($FFF0) is used for reading or updating the current cursor position.
(X) = Row Position (Vertical)
(Y) = Line Position (Horizontal)
(P) = Carry Set-Read Position
(P) = Carry Clear—Write Position
WRITE NEW CURSOR POSITION

```
READY.

 100 REM 'KSCOPE'
 110 :
 120 CLR:RESTORE
 130 :
 140 REM ** SCREEN TO BLACK
 150 :
 160 POKE53280,0:POKE53281,0
 170 :
 180 PRINTCHR$(147);
 190 :
 200 DIM CHAR(25,4),CL(16)
 210 :
 220 FOR I=1TO24
 230 FOR J=1TO4:READ CHAR(I,J):NEXT J
 240 NEXT I
 250 :
 260 FOR J=1TO16:READ CL(J):NEXT J
 270 :
 280 FOR J=1TO4:READ X(J):NEXT J
 290 :
 300 FOR J=1TO4:READ Y(J):NEXT J
 310 :
 320 REM ***** GENERATE NEW VALUES ****
 330 :
 340 HX=INT(RND(1)*20)  :REM 0-19
 350 VY=INT(RND(1)*12)  :REM 0-11
 360 CI=INT(RND(1)*24)+1  :REM 1-23
 370 CP=INT(RND(1)*16)+1  :REM 1-16
 380 :
 390 REM ** CHANGE CURRENT COLOR
 400 :
 410 PRINTCHR$(146)CHR$(CL(CP));
 420 :
 430 REM ** PRINT THE FOUR CHARS
 440 :
 450 FORI=1TO4:XX=X(I):YY=Y(I)
 460 IF XX=20 THEN XX=XX+HX
 470 IF XX=19 THEN XX=XX-HX
 480 IF YY=12 THEN YY=YY+VY
 490 IF YY=11 THEN YY=YY-VY
```

RX=781:RY=782:RP=783
CALL=65520
POKE RP, O : REM CLEAR CARRY
POKE RX,ROW : POKE RY,LINE
SYS CALL
READ CURRENT CURSOR POSITION
RX=781:RY=782:RP=783
CALL =65520
POKE RP,1 : REM SET CARRY
SYS CALL
ROW=PEEK(RX) : LINE=PEEK(RY)

These two routines give the programmer effective X, Y control of the cursor for input positioning or graphics.

A 'SYS' call to the standard jump table in ROM is used, any programs that use these routines will be useable if the ROM is revised (Commodore is famous for ROM revision). Also, these routines should be useable on the VIC.

In the C-64 there are two effective cursors. The video memory cursor we see blinking if we are in the Command Mode or during an 'INPUT' statement. The other cursor tracks with the video cursor and points to the current color nibble. The routine called will update both cursors and return the values for the video cursor.

The kaleidoscope program illustrates the use of direct cursor control using the 'SYS' command. The program design is for ease of keyboard entry and not speed. The characters that are to be printed with reverse on, are given a negative value in the data table. The positive values are preceded by a reverse off character.

Peek & Poke #2

Repeat Key Feature

When the C-64 is turned on location #650 ($028A) contains a zero. If this location is poked with the value 128, then the repeat key function will work for all keys.

POKE 650,128 : REM REPEAT ALL KEYS

POKE 650,0 : REM REPEAT ONLY CURSOR AND SPACE KEYS

This is handy if you are editing graphic routines which require long strings of the same type character.

```
500 POKE 781,YY :POKE 782,XX
510 POKE 783,0 :SYS 65520
520 ZC=CHAR(CI,I)
530 IF ZC<0 THEN PRINTCHR$(18);
540 IF ZC>0 THEN PRINTCHR$(146);
550 ZC=ABS(ZC)
560 PRINTCHR$(ZC);
570 :
580 NEXT
590 :
600 GOTO340
610 :
620 REM ***** CHARACTER VALUES *******
630 REM ** NEGATIVE VALUE INDICATES
640 REM ** CHAR REVERSED
650 :
660 DATA   32,  32,  32,  32
670 DATA  -32, -32, -32, -32
680 DATA   32,  32,  32,  32
690 DATA  -32, -32, -32, -32
700 DATA   32,  32,  32,  32
710 DATA  -32, -32, -32, -32
720 DATA  161,-161,-161, 161
730 DATA -161, 161, 161,-161
740 DATA  162, 162,-162,-162
750 DATA -162,-162, 162, 162
760 DATA  169, 223,-169,-223
770 DATA -169,-223, 169, 223
780 DATA  223, 169,-223,-169
790 DATA -223,-169, 223, 169
800 DATA  172, 187, 190, 188
810 DATA -172,-187,-190,-188
820 DATA  188, 190, 187, 172
830 DATA -188,-190,-187,-172
840 DATA  187, 172, 188, 190
850 DATA -187,-172,-188,-190
860 DATA  190, 188, 172, 187
870 DATA -190,-188,-172,-187
880 DATA  191,-191, 191,-191
890 DATA -191, 191,-191, 191
900 :
910 REM ***** COLOR DATA *************
920 :
930 DATA 144,  5, 28,159
940 DATA 156, 30, 31,158
950 DATA 129,149,150,151
960 DATA 152,153,154,155
970 :
980 REM ***** COORDINATES ************
990 :
1000 DATA  20, 19, 19, 20 :REM X
1010 DATA  11, 11, 12, 12 :REM Y
READY.
```

```
100 REM 'JOY2'
110 GOSUB 380 : REM INITALIZE
120 :
130 GOSUB 330 : REM READ JOYSTICK
140 :
150 YY = YY+UD(JD)
160 XX = XX+LR(JD)
170 IF YY < 0 THEN YY = 24
180 IF YY > 24 THEN YY = 0
190 IF XX < 0 THEN XX = 39
200 IF XX > 39 THEN XX = 0
210 POKE RP,0 : REM CLR CARRY
220 POKE RX,YY
230 POKE RY,XX
240 IF ((XX = 39) AND (YY = 24)) THEN 130
250 SYS CU : REM LOG CURSOR
260 PRINT CH$; : REM DON'T FORGET SEMICOLON
270 IF BT = 16 THEN 300 : REM NO BUTTON
280 FOR TT = 1 TO 6 : NEXT : REM WAIT FOR RASTER
290 PRINTCHR$(157)CHR$(32); : REM CURSOR LEFT-BLANK
300 GET IN$: IF IN$ <> "" THEN CH$ = IN$
310 GOTO130
320 :
330 JP = PEEK(J1) : REM READ JOY1
340 BT = JP AND 16 : REM GET BUTTON VALUE
350 JD = 15-(JP AND 15) : REM GET DIRECTION
360 RETURN
370 :
380 PRINTCHR$(147) : REM CLR SCREEN
390 :
400 FOR K = 0 TO 10
410 READ UD(K) , LR(K)
420 NEXT
430 :
440 CU = 65520 : J1 = 56320
450 POKE 53280,0 : REM BLACK SCREEN
460 POKE 53281,0 : REM & BORDER
470 XX=20 : YY= 12 : REM ABOUT CENTER
480 CH$ = CHR$(113)
490 RX=781 : RY=782 : RP=783 : REM REGISTERS
500 RETURN
510 :
520 REM DIRECTION UP-DOWN,LEFT-RIGHT
530 DATA  0 , 0
540 DATA -1 , 0 : REM UP
550 DATA  1 , 0 : REM DOWN
560 DATA  0 , 0
570 DATA  0 ,-1 : REM LEFT
580 DATA -1 ,-1 : REM UP-LEFT
590 DATA  1 ,-1 : REM DOWN-LEFT
600 DATA  0 , 0
610 DATA  0 , 1 : REM RIGHT
620 DATA -1 , 1 : REM UP-RIGHT
630 DATA  1 , 1 : REM DOWN-RIGHT
READY.
```

Peek & Poke #3

Device Numbers Redefined

Page 122 of the C-64 user's guide has an error. The following notes should be added:

Device 0 = KEYBOARD
Device 2 = RS-232
Device 3 = SCREEN

If you are loading a machine language program from a cassette, a secondary address of 3-15 should be used.

LOAD"EXMON.C",1,9
LOAD"",1,5

If the secondary address is not given, it defaults to zero. This tells the C-64 to load a basic program starting at address #2048 ($0800). The results are not very nice.

Peek & Poke #4

The Atari Joystick and the C-64

The program 'JOY2' demonstrates the use of the Atari joystick for cusor positioning. The program leaves a track on the screen of user selected colors and characters. Pressing the fire button activates an erase mode.

JOYSTICK PORT2 is used for input as this also allows keyboard input. If JOYSTICK PORT1 is used the 'GET' command in the program will be very unstable as the same address latch is used by the system for both functions.

JOYSTICK PORT2 #56320 $DC00
JOYSTICK PORT1 #56321 $DC01

The program as written, allows for screen wraparound or an open universe. To close the screen boundries, make the following changes:

LINE 170 ...THEN YY=0
LINE 180 ...THEN YY=24
LINE 190 ...THEN XX=0
LINE 200 ...THEN XX=39

If you wish to adapt the routine to a game, the cursor logs of all moveable objects can be kept in a pair of variables and the current position can be blanked prior to moving to the next position.

X1=(INT(3*RND(1)))-1

This statement will provide random values of -1, 0 and 1 for moving an

```
100 REM 'BLOCKS'
110 :
120 GOTO400
130 :
140 REM LOG CURSOR
150 POKE 781,YY : POKE 782,XX
160 POKE 783,0
170 SYS 65520
180 RETURN
190 :
200 REM LEFT JOYSTICK
210 JL= PEEK(J1) : XY = 15-(JL AND 15)
220 X = JX(XY) : Y = JY(XY)
230 TF=(X OR Y)
240 RETURN
250 :
260 REM RIGHT JOYSTICK
270 JR= PEEK(J2) : XY = 15-(JR AND 15)
280 X = JX(XY) : Y = JY(XY)
290 TF=(X OR Y)
300 RETURN
310 :
320 REM TEST MOVE
330 IF (XX<0 OR XX>39) THEN WI=1
340 IF (YY<0 OR YY>24) THEN WI=1
350 IF (XX=39 AND YY=24) THEN ZP=1
360 PV=(YY*40)+XX+VM : PK=PEEK(PV)
370 IF PK<>32 THEN WI=1
380 RETURN
390 :
400 GOSUB 1550 : REM INITALIZE PGM
410 :
420 GOSUB 1280 : REM INITALIZE START
430 :
440 REM LEFT PLAY
450 P=0 : GOSUB 210
460 IF TF = 0 THEN 480
470 XL=X : YL=Y : LL=XY
480 XX=LX : YY=LY
490 GOSUB 150
500 XX=XX+XL : YY=YY+YL
510 GOSUB 330
520 IF WI=1 THEN 1050
530 IF ZR=0 THEN 560
540 POKE 2023,WA
550 ZR=0 : GOTO570
560 PRINT CL$WA$;
570 LX=XX : LY=YY
580 GOSUB 150
590 IF ZP THEN 620
```

object randomly on the screen. The routine should be called twice; once for the vertical value and once for the horizontal value. Before moving we need to see if the new location is blank or already occupied by some character.

T1=PEEK(YY*40)+XX+1024)

This routine will convert the X, Y values to an address which can be tested to see if the space is blank or contains something designated as an obstical, wall, or some moveable object such as a monster or treasure.

The bottom right-hand corner is not used in the 'JOY2' program. Line 230 prevents this. This location cannot be printed without causing a scroll. The location can be poked if it is necessary to fill it in (also poke the corresponding color address).

The second program 'BLOCKS' demonstrates the use of both joysticks and sound in a two player game. Both players leave a wall of blocks behind them. The object is not to go off the screen or run into any blocks or the other player.

Peek & Poke #5

ROM or RAM???

```
FOR I=40960 TO 49151 :REM $A000
  TO $BFFF
A=PEEK(I) : POKE I,A
NEXT
A=PEEK(1) : A=A AND 254 OR 2
POKE 1,A : REM ENABLE LO RAM
```

This routine copies the basic ROM to RAM then switches from ROM to RAM. When the C-64 is turned on, the memory is configured for the basic and kernal ROMs. These ROM banks can be selectively disabled and replaced by RAM. To switch the basic ROM back in, 'OR' 1 to the value at address #01.

While the ROM is active, a write to the ROM address actually writes to the masked RAM. This is why the value read in the above routine is poked back to the same address.

```
600 PRINT CL$TK$;
610 GOTO 660
620 POKE 2023,TK : POKE 56295,CL
630 ZR=1:ZP=0
640 :
650 REM SOUND
660 POKE V1,16 : POKE V3,16
670 POKE A1,9 : POKE S1,9
680 POKE A3,9 : POKE S3,9
690 T1=INT(LL/2) : T2 = INT(RR/2)
700 POKE F1,FA(T1) : POKE F2,FB(T1)
710 POKE F5,FE(T1,T2):POKEF6,FF(T1,T2)
720 POKE V1,17 : POKE V3,17
730 :
740 :
750 REM RIGHT PLAY
760 P=1 : GOSUB 270
770 IF TF = 0 THEN 790
780 XR=X : YR=Y : RR=XY
790 XX=RX : YY=RY
800 GOSUB 150
810 XX=XX+XR : YY=YY+YR
820 GOSUB 330
830 IF WI=1 THEN 1050
840 IF ZL=0 THEN 870
850 POKE 2023,WA
860 ZL=0 : GOTO 880
870 PRINT CR$WA$;
880 RX=XX : RY=YY
890 GOSUB 150
900 IF ZP THEN 930
910 PRINT CR$TK$;
920 GOTO 960
930 POKE 2023,TK : POKE 56295,CR
940 ZL=1:ZP=0
950 REM SOUND
960 POKE V2,16 : POKE V3,16
970 POKE A2,9 : POKE S2,9
980 POKE A3,9 : POKE S3,9
990 T1=INT(LL/2) : T2 = INT(RR/2)
1000 POKE F3,FC(T2) : POKE F4,FD(T2)
1010 POKE F5,FE(T1,T2):POKEF6,FF(T1,T2)
1020 POKE V2,17 : POKE V3,17 : GOTO450
1030 :
1040 REM WIPEOUT
1050 SX=VAL(TI$)
1060 POKE53280,1
1070 IFP=0 THEN PV=(LY*40)+LX+VM
1080 IFP=1 THEN PV=(RY*40)+RX+VM
1090 JJ=20
1100 POKEV1,0:POKEV2,0:POKEV3,0
1110 POKE F1,136 : POKE F2,19
1120 POKE F3,236 : POKE F4,19
```

```
FOR I=40960 TO 49151 :REM $A000
  TO $BFFF
FOR J=57344 TO 65535 :REM $E000
  TO $FFFF
A=PEEK(I) : POKE I,A
A=PEEK(J) : POKE J,A
NEXT : NEXT
A=PEEK(1) : A=A AND 252 OR 1
POKE 1,A : REM ENABLE LO & HI
  RAM
```

This routine will switch both the Basic and kernal ROM to RAM. The actual switching is done by changing Bit0 and Bit1 in the 6510 port.

BIT1	BIT0	
1	1	LO ROM & HI ROM
1	0	LO RAM & HI ROM
0	1	LO RAM & HI RAM
0	0	ALL RAM

If we change to the all RAM mode, this will also include the addresses in the area of $D000 to $DFFF. If this area is switched we will lose the port latches and character generator ROM which will crash the system if done from Basic.

Bit2 at address #01 is used to toggle the character ROM in and out of memory. This is also best controlled by machine language routines as it appears this ROM is also masked by the port and video chip latches (see 'crash' above).

The rest of the bits at address #01 are used for cassette I/O. There are control lines on the cartridge connector which will give other memory configurations. They are beyond the scope of this article.

The C-64 has the ability to switch major areas of addressable memory from ROM to RAM under software control. This makes it very easy to make machine language changes to Basic and save different versions on tape or disk for later use (A VIC/C-64??). The 8K of Basic ROM can be deleted and the space used by other languages such as FORTH or PASCAL. In time critical applications, the kernal ROM rom can be switched and I/O routines modified or interrupt vectors changed. (COMMODORE—How about an IEEE-488 overlay for the kernal ROM can be switched and I/O routines modified or interrupt

```
1130 POKE F5,180 : POKE F6,20
1140 POKES1,240:POKES2,240:POKES3,240
1150 POKEV1,21:POKEV2,21:POKEV3,21
1160 POKEPV,32
1170 FORII=0TO50:NEXT
1180 POKEPV,86
1190 FORII=0TO50:NEXT
1200 JJ=JJ-1:IFJJ<>0THEN1160
1210 POKEV1,0:POKEV2,0:POKEV3,0
1220 IF P=0 THEN SR=SR+INT(SX+25)
1230 IF P=1 THEN SL=SL+INT(SX+25)
1240 FORII=0TO2000:NEXT
1250 GOTO 420
1260 :
1270 :
1280 PRINTCHR$(147)
1290 XX=17 : YY=8 : GOSUB 150
1300 POKE 53280,0 : POKE 53281,0
1310 PRINTCHR$(18)"BLOCKS"
1320 PRINT:PRINT:PRINT
1330 PRINT" TOTAL SCORE - GAME"GA
1340 PRINT:PRINT
1350 PRINT" LEFT PLAYER  :"SL
1360 PRINT:PRINT
1370 PRINT" RIGHT PLAYER :"SR
1380 GOSUB 1930
1390 TI$="000000"
1400 PRINTCHR$(147)
1410 POKE 53280,9 : POKE 53281,11
1420 POKEV1,0:POKEV2,0:POKEV3,0
1430 POKEVL,15
1440 LX=0 : LY=12 : RX=39 : RY=12
1450 XX=LX : YY=LY : GOSUB 150
1460 GOSUB150
1470 PRINT CL$TK$;
1480 XX=RX : YY=RY : GOSUB 150
1490 GOSUB150
1500 PRINT CR$TK$;
1510 LL=8 : XL=1 : YL=0 : WI=0
1520 RR=4 : XR=-1: YR=0 : GA=GA+1
1530 RETURN
1540 :
1550 REM SETUP DIRECTION ARRAY
1560 FOR K = 0 TO 10
1570 READ JY(K) , JX(K)
1580 NEXT
1590 :
1600 REM SOUND ARRAYS
1610 FOR K = 0 TO 4 :READ FA(K) : NEXT
1620 FOR K = 0 TO 4 :READ FB(K) : NEXT
1630 FOR K = 0 TO 4 :READ FC(K) : NEXT
1640 FOR K = 0 TO 4 :READ FD(K) : NEXT
1650 :
1660 DIM FE(4,4),FF(4,4)
1670 :
1680 FOR KK = 0 TO 4 : FOR K = 0 TO 4
1690 READ FE(KK,K) : NEXT : NEXT
1700 :
```

```
1710 FOR KK = 0 TO 4 : FOR K = 0 TO 4
1720 READ FF(KK,K) : NEXT : NEXT
1730 :
1740 REM CONSTANTS
1750 VM= 1024
1760 J1= 56320 : J2= 56321
1770 F1= 54272 : F2= 54273
1780 F3= 54279 : F4= 54280
1790 F5= 54286 : F6= 54287
1800 A1= 54277 : A2= 54284 : A3= 54291
1810 S1= 54278 : S2= 54285 : S3= 54292
1820 V1= 54276 : V2= 54283 : V3= 54290
1830 VL=54296
1840 :
1850 :
1860 TK=81 : WA=102
1870 CL=13  : CR=14
1880 CL$=CHR$(153) : CR$=CHR$(154)
1890 TK$=CHR$(215)  : WA$=CHR$(166)
1900 GA=0 : SL=0 : SR=0
1910 RETURN
1920 REM WAIT FOR BOTH BUTTONS
1930 JL=PEEK(J1): JR =PEEK(J2)
1940 BL=JL AND 16 : BR =JR  AND 16
1950 IF (BL=0 AND BR=0) THEN RETURN
1960 FORII=1TO50:NEXT
1970 GOTO1930
1980 :
1990 REM DIRECTION UP-DOWN,LEFT-RIGHT
2000 DATA  0 , 0
2010 DATA -1 , 0 : REM UP
2020 DATA  1 , 0 : REM DOWN
2030 DATA  0 , 0
2040 DATA  0 ,-1 : REM LEFT
2050 DATA  0 , 0
2060 DATA  0 , 0
2070 DATA  0 , 0
2080 DATA  0 , 1 : REM RIGHT
2090 DATA  0 , 0
2100 DATA  0 , 0
2110 :
2120 DATA 207, 66, 48, 0, 69
2130 DATA  16, 21, 25, 0, 28
2140 DATA 159, 92, 95, 0,138
2150 DATA  33, 42, 50, 0, 56
2160 :
2170 DATA 104,151,152, 0, 35
2180 DATA  35,104,151, 0,152
2190 DATA 152, 35,104, 0,151
2200 DATA   0,  0,  0, 0,  0
2210 DATA 151,152, 35, 0,104
2220 :
2230 DATA   8, 10, 12, 0, 14
2240 DATA  14,  8, 10, 0, 12
2250 DATA  12, 14,  8, 0, 10
2260 DATA   0,  0,  0, 0,  0
2270 DATA  10, 12, 14, 0,  8
READY.
```

Commodore 64K Memory Expansion

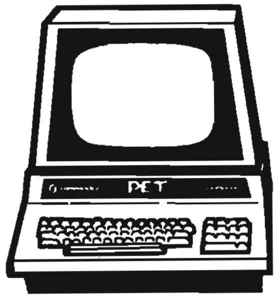
by Neil Omvedt
Roseville, Minnesota

If you have tried to use Visicalc on the Commodore you know that the amount of memory available for spreadsheets is somewhat limited. There is a solution to this for Commodore users. This is the 64k memory expansion board which makes the total available memory 96k.

Installation of the memory expansion is fairly easy. There are two sets of instructions, one in the documentation book and a separate printed she. . I suggest use of the printed sneet. The memory expansion board mounts inside the computer above the rear half of the printed circuit board. The expansion board has its own 6502 processor and you remove the computer's processor and plug in a cable to the memory board. There are two other plug in cables for the power supply. The board does block off access to the ROM sockets for user Rom so if you need to install any of these that should be done first.

The additional memory is not directly accessible to Commodore BASIC, but requires memory mapping. This means that the additional memory is temporarily mapped into normal ROM areas of the Commodore. A diskette of programs is provided which have test programs for the additional memory and sample programs showing how to use it.

Actually the average programmer will probably not have that much use for the additional memory. The real advantage of this product is that it will allow more sophisticated software to be developed for the CBM. At this time several products are on the market which use the additional memory. The only one I had available at this time was the Visicalc program. What a difference there was — my available memory for worksheets went from 11 to 72 when the additional memory was plugged in. Two other products available require the memory expansion for use. One is Word Pro V Plus - the top of the line of the Word Pro word processing programs for the Commodore computers. The other is the UCSD Pascal software system whose availability should increase the amount of software available for the CBM 8096. Also A B Computers of Pennsylvania is advertising a BASIC interpreter which uses the full 96k of memory.



Stock Plot Chart

by Claud E. Cleeton
Bellevue, Washington

This program prints a high-low bar chart of stock prices on a PET. Lines 60-90 determine whether the machine is of the 40 or 80 column type. The unique graphical features of the PET are used to provide vertical resolution to eight parts per line, which for most cases is one-eighth of a point. The reversed space character is used to draw the portion of the bar covering whole price intervals. The fractions are drawn by using a combination of horizontal bars of various heights and their reverse characters. These characters are read into three arrays. The fractions at the high price are set up at line 500 as array HF. The fractions at the low price end are the reverse of HF and are generated at line 600 as array LF. When HF and LF elements are in a single price interval array EF is used, line 700, which merely draws a single bar at the average value location.

The program provides for keyboard entry of data starting at line 2000. The date is entered as a seven character string using two numerical digits for the day, followed by three letters for the month and two digits for the year without punctuation or spaces, line 2030. The price values, H$ for high and L$ for low are entered with one digit to the right of a decimal point corresponding to the number of eighths (0 to 7) in the price fraction. Lines 2060 and 2090 check to see that the left digit of the two right digits are periods. Also line 2110 checks that the value entered as the low is not greater than the value entered as a high.

The average of each high and low is found and the data is averaged to give A, line 2130, which is used in the charting program to establish the ordinate, or price scale, so that the plot is near the midportion of the screen. This midposition is established as Z, lines 3020-3030. The differences between Z and the high values are checked to make sure the price plot will not exceed the screen upper limit, line 3050, and a divisor, Q, is established, line 3070, if necessary, to insure that the prices remain on the screen.

Lines 3100-3120 print ordinate scale values for the stock prices in terms of the central value Z and price divisor Q which are separated by ten lines. Lines 3130-3150 print a horizontal mark (ASCII 100) at the integral value of the designated scale values. The first screen location is 32768. PV is a one for 40 column machines and two for 80 column types, thus line 3130 prints at the fourth position in the second line. The next POKE prints 10 lines lower, etc.

The loop at 3200 prints the date at the bottom of the display. The cursor is HOMEd for each increment of I and line 3210 separates the first digit of the date as A$, the second as B$ and the remainder of the date as M$. If the day of the month is less than 10, the day is printed down 22 lines and for values greater than 10, 21 lines down with the second digit below. When the day of the month decreases (new month) line 3240 prints the month and year immediately above the day. The inner J loop at 3250 prints a vertical row of colons marking the successive values of time (X).

Starting at line 3300, the high-low bars are plotted. Lines 3340 and 3380 separate the integral portion of the price values, change the fractional portion to a decimal fraction, divides by the scale factor Q and finds the integral value which is to be plotted. A one is added to the low integral value since the fractional value extends downward from the continuous bar. The fractional values are found, line 3390, as the number of eighths (0 to 7) for the subscript in the arrays defining the graphic symbols for printing the fractional portions of the bar.

Line 3410 prints the portion of the bar made up of whole price intervals as a reverse space (ASCII 160). Line 3420 adds a graphic character to the lower end of the bar corresponding to one minus the lower price fraction, and line 3430 adds to the top of the bar the fraction that the high price exceeds the integral value. If both the high and low are found in the same price box, line 3395 sends the plotting to line 3500 and a single horizontal mark is printed at the location of the average of the high and low.

```
50 PRINT"{CLEAR}"
60 PV=PEEK(50003)
70 IF PV=160 THEN 90
80 PV=1:GOTO 100
90 PV=2
100 DIM K(35*PV),D(35*PV),D$(35*PV),H$(35*PV),L$(35*PV)
500 DATA 32,100,111,121,98,248,247,227
510 FOR I=0 TO 7
520 READ HF(I):NEXT
600 DATA 160,228,239,249,226,120,119,99
610 FOR I=0 TO 7
620 READ LF(I):NEXT
700 DATA 100,82,70,64,67,68,69,99
710 FOR I=0TO7
720 READ EF(I):NEXT
1000 PRINT TAB(7)"** HIGH-LOW STOCK CHART **"
1010 PRINT"TO INPUT DATA, ENTER 1"
1020 PRINT"TO PLOT CHART, ENTER 2"
1030 PRINT"TO END PROGRAM, ENTER 3"
1040 INPUT I
1050 ON I GOTO 2000,3000,10000
2000 PRINT"ENTER DATE IN FORM 08DEC82";PRINT
2005 PRINT"ENTER PRICE FRACTION AS 1 DECIMAL DIGIT":PRINT
2010 PRINT"WHEN FINISHED, ENTER DONE FOR DATE":PRINT
2020 PRINTTAB(2)"DATE",,"HIGH","LOW":K=1:V=0
2030 PRINTSPC(1);:INPUT D$:D$(K)=D$
2035 IF D$="DONE"GOTO1000
2040 IF LEN(D$)<>7 THEN PRINT"ERROR":GOTO 2030
2050 PRINT"{UP}";SPC(20);:INPUTH$:H$(K)=H$
2060 E$=RIGHT$(H$,2):IF LEFT$(E$(1)="."GOTO 2080
2070 PRINT"ERROR":GOTO 2050
2080 PRINT"{UP}";SPC(30);:INPUT L$:L$(K)=L$
2090 E$=RIGHT$(L$,2):IF LEFT$(E$(1)="."GOTO2110
2100 PRINT"ERROR":GOTO 2080
2110 IF VAL(L$)>VAL(H$)GOTO2070
2120 PRINT"{UP}";D$,,H$;"{6 SPACES}";L$;"{3 SPACES}"
2130 V=(VAL(H$)+VAL(L$))/2+V:A=V/K:K=K+1:GOTO2030
3000 X=5:Q=1
3020 IFA>10THEN Z=INT(A/10)*10:GOTO3040
3030 Z=10
3040 FOR I=1 TO K-1
3050 IF VAL(H$(I))/Q-Z>10 THEN 3070
3060 NEXT I:GOTO 3080
3070 Q=Q*2:I=1:GOTO3040
3080 PRINT"{CLEAR}","TO CONTINUE, HIT RETURN"
3100 PRINT Z+10*Q
3110 PRINT"{9 DOWN}";Z
3120 PRINT"+9 DOWN}";Z-10*Q
3130 POKE 32768+44*PV,100
3140 POKE32768+444*PV,100
3150 POKE 32768+844*PV,100
3200 FOR I=1 TO K-1:PRINT"{HOME}"
3210 A$=LEFT$(D$(I),1):B$=MID$(D$(I),2,1):M$=RIGHT$(D$(I),5)
3220 IF A$="0"THEN PRINT"{22 DOWN";TAB(X);B$:GOTO3240
3230 PRINT"{21 DOWN}";TAB(X);A$:PRINT TAB(X);B$
3240 D(I)=VAL(D$(I)):IF D(I)<D(I-1)THEN PRINT"3 UP"TAB(X);M$
```

```
3250 FOR J=2 TO 20
3260 POKE 32768+X+PV*40*J,58
3270 NEXT J
3280 X=X+1:NEXT I
3290 X=5
3300 FOR I=1 TO K-1
3310 FOR J=1 TO LEN(H$)
3320 IF MID$(H$(I),J,1)="."THEN 3340
3330 NEXT J
3340 H=(VAL(LEFT$(H$(I),J-1))+VAL(RIGHT$(H$(I),1))*.125)
     /Q:H%=INT(H):J=1
3350 FOR J=1 TO LEN(L$)
3360 IF MID$(H$(I),J,1)="."THEN 3380
3370 NEXT J
3380 L=(VAL(LEFT$(L$(I),J-1))+VAL(IGHT$(L$(I),1))*.125)
     /Q:L%=INT(L+1):J=1
3390 F1=INT((H-H%)/.125):F2=INT((L-L%+1)/.125)
3395 IF H%=L% GOTO 3500
3400 FOR J=L% TO H%
3410 POKE 33208+X-40*PV*(J-Z/Q),160:NEXT J
3420 POKE 33208+X-40*PV*(L%-1-Z/Q),LF(F2)
3430 POKE33208+X-40*PV*(H%-Z/Q),HF(F1)
3440 GOTO 3520
3500 F=INT((F1+F2)/2)
3510 POKE 33208+X-40*PV*(H%-Z/Q),EF(F)
3520 X=X+1:NEXT I
5000 GET A$:IFA$=""THEN 5000
5010 GOTO 1000
10000 END
READY.
```

Higher Interest SAVEing

by John R. Sherburne
Fairfax, Virginia

If you are tired of struggling with cassette data files but aren't prepared to get a disk system, cheer up, storing data may be easier than you thought. If the volume of data isn't too large, you can store your data right in the same file as the BASIC program that uses it; no more problems remembering which is the current data tape; no more juggling tapes or waiting while data is read in. And - perhaps best of all - no more wondering if data has been properly saved; now you can use the VERIFY command for data too.

Suppose, for example, you want to write a check book program to keep track of your bank balance. Using a separate data file, you would have to load a BASIC program first, then load the existing balance and other previous data. When finished, you would have to rewrite the data file and, if you wanted to verify, you would need a separate routine to accomplish that. On the other hand, if you were to store data as a part of the program file, the data would be there and ready to go as soon as the program was loaded. There would be no extra files to worry about. When you finished, the normal SAVE and VERIFY commands would take care of everything. As a result, an application that might be too much trouble to bother with using separate data files can be accomplished quickly and simply by using integrated data/program storage. Other possible uses of the technique are: to make it possible to stop a game and then re-start where you left off; to incorporate moving averages, probabilities and other changing statistics into a program; or to keep gas mileage and expense logs.

The key to the process is "tricking" PET into thinking that it's variable storage area is part of the BASIC program. The amount of programming required is modest. The necessary "trickery" is accomplished by two short machine language routines which occupy a total of 30 bytes. These routines, stored in a REM statement at the beginning of the BASIC program, are POKED into place with a small BASIC subroutine and are accessed via the SYS command. Step-by-step, the process goes like this:

Step 1. The BASIC program which will use the data storing technique should first be written and thoroughly debugged. Changing the program later will be more difficult. There are two rules that must be observed in writing the program. First, statement numbers 20 and below should be kept free for statements that will be added in Step 2. Also, all DIM statements must be at the beginning of the program and must precede any other program statements except REMs. The reason for this rule will be explained later.

Step 2. Next, the BASIC statements shown in Program 1 are added to the debugged program. The subroutine in lines 1000 to 1070 POKEs the two machine language routines into the area between the quotes of the first REM statement, replacing the numbers initially there. Line 2 checks to see whether the program has been run before, that is, whether the numbers in the REM statement have been replaced. If not, the GO TO skips around line 10. The n in line 2, therefore, should be replaced with the first line number after line 10. The GO TO in line 10 is necessary because BASIC will not allow a given DIM statement to be executed more than once. DIMs must be executed the first time through, but skipped on later iterations. Thus, the m in line 10 must be replaced with the line number of the next non-DIM statement. All DIM statements must be entered starting with line n and ending before line m. If there are no DIM statements, "GO TO m" may be left out of line 10. The first REM statement should be typed just as shown without extra spaces. A double check can be made by entering the command "?PEEK(1031)". A value of 48 will be returned if the REM is correct.

Step 3. The program must be terminated only through line 999. This line sets up the machine language and executes the first part of the routine. After 999 has been executed, the program should be saved immediately using a normal SAVE command. Thereafter, no reference to program variables can be made without first executing SYS (1056).

Step 4. The program is reloaded using the normal LOAD command. After LOAD and RUN, the program variables will be just as they were before being SAVEd. Variable values can be accessed either through the program or in the direct mode, i.e., with a command such as ?A, B, C.

It is not necessary to understand how the technique works to use it, but knowing how it operates shows a lot about how PET BASIC manages RAM usage.

PET loads BASIC programs in RAM beginning with memory location 1025 (decimal). As the program is run, variables are stored at the end of the BASIC program in the order they are encountered. Array variables are stored separately from simple, non-subscripted variables and PET keeps track of where everything is with a set of pointers (Figure a).

When a SAVE command is executed, the Start of Variables pointer is both entered into the tape file header and passed to the SAVE routine as the end of program indicator. The other pointers are ignored. On LOAD, the Start of Variables pointer value is retrieved from the tape header and all three pointers are set to that value. By moving the Start of Available RAM pointer to the Start of Variables pointer location, the PET can be fooled into considering the variables to be part of the BASIC program and saving them along with the program. At the same time, however, the other pointers must somehow be saved so that on reloading, all three pointers can be restored to their proper values. Both saving and restoring the pointers is accomplished by the short routine which is stored in the REM statement of Program 1 and which is shown in assembly language form in Program 2. Six additional bytes in the REM statement are used to store the three two-byte pointers when SYS (1037) is executed. After the pointers have been saved, the Start of Available RAM pointer value is inserted in the Start of Variables pointer location. On LOAD, a SYS (1056) causes the three pointers to be restored. The whole process is illustrated in Figures b and c.

An example of how the technique can be used is contained in Program 3. The program is a very simple one designed to store scores for a bowling team and to compute weekly averages. Each week the program is loaded, the new scores entered, and the averages recalculated.

A point to remember is that since PET doesn't normally save BASIC variables, some commands will inadvertantly cause problems. Already mentioned is the fact that a DIM statement cannot be encountered twice for the same array. A second problem is that any attempt to add or delete program statements after the first run will confuse the PET and usually results in an error condition.

A few final notes:

The programs above intentionally ignore one PET working area. PET handles strings in two ways. Simple strings such as those in Program 3 are handled by simply using a pointer which points to the string as it occurs within the BASIC program. More complicated strings, such as C$ = A$ + B$, however, use a working area outside of the BASIC program area. The PET creates this working area by starting at the upper limit of RAM and working down. For simplicity, the program above ignores this string working area and only simple string variables can be saved.

The programs above are written for and have been tested on the original ROM PET. They should run without difficulty on later PETs by changing the values "123" in line 1010 and 1040 to "41", "124" in line 1020 to "42", and "125" in line 1030 to "43".

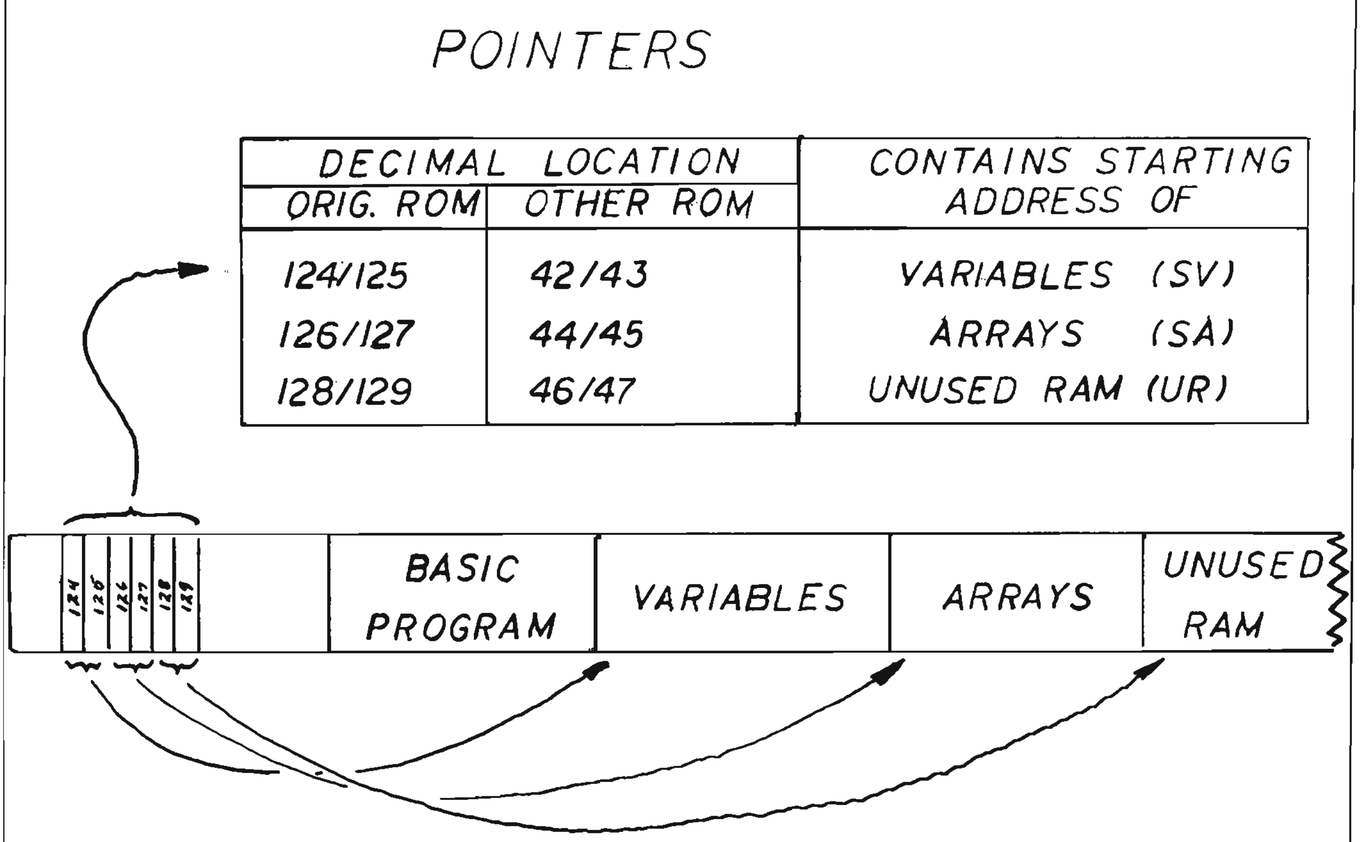


Figure a. PET uses pointers in system working area to show how RAM is organized.

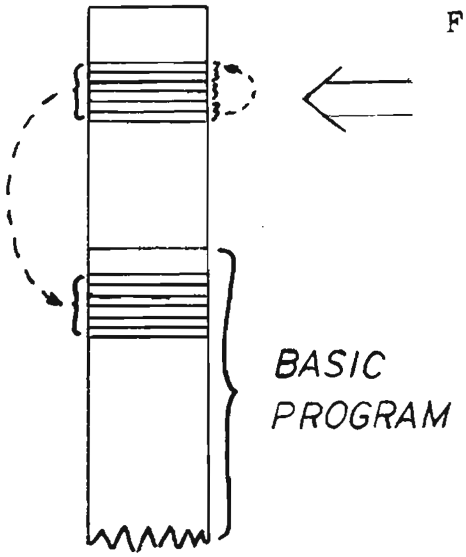


Figure b. Before a SAVE, pointers are moved from system working area to BASIC program. UR pointer is moved to SV.

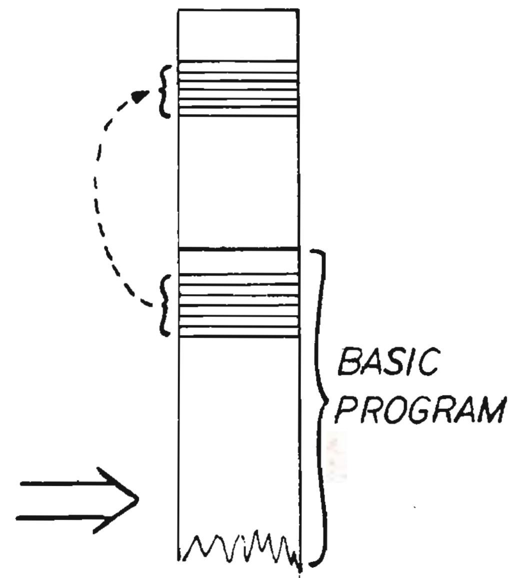


Figure c. After a LOAD, pointers are moved from BASIC program back to working area.

An Introduction to Assembly Language Programming on the VIC-20

by Eric Giguere
Canada

Part 1: Introduction

This is the first in a series of articles designed to help teach the fundamentals of 6502 assembly language programming on the VIC-20. It was created with beginners in mind, so those who already know a bit about assembly language may find the first parts boring, but later on it gets more interesting, I promise. NOTE: Although programs and methods presented in this series will be made for the unexpanded VIC-20 with the VICMON machine language cartridge, most should be able to work on any Commodore machine with only a few address changes. This month, though, I'm going to discuss the definition of assembly language.

The 6502

Inside your Commodore computer is a chip called the 6502 (the new Commodore 64 has a different version of the 6502, the 6510, but basically it is the same). Developed by MOS Technology, this marvel of miniature electronics is in fact **the** controlling element of your computer (if you don't count your control over the power switch). This chip has virtual total control over the rest of your computer, directing the reading, storing and processing of data. The amazing thing about this is that the 6502, like any other microprocessor, does all of this using only pulses of electricity. "How", you ask? It's a bit complicated, but I'll try to explain it.

Bits & Bytes

As I mentioned before, the 6502 does everything using only electricity. And what is the easiest way to represent things using electricity? Say your computer used lightbulbs inside instead of transistors. What would be the easiest way to use electricity in this set-up? Simple: you can have the power either on or off. So some lightbulbs in the computer might be ON at a given time, and some might be OFF. But that leaves us with a question: so what? I mean, what's the big deal? Either your lightbulbs are on or they're off. Of what use is it to us?

I'm glad your asked that, because you're absolutely right. Whether the bulbs are on or off is of no use by itself. But say we let the lightbulbs represent something, something like numbers. In that case, we could say that the bulbs that are ON could represent 1, and OFF means 0. Wouldn't this change the picture? Yes, it most certainly would, because now you have a way of counting. Counting? Yes, counting. Using only those two numbers, 1 and 0, we can represent any number in our "normal" numbering system, decimal. This numbering system is called BINARY, as it is based on only two numbers (base 2), just as decimal, or base 10, is based on ten numbers (0 to 9). If this sounds weird to you, then just have patience please, because as you read along it will become very clear.

Binary is a base two numbering system. This means that each position in a binary number represents a power of two, just as each position in a decimal number represents a power of ten (see figure 1). This means that the numeral at the starting point, the rightmost position, would represent 2 to the power of zero, which equals 1. Positions to the left of this also represent powers of two, with the exponent increasing by one each time you move over one position. In this way, the next position on the left of the starting point would represent 2 to the first power (2), the next 2 to the 2nd power (4), and so on. Following this line of thinking, the binary number 101 would represent the following in decimal: (starting from the leftmost digit)

$1 \times 2^2 = 4$
$0 \times 2^1 = 0$
$1 \times 2^0 = 1$
Total = 5

Thus you see how 101 binary represents 5 in decimal. This always works that way, and it helps to know this method to find a decimal number from a binary number so that you can do quick conversions. Quick! What is 111 binary? (If you said 7, then you're on the right track) . . . . .

Now that you (hopefully) understand binary, it's time to talk about bytes. A byte is a grouping of eight binary numbers, and can represent a decimal number from 0 to 255. Each binary number inside the byte are called **bits**, which stands for BInary digiTS. Makes sense, right? Remember that each of the eight bits in a byte represents a power of two, so the leftmost bit (usually called bit 7, with the rightmost bit called bit 0) will have a value of 2 to the seventh power, or 128. If all the bits are on, then the value of that byte would be $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$, the maximum value. Once this concept of bits and bytes is understood, it becomes quite easy to see exactly how your computer works.

Numbers, numbers, numbers . .

Remember how I said that the 6502 can only use and understand electricity? Well now that I've shown you how electricity can be used to represent numbers, this means that the microprocessor can also understand these numbers, because they are only electricity, electricity that is either on or off. And now that the microprocessor **understands** numbers, it can also do something with these numbers. It's this something that we are concerned about, and this brings us to what is called **machine language**, the chip's native tongue.

Machine language? But isn't this article about assembly language? Yes, it is, but to understand assembly language, you must first know what machine language is (why do I have the feeling that we've heard this before? . . .). Quite simply, machine language is nothing but numbers. It's all numbers; no letters or anything else. These numbers are fed to the microprocessor (in binary form, of course) and are interpreted by the chip as an instruction to do something, which it will then do. Alternatively, the number might be interpreted as data to be processed, and the chip will then use that data, processing it as instructed by another number. The thing to remember, though, is that machine language is just that: the only language the computer really understands. Every other computer language that we use is actually interpreted by a huge machine language program into a form understandable by the computer. Because the computer has to stop and interpret your BASIC or FORTRAN program, pure machine language code can run hundreds to thousands of times faster, and use much less memory space. But people usually can't relate to numbers that easily (remember algebra?), and this usually makes it difficult to program in machine language. Wouldn't it make sense then if we could code the machine language instructions into some easily understandable form and then have this code converted back to machine language after we'd done what we wished with it? That is exactly what assembly language is! Aren't we smart?

Assembly Language

When it comes to converting numbers to some easily understood code, what comes up in your mind first? Letters? Words? Good, because that's the right answer. Each coded instruction in machine language can be converted to a three-letter word, called a mnemonic, in assembly language. These mnemonics are much easier to understand than their respective numbers in machine language. Take for example the instruction "load the accumulator". In machine language this would be shown as 133 (decimal), whereas in assembly language it would be coded at LDA. Isn't LDA much easier to understand and remember than 133? That's the basic reason why assembly language is easier to use than pure machine language. In essence, though, you have to remember that they are basically the same, since the assembly language mnemonics are converted to numbers and stored as machine language in memory when you are finished with the program. Now that you understand what assembly language is, you're almost ready to do some programming. First, though, we'll have to go over the equipment you'll need, and we'll have to leave that for next month, because I'm simply running out of room here. In the meanwhile, if you have any questions on assembly language for the VIC, please send them to the address below and I'll be more than happy to answer them.

Eric Giguere, Box 901, Peace River, Alberta, Canada TOH 2X0

Figure 1: Powers of ten and powers of two

In decimal each number represents a power of ten multiplied by that number. For example, 309 would be represented as:

$3 \times 10^2 =$	3×100	=	300
$0 \times 10^1 =$	0×10	=	0
$9 \times 10^0 =$	9×1	=	9
			309

In the same way a binary number can be represented by using powers of two. Thus the binary number 110101 could be shown as:

$1 \times 2^5 =$	1×32	=	32
$1 \times 2^4 =$	1×16	=	16
$0 \times 2^3 =$	0×8	=	0
$1 \times 2^2 =$	1×4	=	4
$0 \times 2^1 =$	0×2	=	0
$1 \times 2^0 =$	1×1	=	1
			53

Thus 110101 binary is the same as 53 decimal.

Figure 2: Table of binary numbers 1-10.

Decimal	Binary
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010

Word Pro + TP-I = Word Processing

by Neil Omvedt
Roseville, Minnesota

In the process of setting up my new office I decided I needed a word processor to handle most of my paperwork. Since I was going to need a microcomputer for my work, I decided that a combination word processor-microcomputer was what I needed. Some amount of time was spent investigating various micro-computers including the IBM PC, the Apple III, the Xerox and several others. I did set several specifications for the machine. Most important of these was that it have an 80 column screen (both for word processing and for use as a terminal). Other major concerns were memory, disk storage space and availability of software.

Another concern of course was price. At the time a local computer store was having a closeout sale on their Commodore equipment. Since I have for some time used a Commodore Pet computer (original equipment with many upgrades) I was familiar with their equipment. While Commodore has received a bad name in some quarters I have always felt that this partially was due to their low prices (not as much profit margin for the dealers). Their equipment never gave me a significant amount of trouble. Due to the closeout sale I was able to purchase a CBM 8032 with a CBM 8050 disk drive and Word Pro IV Plus at about $1100 off Commodore's already low prices. Since this computer will store about 500K of data on one 5½" floppy diskette I felt this would leave me a lot of room for data files. Although the free RAM memory is only 32K since there is an interpreter in ROM this is almost as much free memory as is available on most other 64K machines.

The Word Pro IV Plus word processing system is one of a series developed by Professional Software Inc. for the Commodore Computers. The series started with Word Pro I and runs up to Word Pro V Plus. The Plus series are the programs for 80 column machines. Word Pro IV Plus is the top for the CBM 8032 and Word Pro V Plus is for the CBM 8096 (a CBM 8032 with 64K additional memory). The Word Pro IV is a full featured word processor. It basically works with two files in direct storage, the main text file and the alternate text file. The total lines allowed is 139. At the time you sign on you must allocate between 70 and 116 for main text with the rest being for alternate text. Alternate text can be used for various purposes, one of them being to load in a list of names and addresses for filling in form letters. One problem is that if you bring up the program for a large alternate text area and then wish to load an oversized file into the main text you have to go back and restart the program for the new job. The Word Pro IV Plus does allow chained files so long texts may be printed out. There is a full selection of both local and global editing features. Also, access is allowed to the Commodore disk files so that disk commands can be given through Word Pro IV Plus. The program also allows output to be continuous or to stop so that each page may be inserted in the printer. The global output command allows you to stop in the middle of a page, correct that page and resave the file, then restart at the beginning of that page. The program provides page headers and automatic page numbering. There are also some numeric features, but they are very limited. Continuous output may be done through use of a special disk file while you are using the program for editing another file.

Documentation for the system is very complete. The chapters are written in lesson format with exercises on how to use the system, but it can still be used as a reference guide. I found it useful to put tabs on the notebook to faster locate important items. There is a chapter near the end of programmers notes for use in more technical applications. There is also a section giving printer modification for more common word quality printers and interfaces.

In addition to the comments above about size of files there are two other shortcomings in the system. It uses one of the ROM spaces (UDII on the CBM 8032). This either rules out use of other programs requiring this location or requires purchase of a ROM switch. Also there is no help text on the screen to explain commands. This is partially overcome by use of special tabs on the front of the command keys.

The only other item was to find a word quality printer to use with my word processing system. I decided to try out the Smith Corona because of its low cost. Since I had a serial interface I ordered one with a serial RS 232 interface expecting to be able to use it with possibly a few simple changes such as the baud rate. When the day came for the printer to arrive I was in for a surprise. The first thing I noticed was the connector on the computer was a female connector which didn't mate very well with my female connector interface. However I figured this would be simple to correct and forged on to the next problem.

The next thing was to read the manual. It was very uninformative. It said data format would allow 1, 1½ or 2 stop bits selected by jumpers on the interface PC board. Word length may be 7 or 8 bits and is determined by jumpers on the interface PC Board. There are 16 baud rates selected by jumpers on the interface PC Board. The only problem with all this information is that it didn't tell how to access the interface PC board or how to set any of these items. There was no clear access to the interface.

My first thought was to ship the printer back and keep looking or to find a printer locally. After deciding that it would practically require taking the printer apart to do anything this was my second thought also. However before returning the printer I decided to call the dealer. They indicated that there was a data sheet for RS-232 interfacing that should have been sent with the printer. I decided to wait for the data sheet before returning the printer. After four days of waiting I called again and they said they had sent the sheet out - blame the post office. So I asked for information over the phone and this is what I found out:

1. To access the switches use the following procedure (which of course should only be done by a trained service technician):
   a. Remove the two screws inside the front cover.
   b. With a 1/16 inch allen wrench loosen the two arm screws on the right platen knob.
   c. The case is snap locked in back — loosen it.
   Note: You should be careful not to pull loose any of the wires when you do this.
2. Adjust the switches to the proper settings for you interface:
   a. Switch one is on for 7 data bits and off for 8 data bits.
   b. Switches two and three are set as follows:

Switch	No parity	Odd parity	Even parity
2	On	Off	Off
3	Off	On	Off

   c. Switches four through seven control the baud rate. For three hundred baud the setting was on, off, off, on.

The next thing I needed to do was find an RS-232 cable with two male ends on it. After checking a number of stores (about 10) I found it available at three places — one for $50, one offered to make one up for $35 and the third one had one available for $29. Naturally I bought the $29 cable (which was very high quality). It seemed like a lot of money to pay for the short cable (maybe I should invest in a company that makes cables), but what choice does one have.

The next item was to set the switches. I followed the instructions to remove the cover and located the small switches on the circuit board. I set the switches as above (or so I thought), put the cover back on and plugged the printer into the RS-232 interface and the power. Then I ran a test program using my Word Pro IV Plus word processor. My interesting result was a line of "@"'s. Since it was late on a Friday night I decided to wait for the sheet to recheck all the settings.

When the information sheet came it indicated that all my settings were correct, but a more careful reading indicated the problem. The slide switches used for the setting have two little red dots near the ends. Only one shows at a time. I had assumed if the red dot near the "on" end was showing the switch was on. However, apparently to have the switch on the red dot must be showing near the "off" end of the switch. So I had to go back and reverse the settings on all the switches. When this was done the printer worked fine.

My advice to you if purchasing this printer to use with an RS-232 interface is to determine what settings you need ahead of time and have the dealer do this. Although the actual procedure is not much more complicated than I assumed, due to the problems finding the cable and the problems I had due to not having the interface sheet on hand the actual amount of time used was more than I expected. Also taking the cover off yourself would invalidate any warranty. Actually I haven't yet found any sheet that indicates there is any warranty, but you still might invalidate the implied serviceability warranty of a new product.

The Word Pro IV Plus has several features that use special characters codes for printing:

1. Automatic underlining: This feature worked fine on the Smith-Corona TP-I.
2. Subscripts and superscripts: The printer ignored the special characters for subscripting and superscripting and just printed the characters.
3. Bold face: The printer did not support the bold face printing. Again it ignored the special characters related to this feature.

The printer book indicated two other special control codes that are accepted. One is the setting and releasing of program margins. The margin setting of Word Pro worked fine. Whether this was done before the text is sent out or by the printer I do not know. The other function is tab settings. As far as I know Word Pro IV Plus has no feature for tab settings on the printer (it does have this feature for input into program text files) although this could be used if necessary by defining one of the special characters allowed by Word Pro IV Plus for this feature.

In trying to print this article the first time I discovered some peculiarities of the printer top of form handling. In order to print a multi page report use the following procedure:

1. Set your printer page size to 66 by using the pp command.
2. Set the TP-I top of form switch to the clear position.
3. Print your report without using the Word Pro continuous feature. The Word Pro program will stop after each page to allow you to insert paper.

My overall evaluation of the printer is that it is an excellent printer for the price you pay for it (list price of $895 with mail order availability as low as $650). There are not too many parts so maintenance should be fairly simple if any problems do occur. The shortcomings of the system are as follows:

1. The fan on the back of the printer is always on and is slightly noisy.
2. Only one pitch setting is allowed with the printer—it may be either 10 pitch or 12 pitch.
3. The printer does not allow superscripts or subscripts and it does not allow bold face printing.
4. The printer is somewhat slow—only 12 characters per second compared to 40 or more for more expensive printers.
5. The printer does not have a form feed for continuous paper.

I did not consider any or all of these to be of enough importance to pay a considerable amount more for these features.

6502 MPU Hybrid

by Gary G. Cordelli
Linglestown, Pennsylvania

I have been the owner of a PET 2001 microcomputer for about 5 years now, but have recently purchased a commodore VIC-20 because of its portability and compatability with my Commodore PET. I like the fact that I can put this little computer under my arm and take it wherever I go, and I like the fact that I can transfer tapes—and in fact the entire tape drive—from one computer to the other.

Intending to do some graphics and animation work on the VIC-20, I decided that I'd be programming in machne language a lot and so I wanted a debugger. I remembered that I had written one for my old PET, and while digging up the documentation on this program, I cam across some old notes that I found interesting.

About 4 years ago I wrote a machine language debugging aid for the Commodore PET. This debugger did not actually trace through the user's program the way some do, but instead simulated the execution of the program instructions for the most part. This was done to eliminate the possibility of "hanging up" the PET since it had no "warm" reset. The program included a table showing the number of bytes for each instruction from ØØ through FF. A ØØ entry in this table signified an "illegal" opcode, and execution of the program was halted, noting that an illegal opcode was encountered.

By experimenting with values between Ø1 and Ø3 in place of the ØØ entries for those opcodes officially listed as "illegal" I was able to determine the behavior of some of these instructions. I first started with the lowest valued "illegal" opcode (Ø2) and worked upward. This proved very frustrating, and then I noticed that certain patterns were present among these "illegal" opcodes. Most notably, ALL of the opcodes ending in 3,7,B, and F are not "legal".

After experimenting with these opcode types, I discovered another pattern. The instructions appeared not to be performing totally alien operations, but instead seemed to be performing two already existing operations!

As an example of this pattern, the "illegal" Ø7 opcode performs an ASL on a Zero Page location and an ORA on the same location. These two operations are performed separately by the Ø6 and Ø5 opcodes. The bit pattern of the opcode itself seems to shed some light on these special "double instructions":

Opcode	Operation
Ø6= 00000110	ASL zp
Ø5= 00000101	ORA zp
Ø7= 00000111	ASL zp and ORA zp

As you can see, Ø7 includes the bits for **both** Ø5 and Ø6, and these are the instructions actually executed. The actual order of processing (to the external user) is Ø6 then Ø5.

This works the same for the Ø7,17,27,37,47,57,67,77,A7,C7,D7, E7, and F7 opcodes. The B7 opcode is an exception in that the B5 opcode is not strictly executed. B5 should be a LDA zp,X but B7 performs a LDA zp,Y instead. Thus the Accumulator and the X register are both loaded from the same location (ie, from zp,Y). Two other exceptions are the 87 and 97 instructions. The reason for these exceptions is that they execute two instructions that call for a modification of the same memory location. The 85 instruction stores the Accum. at a Zero Page location, and the 86 instruction stores the X register at the **same** location. How is this possible (you may ask)?

Inside the 6502 MPU both the Accumulator (AC) and the X register (XR) attempt to command the internal data bus running between the registers and the ALU. If you think of the internal register outputs as being like "tri-state" open-collector type buffers, you would find that a Ø in a bit position in either register would result in a Ø on the bus, where as a "1" in a bit position in both registers would result in a "1" on the bus in that position. The result would be the equivalent of a positive-logic AND performed on the two registers. In fact, this is exactly the result of the 87 instruction. In attempting to execute both the STA zp and STX zp instructions, the 87 opcode causes the result of the expression (AC).AND.(XR) to be placed in the Zero Page location specified in the operand. The 97 opcode performs the same operation, placing the result in the location ZP,Y instead.

In addition to the "illegal" instructions ending in a "7", this "double instruction" behavior is evidenced in those opcodes ending in an "F". As before, we can see:

	Opcode	Operation
ØE	00001110	ASL Absolute
ØD	00001101	ORA Absolute
ØF	00001111	ASL Abs. & ORA Abs.

Everything about these instructions is the same as the "illegal-7" instructions (operating on Absolute memory instead of a Zero Page location) for all except the 9F opcode which does not function the same as its 97 counterpart. We might expect the 9F hybrid to perform (AC).AND.(XR) and store the result in a location specified by Absolute,Y, but the problem appears to be that there is no 9E instruction to execute. As far as can be determined, the 9F opcode affects only the location specified by Abs,Y but the expression result stored there is (AC).AND.(XR).AND.$01 (ie, the result is $00 if either is even and $01 is both are odd).

Another set of "double instructions" is the "illegal-3" type. From the examination of the "illegal-7" and "-F" instructions, you might expect these instructions to perform both the _1 and _2 instructions. A check of the list of documented legal instructions, however, shows that the opcodes ending with a "2" (with the exception of A2) are themselves "illegal". Moreover, the "illegal-2" instructions (with few exceptions) do NOT perform any undocumented operations, but instead perform the infamous JTH (Jump To Hyper-space), thus "hanging" the processor.

But let's look a little closer at the bit patterns of opcodes again. The examination of the "illegal-7" and "-F" opcodes showed that the 8 bit (ie, bit 3) controlled the operand address; that is, when bit 3 was "Ø" the address was Zero Page (ZP), and when bit 3 was "1" the address was Absolute (Abs). But there are more addressing modes than can be controlled by a single bit value. To uncover a more complete explanation of addressing mode control, let's look at some of the opcodes beginning with "Ø" and "1" (hex):

Hex	Binary	Operation
Ø1	00000001	ORA (Indirect, X)
Ø5	00000101	ORA ZP
Ø9	00001001	ORA Immediate
ØD	00001101	ORA Abs
11	00010001	ORA (Ind.), Y
15	00010101	ORA ZP, X
19	00011001	ORA Abs, Y
1D	00011101	ORA Abs, X

From this we see that the code 000xxx01 seems to define the ORA instruction itself, while the codes xxx000xx through xxx111xx (ie, bits 4,3 &2) control the addressing mode. (This does not hold true for cases when bit 7 is set, such as 96— or 10010110 binary—which is STX zp,Y not STX zp,X).

Now let's look at how this parallels the ASL instruction:

Hex	Binary	Operation
02	00000010	"illegal"
06	00000110	ASL ZP
0A	00001010	ASL A
0E	00001110	ASL Abs
12	00010010	"illegal"
16	00010110	ASL zp, X
1A	00011010	"illegal"
1E	00011110	ASL Abs, X

From this we see that the addressing modes match those of the ORA instructions with the same b4, b3 & b2 values (with Accum. mode & sometimes Implied mode replacing Immediate mode whenever both b3 **and** b1 are "1") except that the opcodes ending with a "2" are "illegal". (More on the "illegal" 1A later).

Referring back to the ORA instructions we see that Ø2 should be ASL (Ind,X) and 12 should be ASL (Ind,Y), although they do not execute these operations. This is where some light is shed upon the "illegal-3" instructions. The Ø3 instruction performs the missing ASL (Ind,X) then ORA (Ind,X). Furthermore, this holds true for nearly all the "illegal-3" opcodes (ie, they perform the "phantom-2" instruction then the _1). The exceptions are the A3 (where A2 is legal) and 93 (which has the same peculiarity as the 97 and 9F opcodes). The A3 opcode performs the A2 instruction as if it were a "phantom-2" type (ie, it does a LDX (Ind,X) not LDX Immediate) and the

A1 instruction (LDA (Ind,X), however the order appears reversed since both operations use the "old" value of the X register as the indirect index. The 93 instruction performs the same odd/even check of the 9F opcode except that the addressing mode is (Ind),Y.

The last fairly consistent set of "double instructions" is the "illegal-B" type. Just as with the _7 and _F opcodes, this type differs from another illegal set ("illegal-3") by just a single bit (b3). We would thus expect the operations to be similar to the "illegal-3" type, differing only in addressing mode. Thus, the _B should perform the _A then the _9 operations. However, looking at the documented opcodes, we see that the _A opcodes with even first digits (ØA,2A...) all use Accumulator or Implied addressing, where as the _9 opcodes with even first digits use Immediate addressing. This represents a conflict because the former are single-byte instructions (no operand byte), while the latter are double-byte instructions (one operand byte). It appears this conflict may cause some problem for the 6502 MPU as the "even-B" opcodes have inconsistent results (more later). The "odd-B" opcodes do perform the expected _A and _9 instructions, however. As we examined the "illegal-3" opcodes in depth, we noted that the 1A instruction (and, in fact, all the "odd-A" opcodes except 9A and BA) is illegal itself. But just as with the "illegal-3" opcodes, the "illegal odd-B" opcodes perform the "phantom odd-A" then the _9. Since the odd-A codes have bit patterns of xxx11010 we expect the addressing mode to be Abs,Y. This is just the case.

The 9B instruction does not follow the other "illegal odd-B" pattern, but again displays the odd/even check operation of the 9F instruction with an addressing mode of Abs,Y. The BB instruction is the oddest of all opcodes, as its execution places the same value into the AC, XR, and Stack Pointer (SP). It appears that it may perform a LDA, a TAX, and a TXS, but the source of the byte placed in these three registers is questionable. The B9 opcode uses Abs,Y addressing, but the AA & BA instructions (TAX & TXS) use Implied addressing. All that can be known for sure is that the processor does, in fact, fetch two bytes as the operand address. From experiments it appears that the byte stored in the three registers is the result of a logical AND between the "old" SP value and the byte at Abs,Y. This could be expressed as SP ←XR ←AC ← SP.AND.(Abs,Y). If anybody finds a practical use for the BB opcode, please tell the rest of us.

If we look now at the even-B opcodes, we find that the previous patterns do not hold. The ØB and 2B opcodes perform a simple AND Immediate. The 4B and 6B opcodes do perform the 4A and 6A operations, but not the 49 and 69. Instead, they perform an AND Immediate, and then the 4A and 6A instructions, respectively. The 8B opcode performs a logical AND between the XR, an Immediate operand, and the value $Ø2 and places the result in the AC. The AB opcode performs the AA and A9 operations, but in reverse order (ie, LDA Immediate then TAX), thus both the AC and XR are loaded with the same byte. The CB opcode is very nice as it performs a logical AND between the AC and XR and then subtracts the value of an Immediate operand and stores the result in the XR. If the instruction is CB ØØ the result will be simply (XR).AND.(AC). It should be noted that the subtract performed by the CB opcode is a Subtract Without Carry (SUB) and the Carry flag is unmodified by the operation though the Negative and Zero flags are. If the XR contains an $FF, the result of the CB operation would then be (AC)-(Immediate operand) and would be placed in the XR, providing the only method of performing a SUB (versus SBC) on the 6502 without affecting the Carry flag (interesting, but not very practical). Finally, the EB opcode actually follows the normal pattern, performing the EA and then E9 instructions (NOP then SBC Immediate), though there is, in fact, no way of determining the order since EA (NOP) does nothing.

It would seem obvious that the most powerful "double" or "hybrid" instructions would be the 83,87,97, and 8F opcodes. We have all at one time or another probably found ourselves with two values—both the result of internal register manipulations—that we wished to AND together without losing either value. Previously, one would have to store the values in the XR and AC memory, AND the location of the XR value with the AC, and then if we wished to reuse the AC or place the old value back in it, we had to store the result of the AND somewhere in memory. With these hybrid instructions, one can now AND two values from the AC and XR and Store the result in memory with a single instruction, and both the original values in the registers would be unaffected.

The CB instruction is also a powerful one, performing a two register AND and placing the result in the XR instead of memory (with the option of altering the result by a SUB on the XR by an Immediate operand if one chooses as this operand a non-zero value).

The AB opcode is powerful as a time saver when initializing registers because it loads the AC and XR with the same Immediate value. Both registers can thus be cleared by a single instruction (ie, AB ØØ).

The use of most of the other hybrid instructions are somewhat limited to specific situations, but there are possible applications. Figures 1 through 4 list the operation of the four hybrid instruction types, as well as the type of application suggested by the operations. It should be noted that the Processor Status Register (PSR) reflects the status of the **entire** "double" or hybrid instruction, and thus the status of the first operation may be obscured by the modifications of the PSR by the second operation if both affect the same flags. It is interesting to note, however, that the "illegal-3" and "illegal-B" instructions represent the ONLY way certain operations can be performed with addressing modes of (Ind,X); (Ind),Y ; and Abs,Y (eg, ASL (Ind,X); DEC (Ind),Y ; and INC Abs,Y). One would have to be careful to make sure that the second half of the hybrid instruction (affecting only the AC and PSR and **not** the value at the operand address) had no unexpected or unwanted effects.

I should mention one more thing about timing with these hybrid instructions. Throughout this discussion I have described these instructions as performing **first** one operation, and then a **second** operation. This description was used to explain how the operation of the hybrid instructions appear to the user—**outside** the processor. In actuality, the 6502 MPU does not execute one instruction and then the second. Both instructions included in the hybrid "double instructions" are actually executed **at the same time**. The total time required by these type of hybrids is equal to the time required to execute the **longest** of the included operations. For example, the ∅5 opcode takes 3 cycles to execute (3uS), and the ∅6 opcode takes 5 cycles to execute. The ∅7 hybrid, then, takes 5 cycles to execute both the ∅6 and ∅5 opcodes. The ∅7 hybrid is thus 38% faster than the equivalent ∅6 and ∅5 pair. (It also uses 50% less space). On one of the more practical hybrids, the savings is more pronounced. In the situation described in the paragraph discussing the power of the 83,87,97, and 8F opcodes, the shortest routine for performing the logical AND on the AC and XR while preserving both registers and the result of the AND follows:

```
PHA                ; save AC
STX    Z PLOC      ; set up for AND
AND    Z PLOC      ;AND AC&XR values
STA    Z PLOC      ; save result
PLA                ; restore AC
```

The total time to execute this section of code is 16 cycles (16uS on a 1MHz 6502). The equivalent hybrid instruction for ALL of this code is 87 xx, which I write as: STAX ZPLOC. The total time for this code is the same as for either the STA zp or STX zp (in this case they are both equally long) which is 3 cycles. This represents a savings of 81% in time! The savings in space is 75% in this same example! It's very nice to be able to perform this operation in 1/4 the space and 1/5 the time, don't you think?

Alas, there had to be a catch somewhere. Because these are undocumented instructions, there may come a time when MOS Technology (a division of Commodore International Business Machines) decides to add some

```
        Performs                        Use

07xx   ASL zp        then   ORA zp    bit sets
17xx   ASL zp,X       "     ORA zp,X     "
27xx   ROL zp         "     AND zp    bit check
37xx   ROL zp,X       "     AND zp,X or clears
47xx   LSR zp         "     EOR zp    bit flips
57xx   LSR zp,X       "     EOR zp,X     "
67xx   ROR zp         "     ADC zp
77xx   ROR zp,X       "     ADC zp,X
87xx   (AC).AND.(XR) stored at  zpsee text
97xx   (AC).AND.(XR) stored at  zp,Y  "
A7XX   LDX zp        then   LDA zp
B7xx   LDX zp,Y       "     LDA zp,Y
C7xx   DEC zp         "     CMP zp   looping
D7xx   DEC zp,X       "     CMP zp,X    "
E7xx   INC zp         "     SBC zp Actuallydoes
F7xx   INC zp,x       "     SBC zp,X(AC-zp-C-1)
```

Figure 1: "illegal-7" Operations

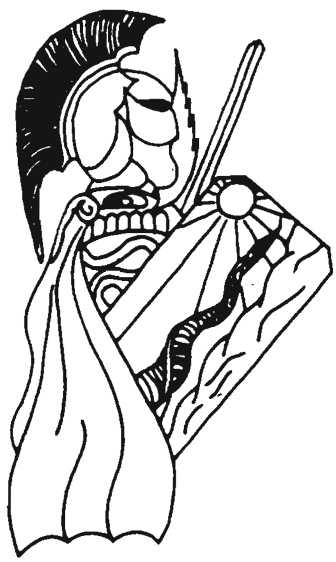
```
        Performs                        Use

0FxxxxASL Abs    then   ORA Abs   bit sets
1FxxxxASL Abs,X   "     ORA Abs,X    "
2FxxxxROL Abs     "     AND Abs   bit checks
3FxxxxROL Abs,X   "     AND Abs,X or clears
4FxxxxLSR Abs     "     EOR Abs   bit flips
5FxxxxLSR Abs,X   "     EOR Abs,X    "
6FxxxxROR Abs     "     ADC Abs
7FxxxxROR Abs,X   "     ADC Abs,X
8Fxxxx(AC).AND.(XR)stored in Absoluteseetext
9Fxxxx(AC).AND.(XR).AND.$01 in Abs,Y    "
AFxxxxLDX Abs    then   LDA Abs
BFxxxxLDX Abs,X   "     LDA Abs,Y
CFxxxxDEC Abs     "     CMP Abs  looping
DFxxxxDEC Abs,X   "     CMP Abs,X   "
EFxxxxINC Abc     "     SBC AbsActually does
FFxxxxINC Abs,X   "     SBC Abs,X(AC-abs-C-1)
```

Figure 2: "illegal-F" Operations

instructions to their 6502 MPU. If they use some of these opcodes in the design of new instructions, there is a possibility that programs with hybrid code in them will not run correctly. Also, there is no guarantee that second-source manufacturers of 6502's such as Rockwell International will produce MPU's that behave the same when executing "illegal" opcodes as the Commodore MPU's do. As long as you are running on the same machine, there is no problem.

As a final note, I should mention the greatest benefit I ever realized from using hybrid instructions. Not long after I first discovered the "illegal-7" set of hybrids, I was required to write a program in a class on a 6502 based system (a KIM). One day in the library I lost my program and was disgusted at the prospect of having to rewrite the entire thing. The next day I heard from another student that someone was asking around about what an 87 instruction did. Apparently someone else in the class had found it and decided to save time by using my program rather than write one of his own. The 87 question was a dead giveaway that he had my program, and it didn't take much talking to convince the guy that he would be in big trouble if he didn't return the program to me. In that case, using a hybrid saved me hours of rewriting a program!



```
           Performs

03xx  ASL (Ind,X)  then  ORA (Ind,X)
13xx  ASL (Ind),Y   "    ORA (Ind),Y
23xx  ROL (Ind,X)   "    AND (IND,X)
33xx  ROL (Ind),Y   "    AND (Ind),Y
43xx  LSR (Ind,X)   "    EOR (Ind,X)
53xx  LSR (Ind),Y   "    EOR (Ind),Y
63xx  ROR (Ind,X)   "    ADC (Ind,X)
73xx  ROR (Ind),Y   "    ADC (Ind),Y
83xx  AND (XR)      "    STA (Ind,X)
93xx  (AC).AND.(XR).AND.$01 in (Ind),Y
A3xx  LDA (Ind,X)  then  TAX
B3xx  LDX (Ind),Y   "    LDA (Ind),Y
C3xx  DEC (Ind,X)   "    CMP (Ind,X)
D3xx  DEC (Ind),Y   "    CMP (Ind),Y
E3xx  INC (Ind,X)   "    SBC (Ind,X)
F3xx  INC (Ind),Y   "    SBC (Ind),Y
```

Figure 3: "illegal-3" Operations

```
              Performs

Obxx    AND Immediate
1Bxxxx  ASL Abs,Y     then  ORA Abs,Y
2Bxx    AND Immediate
3Bxxxx  ROL Abs,Y      "    AND Abs,Y
4Bxx    AND Immediate  "    LSR A
5Bxxxx  LSR Abs,Y      "    EOR Abs,Y
6Bxx    AND Immediate  "    ROR A
7Bxxxx  ROR Abs,Y      "    ADC Abs,Y
8Bxx    LDA ((XR).AND.$02.AND.Immediate)
9Bxxxx  (XR).ND.(AC).ND.$01 in Abs,Y
ABxx    LDA Immediate then  TAX
BBxxxx  LDA ((SP).AND.(Abs,Y))then TAX thenTXS
CBxx    LDX (((XR).AND.(AC))-Immediate)
DBXXXX  DEC Abs,Y     then  CMP Abs,Y
EBxx    NOP            "    SBC Immediate
FBXXXX  INC Abs,Y      "    SBC Abs,Y
```

Figure 4: "illegal-B" Operations

Commander Dealers

Given here, in zip code order, is a partial list of the Charter Dealers who will be carrying the COMMANDER. We will provide updates for this list in following issues as a service to provide our readers with a local source at which they will find information, hardware, or software for their Commodore Computers.

U.S.A.

New Hampshire

Compucraft, Inc.
17 Dunbar St.
Keene, NH 03431
(603) 357-3901
Manager-Owner: Richard Bishop

Maine

Maine Micro Systems, Inc.
555 Center St.
Auburn, ME 04210
(207) 786-0696
Manager: Nancy Lecompte

New Jersey

Computer Workshop
1200 Haddenfield Rd.
Cherry Hill, NJ 07013
(609) 665-4404
Manager-Owner: Charles Kolbe

Software City
147 N. Kinder Ramark Rd.
Montvale, NJ 07645
(201) 391-0931
Manager-Owner: C.M. Hatfield

BB/The Computer Store
216 Scotch Rd.
Trenton, NJ 08628
(609) 883-2050
Manager-Owner: Barry Brown

New York

B.C. Communications, Inc.
World Wide Electronics Dist.
207 Depot Rd.
Huntington Sta., NY 11746
(516) 549-8833

Personal Computers, Inc.
3251 Bailey Ave.
Buffalo, NY 14215
(716) 832-8800
Manager-Owner: Frank C. Smeirciak

Pennsylvania

One Stop Computer Shope
65 N. 5th St.
Lemoyne, PA 17043
(717) 761-6754
Manager-Owner: Joanne Wright

Micro Age Computer Store
1352 Tilghman St.
Allentown, PA 18102
(215) 434-4301
Manager-Owner: Ed Eichenwald

Maryland

Professional Micro Service
100 W. 22nd St.
Balto, MD 21218
(301) 366-0010
Manager-Owner: James A. Breen

Virginia

Virginia Micro Systems
13646 Jeff Davis Highway
Woodbridge, VA 22191
(703) 491-6502
Manager-Owner: Shelli

West Virginia

Computer Associates, Inc.
113 Hale St.
Charleston, WV 25301
(304) 344-8801
Manager-Owner: Jeff Knapp

North Carolina

Bob West Computers
54 West Main St.
Brevard, NC 28712
(704) 883-2595
Manager-Owner: Sylvia West

Florida

Random Access Computers
296 Nelgin Parkway
Ft. Walton Beach, FL 32548
(904) 862-7763
Manager-Owner: Joanne Dodd

Florida Book Store
1614 West University Ave.
Gainesville, FL 32604
(904) 376-6066

Osceola Computer
1300 Dakota Ave.
St. Cloud, FL 32769
(305) 892-1501
Manager-Owner: Raymond Barrieau

Computer Specialties, Inc.
701 E. Lincoln Ave., P.O. Box 1718
Melborirne, FL 32901
(305) 725-6574
Manager-Owner: Otis P. Lutz

Focus Scientific
224 N. Federal Highway
Fort Lauderdale, FL 33301
(305) 462-1010
Manager-Owner: M. Rienhardt

Alabama

Tricelin Corporation
Route 1, Box 128
Bankston, AL 35542
(205) 689-4999

Tennessee

Metro Computer Ctr.
P.O. Box 1406
Chattanooga, TN 37402
(615) 875-6676
Manager-Owner: Wayne F. Wilson

Mississippi

Sunrise Persons Supplies
P.O. Box 38341
Corinth, MS 38834
(601) 287-4721
Manager-Owner: Felex Gathings

Ohio

Office Mart, Inc.
11151 East Main St.
Lancaster, OH 43130
(614) 687-1707

The Computer Store of Toledo, Inc.
18 Hillwyck Dr.
Toledo, OH 43615
(419) 535-1541
Manager-Owner: Al and Jackie Miller

Indiana

Allen's Jewelry & Loan Co.
130 E. 10th St.
Anderson, IN 46016
(317) 642-7978
Manager: Jerry Rubenstein

AVC Corporation
2702 Applegate
Indianapolis, IN 46203
Manager-Owner: Brent Enderle

Custom Software
3197 South 3rd Place
Terre Haute, IN 47802
(812) 234-3242
Manager-Owner: Vicki McEntaffer

Michigan

Computer Mart
915 S. Dort Hwy.
Flint, MI 48503
(313) 234-0161
Manager-Owner: Pat McCollem

Computers and More
2915 Dretom
Grand Rapids, MI 49508

Wisconsin

Majic Business Systems
3519 W. Wanda Ave.
Milw, WI 53221
(414) 282-8072
Manager-Owner: Dennis Woitekaitis

Computerland of Madison
6625 Odana Rd.
Madison, WI 53719
(608) 833-8900
Manager-Owner: James Sullivan

South Dakota

Computerland Rapid City
738 St. John St.
Rapid City, SD 57701
(605) 348-5384
Manager-Owner: John Mattson

Illinois

The Software Store, Inc.
1767 Glenview Rd.
Glenview, IL 60025
(312) 724-7730

Digital World
711 Army Trail Rd.
Addison, IL 60101
(312) 628-9222
Manager-Owner: Sam Gunda

B-A Computer Sys.
2 N. Batavia Ave.
Batavia, IL 60510
(312) 879-2350
Manager-Owner: Robert Appel

Rozel Industries, Inc.
7360 N. Lincoln Ave.
Lincolnwood, IL 60646
(312) 675-8960
Manager-Owner: Fred Whitlock

Kappel's Computer Store
125 E. Main
Belleville, IL 62220
(618) 277-2354
Manager-Owner: Tom Kappel

Data Plus, Inc.
1706 Broadway
Quincy, IL 62301
(217) 222-6502
Manager-Owner: James Moore

Missouri

Common Wealth Computers
5214 Blue Ridge Blvd.
Kansas City, MO 64133
(816) 356-6502
Manager-Owner: Dick York

Kansas

Computer Business Machines
Officenter 357 S. Lulu
Wichita, KS 67211
(316) 267-1150
Manager-Owner: Mrs. R. Santoscoy

Texas

Computer Home
431 East Ave. C
San Angelo, TX 76903
(915) 653-7488
Manager-Owner: Brent DeMoville

Computerland of Amarillo
2300 Bell St.
Amarillo, TX 79106
(806) 353-7482
Manager-Owner: Mark Trowbridge

Colorado

Zero Page, Inc.
2380 Naegele Rd.
Colorado Springs, CO 80904
(303) 633-0211
Manager-Owner: David C. Cooper

Utah

Mnemonics Memory Systems
(DBA Mnemonics Computer Store)
141 E. 200 South
Salt Lake City, UT 84111
(801) 266-7883
Manager: Rick Giolas

Arizona

Personal Computer Place
1840 W. Southern Ave.
Mesa, AZ 85202
(602) 833-8949
Manager-Owner: Roger Smith

Nevada

PCS Computer
3900 W. Charleston, Ste R
Las Vegas, NV 89102
(702) 870-4138
Manager-Owner: Mickey Cole

California

Data Equipment Supply Corp.
8315 Firestone Blvd.
Downey, CA 90241
(213) 923-9361
Manager: Robert Johnson

Computer Place
23914 Crenshaw Blvd.
Torrance, CA 90505
(213) 325-4754
Manager-Owner: Wen T. Huang

Fyrst Byte
10053 Whittwood Dr.
Whittier, CA 90603
(213) 947-9411
Manager-Owner: Darrell Miller

Data Systems West
421 West Las Tunas Dr.
San Gabriel, CA 91776
(213) 289-3791
Owner: Frank J. Mogavero

Consumer Computers
8314 Parkway Dr.
La Mesa, CA 92041
(714) 465-8888
Manager: Steve Scott

Calco Digital Equipment Inc.
1919 Aple St.
Oceanside, CA 92054
(714) 433-4119
Vice President: Ronald N. Paperno

Micropacific Computer Center
5148 N. Palm
Fresno, CA 93704
(209) 229-0101
Manager-Owner: Mike Reinhold

The Radio Place
2964 Freeport Bl.
Sacramento, CA 95818
(916) 441-7388
Manager-Owner: Gary Stilwell

Ray Morgan Co.
554 Rio Lindo Ave.
Chico, CA 95926
(916) 343-6065
Manager: Dave Wegner

Oregon

SW Computers
1125 N.E. 82nd
Portland, OR 97220
Manager-Owner: Jerry

Advertising Index

Edu-Tech
1575 N.W. 9th
Corvallis, OR 97330
(503) 758-5577
Manager-Owner: L. Clark/W. Brown

Washington
Computer Corner
1610 N. LaVenture
Mt. Vernon, WA 98273
(206) 428-1840
Owner: Kirk D. Shroyer

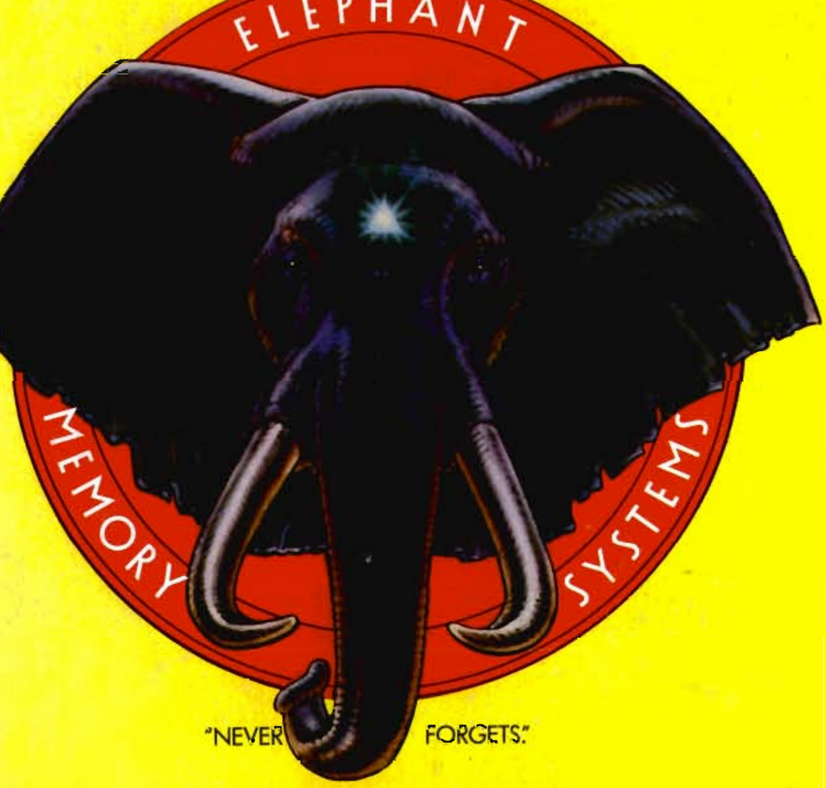
Alaska
BG Systems Co.
204 East International
Anchorage, AK 99502
(907) 276-2986
Manager-Owner: Robert DeLoach

Micro Age Computer Store
2440 Seward Highway
Anchorage, AK 99503
(907) 279-6688
Manager-Owner: Jay Wisthoff

CANADA

Ontario
House of Computers
368 Eglinton Ave. W.
Toronto, ON M5N 1A2
(416) 482-4336
Manager-Owner: Mark Herzod

REMEMBER:



MORE THAN JUST ANOTHER PRETTY FACE.

Says who? Says ANSI.

Specifically, subcommittee X3B8 of the American National Standards Institute (ANSI) says so. The fact is all Elephant™ floppies meet or exceed the specs required to meet or exceed all their standards.

But just who is "subcommittee X3B8" to issue such pronouncements?

They're a group of people representing a large, well-balanced cross section of disciplines—from academia, government agencies, and the computer industry. People from places like IBM, Hewlett-Packard, 3M, Lawrence Livermore Labs, The U.S. Department of Defense, Honeywell and The Association of Computer Programmers and Analysts. In short, it's a bunch of high-caliber nitpickers whose mission, it seems, in order to make better disks for consumers, is also to make life miserable for everyone in the disk-making business.

How? By gathering together periodically (often, one suspects, under the full moon) to concoct more and more rules to increase the quality of flexible disks. Their most recent rule book runs over 20 single-spaced pages—listing, and insisting upon—hundreds upon hundreds of standards a disk must meet in order to be blessed by ANSI. (And thereby be taken seriously by people who take disks seriously.)

In fact, if you'd like a copy of this formidable document, for free, just let us know and we'll send you one. Because once you know what it takes to make an Elephant for ANSI . . .

We think you'll want us to make some Elephants for you.

ELEPHANT.™ HEAVY DUTY DISKS.

For a free poster-size portrait of our powerful pachyderm, please write us.

Distributed Exclusively by Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021

Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.