CDC 00594

The Transactor

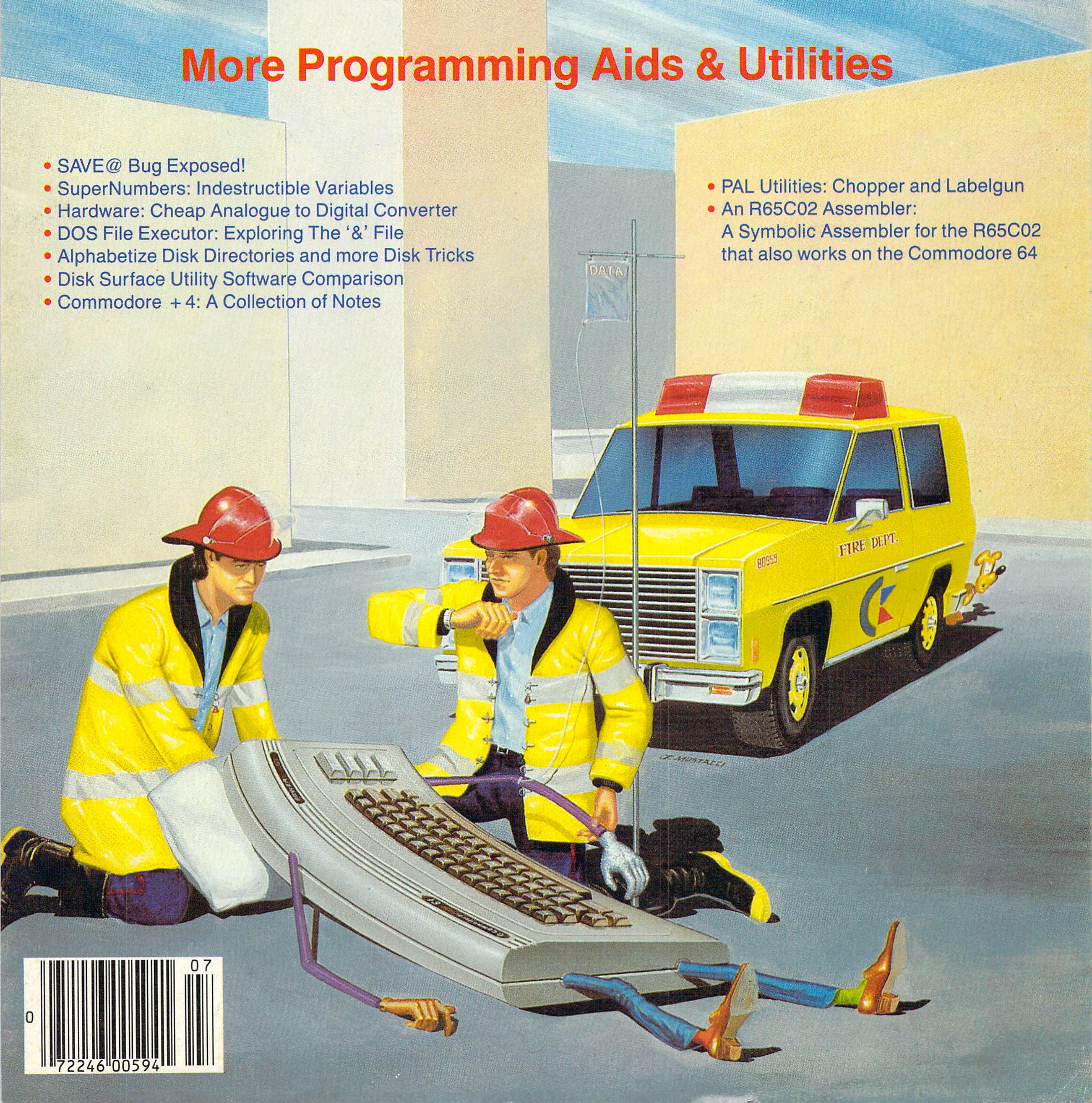
The Tech/News Journal For Commodore Computers

95% Advertising Free! July 1985: Volume 6, Issue 01. $2.95

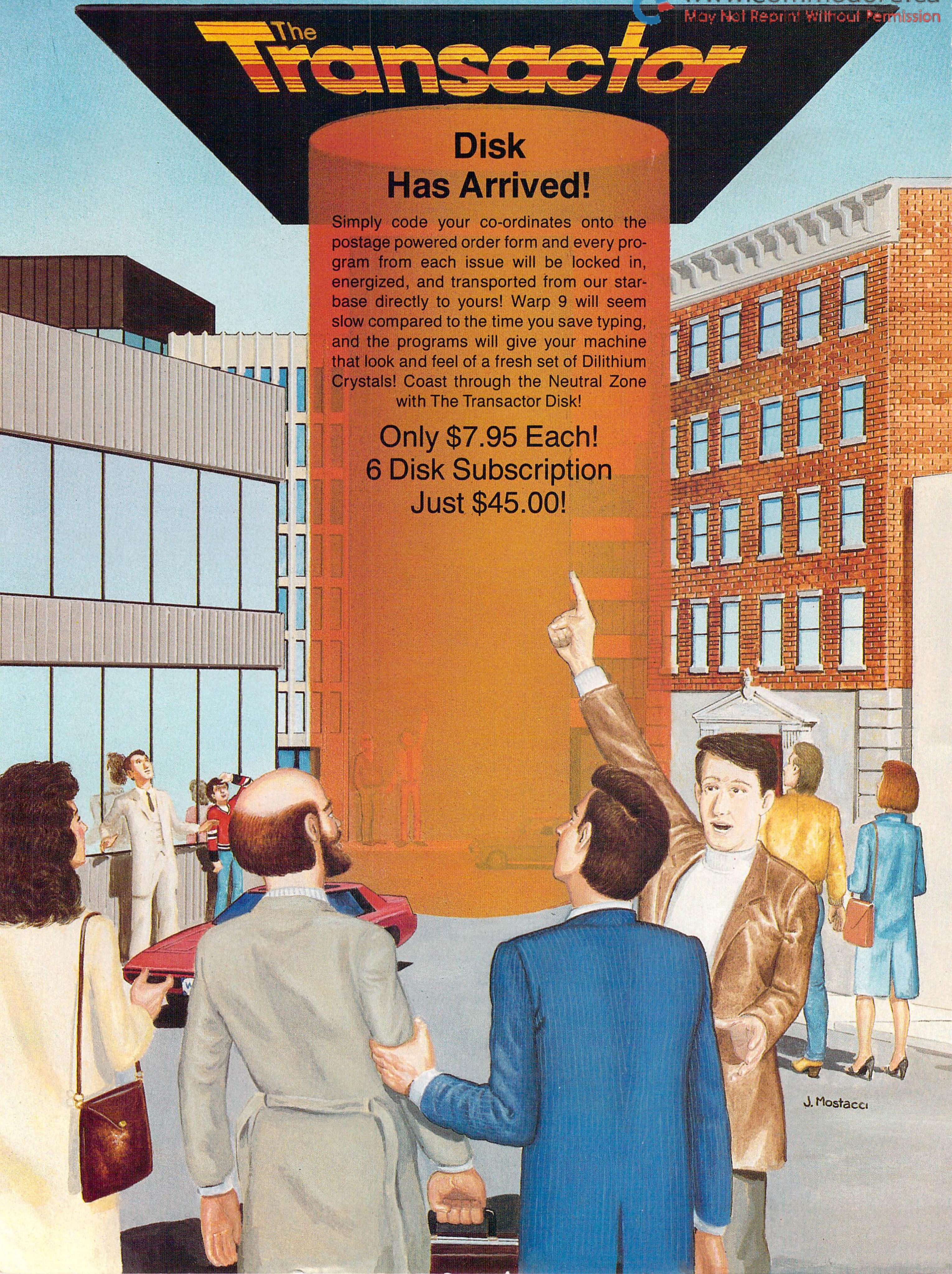
More Programming Aids & Utilities

- SAVE@ Bug Exposed!
- SuperNumbers: Indestructible Variables
- Hardware: Cheap Analogue to Digital Converter
- DOS File Executor: Exploring The '&' File
- Alphabetize Disk Directories and more Disk Tricks
- Disk Surface Utility Software Comparison
- Commodore +4: A Collection of Notes

- PAL Utilities: Chopper and Labelgun
- An R65C02 Assembler:
  A Symbolic Assembler for the R65C02
  that also works on the Commodore 64



0 72246 00594 07



The Transactor

Disk Has Arrived!

Simply code your co-ordinates onto the postage powered order form and every program from each issue will be locked in, energized, and transported from our starbase directly to yours! Warp 9 will seem slow compared to the time you save typing, and the programs will give your machine that look and feel of a fresh set of Dilithium Crystals! Coast through the Neutral Zone with The Transactor Disk!

Only $7.95 Each!
6 Disk Subscription
Just $45.00!

Volume 6
Issue 01
Circulation 64,000

The Transactor

Note:
Before entering programs, see "Verifizer" on Page 19

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Lana Humphries

Contributing Writers
Gary Anderson
Harold Anderson
Don Bell
John Bennett
Daniel Bingamon
Anthony Bryant
Jim Butterfield
Daniel Chernoff
F. Arthur Cochrane
Gary Cobb
Elizabeth Deal
Domenic DeFrancesco
Tony Doty
Bob Drake
Mike Forani
Jeff Goebel
Gary Gunderson
Thomas Henry
David A. Hook
Chris Johnsen
Garry Kiziak
Michael Kwun
Scott Maclean
Stacy McInnis
Chris Miller
Brian Munshaw
Michael Quigley
Howard Rotenberg
Louis F. Sander
K. Murray Smith
Darren J. Spruyt
Aubrey Stanley
Nick Sullivan
Charles Whittern

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

```
print"          flush right"   – would be shown as –   print"[space10]flush right"
```

Cursor Characters For PET / CBM / VIC / 64

Down –	q	**Insert** –	T
Up –	Q	**Delete** –	t
Right –	]	**Clear Scrn** –	S
Left –	[Lft]	**Home** –	s
RVS –	r	**STOP** –	c
RVS Off –	R		

Colour Characters For VIC / 64

Black –	P	**Orange** –	A
White –	e	**Brown** –	U
Red –	£	**Lt. Red** –	V
Cyan –	[Cyn]	**Grey 1** –	W
Purple –	[Pur]	**Grey 2** –	X
Green –	↑	**Lt. Green** –	Y
Blue –	←	**Lt. Blue** –	Z
Yellow –	[Yel]	**Grey 3** –	[Gr3]

Function Keys For VIC / 64

F1 –	E	**F5** –	G
F2 –	I	**F6** –	K
F3 –	F	**F7** –	H
F4 –	J	**F8** –	L

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725–050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209, 716-884-0630. **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 876 4741. From Toronto call 826 1662. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ.

Back Issues: $4.50 each. Order all back issues from Milton HQ.

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4, Issues 04, 05, 06, Vol 5 Issue 03
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05

Quantity Orders:

MICRON DISTRIBUTING

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 438 8854

Micron Distributing
409 Queen Street West
Toronto, Ontario, M5V 2A5
(416) 593 9862
Dealer Inquiries ONLY:
1 800 268 9052
Subscription related inquiries
are handled ONLY at Milton HQ

U.S.A. Distributor:

Capital distributing

Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos of authors or equipment, and illustrations will be included with articles depending on quality. Diskettes, tapes and/or photos will be returned on request.

The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.

Start Address

Since I don't have a lot to say about any one subject, I'm going to blabber about a two or three.

First off, how do like our new logo? We had been pondering the idea for some time and felt this issue was the one. New volume, new look, why not. Last issue is probably the last time you'll see the familiar trapezoid with the rounded top corners. This was the shape of the label on the front panel of the original PET 2001 and it has been used to border "The Transactor" since the very first issue. Many times have I paid tribute to the legacy of the PET and the border was was just another link to the past. However, it's time to let go. The RAMs in my 2001 go tired now after about 20 minutes, but they hold a lot of memories; the cute little keyboard, disappearing cursors, snow, and the second cassette unit was state-of-the-art when it arrived. Remember "Karl J."? Back then, when I was ordering all of 850 copies of the next Transactor, I never thought I'd be changing the logo for the third time.

June the first marks the third year of Transactor independence. A lot has happened in those three years. Recently we passed the 10,000 paid subscribers mark and we're shipping 54,000 mags to the newstand market. The Transactor Disk is now firmly settled in our routine and, as I'm writing this, The Complete Commodore Inner Space Anthology is at the book bindery. My "pack-rat" tendencies were in their most advanced stages and The Anthology is the sum total of the paper jungle that had almost assumed control of my computer room.

Some of you might be asking, "What happened to the Networking and Communications Issue?" and that would be a good question. We had so much material for our last issue, Aids and Utilities, that we thought we would do another one. It seemed a shame to postpone the articles that wouldn't fit into our page limit and the communications issue has a complete line-up of other articles that probably wouldn't allow enough space for the surplus. We'll be back onto our original schedule with the next Transactor and I apologize to the authors of material submitted for Networking and Communications that expected to see their work this time around. One bonus however. . . this issue has allowed us just enough extra time to make additional "stage 1" improvements to the next one. Chris, Richard, and I are really looking forward to making a Communications exposefhat will re-introduce the telecomputing fad to all, us included.

Jack Tremiel and Atari are a hot topic these days. The media seem to enjoy the mileage they get from Jacks flamboyant attributes. (But really guys, don't you think the Atlantic Acceptance incident is growing a little stale?) The defection of several Commodore employees to Atari is the latest publicity cache for Tramiel. At present I can think of at least nine of Commodore's key bees that have buzzed off. Of course you know what happens when the stinger is gone. . . However, Jack deserves a lot of credit. I mean, let's face it - the shock wave of the personal computer is over and I doubt Atari, Commodore, or any of the others can rekindle it. But if anyone can bring Atari back to a comfortable existence, it's Jack Tremiel.

Atari has some fairly bold gladiators lined up but even the one and only J.T. will have a tough time matching the sonic boom of the VIC and 64. No, I believe the market has become mature to the sensationalism; games are no longer enough for the potential micro buyer. Atari's new machines aren't games oriented but it's hard to imagine another market ever as big. The ST series, nicknamed the "Jackintosh" are aimed at, you guessed it, the Macintosh. Although Apple has enjoyed moderate success with this entry, Atari may be in for a workout. After all, this "type" of equipment needs a *following* to maintain continued success. The ST's late start means they forfeit the following to the Mac. The price may turn things around for Atari's ST future, but Commodore is not likely to soon be dethroned, although Atari claims they aren't after Commodore's market. Sure.

On the software scene, "Counterfeiting" seems to be the latest buzz. Hit hardest was probably the Flight Simulator package by Sub-Logic. It's hard to believe that anyone would go to the lengths required to duplicate a package as complete and as detailed as Flight Simulator. Being slightly involved in the printing business, it soon becomes clear that in order to get a half decent per piece cost you cannot aim at any less than 10,000 copies. Just how many counterfeits were produced is not known at this time and I hesitate to release details about those involved (the RCMP will be announcing arrests in just a few days from now) but the lost revenue for Sub-Logic must have been staggering, not to mention the other companies who have had their packages second generationalized. It's a pity that those so close to the industry in which we occupy ourselves are not more understanding to the amount of effort that goes into creating a product to be proud of. It's also a pity that vague legislation surrounding computer software may well result in a mere slap on the wrists. In fact, based on what information I've managed to gather, it looks like several of those involved may never be implicated.

However, there's nothing as constant as change, I remain,

Karl J.H. Hildon, Managing Editor

Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one–year's subscription to The Transactor

VIC/64 Clear Screen Line

There's an easy way to clear any line on the screen right from BASIC:

C–64: POKE 781,line:SYS 59903
VIC–20: POKE 781,line:SYS 60045

here 'line' is the screen line to be cleared, in the range 0 through 24.

Move Screen Line

There's another general ROM routine that can be easily put to good use: this one copies 40 bytes from a specified point on the screen to the current cursor position.

C–64: POKE 780,hi: POKE 172,lo: SYS 59848

Where lo,hi represents the beginning of screen characters to copy (lo + 256*hi must be in the range 0 to 999).

For the **VIC–20**, use SYS 59990

While We're Exploring ROM Routines. . .
The Memory Transfer Subroutine

Often you may wish to move a range of memory, for example to transfer screen or hi–res memory in or out. BASIC is too slow, but you don't need to write a general–purpose memory transfer routine in machine language (although many of you probably have by now, anyway).

The Kernal itself has to move memory around a lot, for example when inserting or deleting BASIC program lines, and there is a memory transfer routine built into ROM in all machines. The article in this issue "LOAD & RUN" uses the routine to transfer a machine language program to its proper address.

Before calling the routine, there are three addresses which must be supplied: source start, source end + 1, and destination *end + 1* (not start + 1). The vital information follows.

	src start	src end + 1	dest end + 1	subrtn entry point
PET (2.0)	$5C	$57	$55	$C2DF
CBM (4.0)	$5C	$57	$55	$B357
C–64	$5F	$5A	$58	$A3BF
VIC–20	$5F	$5A	$58	$C3BF

Cheap Video–Game Dept.

RACER is the concept of John Durko of Toronto, Ont. This has got to be one the simplest game programs around that is so much fun to play. The version below is for BASIC 2.0/4.0 PETs (40 or 80 columns) and is only 13 lines long. If that's too long for you, try this slightly compressed version — you have nothing to lose typing in 4 lines of code!

RACER for 40 or 80 column PETs: (clear screen before running)

```
HB | 1 w=10:y=21: t=20-w/2:x=21:n=y:l=32768+y*80
   |   :forr=1tol:fori=1ton:pokes,32
FF | 2 printtab(t)"(*"spc(w)"*)":t=t+c:s=l+x:getx$
   |   :ifpeek(s)<>32thensr=r:r=l:goto7
KH | 3 pokes,160:k=peek(151):x=x+(k=180)-(k=182)
   |   :c=sgn(c-2*((t<5)-(t+w>40))):next
MG | 7 n=rnd(1)*10:c=int(rnd(1)*3)-1:nextr
   |   :print"q*** crash! score= "sr;sr*y*(20-w)
```

Steer left and right with the 4 and 6 keys (or as indicated with VIC/64/16/+4 versions) to stay within the track as it changes its path. The variables 'W' and 'Y' in line 1 control the width of the track and the screen line that the "car" appears on, respectively. A narrower track makes for a more challenging game, as does a lower screen line (greater value of 'Y'), since you have less time to react to changes in the track.

The "long" version of the program below prompts you for track size and car position, providing defaults. It can also be modified to work on the C–64, VIC–20 (joystick or keyboard),16 or +4. In fact, this program could be made to work on any machine that runs BASIC; all you have to know is the location which stores the current key pressed, and where screen memory lies.

After you crash, your score is given as two values; the first value indicates the number of "turns" that you survived, and the second is scaled to take into account the track width and car vertical position.

Possible enhancements? Dynamically change the width of the track; change the "speed" of the car by changing its line position from joystick or keyboard controls; put random "obstacles" in the track which must be avoided; write it in machine code!

One last point: if you have an 8032, you can speed up the track by setting the top of a window on a line above the car.

Full–featured RACER for PETs:

```
OE | 100 rem"  RACER  -jd/cz
DH | 110 print"S** use 4/6 keys for left/right **"
IB | 120 input"qtrack width  (1-20)   10]]";w
```

```
EA   130 input " car position (1–23)    20]] " ;cy
BP   140 sc = 32768 + 80*cy:kbd = 151:lf = 180:rt = 182
          : rem** machine–specific **
EK   150 b = 1:e = 38:tw = w + 4:tl = 20–tw/2:cx = 20:s = sc:n = cy
NF   160 for i = 1 to n:poke s,32:printtab(tl) " (* " spc(w) " *) "
          :s = sc + cx
IJ   170 if tl<b or tl + tw>e then inc = (tl + tw>e)–(tl<b)
CD   180 get z$:if peek(s)<>32then220
DG   190 poke s,160:k = peek(kbd):cx = cx + (k = lf)–(k = rt)
LA   200 tl = tl + inc:next:sr = sr + 1
KO   210 n = rnd(1)*10:inc = int(rnd(1)*3)–1:goto160
PK   220 print " qq*** you crashed!! ** — score: "
          sr;sr*(40–tw)*cy:print " q run QQQ "
```

C64 mods:

```
   140 sc = 1024 + 40*cy:kbd = 56320:lf = 123:rt = 119
        : rem for joystick
or 140 sc = 1024 + 40*cy:kbd = 197:lf = 51:rt = 0
        : rem keyboard; home/del keys
   165 poke s + 54272,1: rem colour memory (white " car " )
```

VIC–20 mods:

```
   140 sc = 7680 + 22*cy:kbd = 37151:lf = 122:rt = 118
        : rem joystick up/down
or 140 sc = 7680 + 22*cy:kbd = 197:lf = 62:rt = 7
        :  rem home/del keys
   150 b = 1:e = 20:tw = w + 4:tl = 11–tw/2:cx = 11:s = sc:n = cy
   165 poke s + 30720,0: rem black car
```

16/+4 mods:

```
   140 sc = 3072 + 40*cy:kbd = 198:lf = 48:rt = 51
        : rem* crsr left/right keys
```

NEW facts

Many programs, after loading some machine code and changing BASIC pointers to protect the top of memory, contain the following line of code:

```
NEW : CLR
```

This is dumb for two reasons:

1) The NEW command does a CLR automatically, free of charge. In fact, the CLR routine comes directly after the NEW routine in ROM, and NEW just falls through into CLR.

2) Any statement appearing after a NEW, even on the same line, even in direct mode, WILL NOT BE EXECUTED. Thus, NEW:CLR, NEW:PRINT, and NEW:SYS64738 all do the same thing: a NEW.

So even though using the command NEW:CLR doesn't do any harm to your programs, it sure doesn't do any good. Leave off the CLR and let's put an end to this custom before it's carried on to future generations.

C64 Programming Tip

It is often desirable to be able to halt a machine language program by pressing a key. The usual approach is to use the "check stop key" routine at $FFE1 and check the Z flag to see if the STOP key was pressed. This approach will not work, however, with programs that disable IRQs, since the keyboard isn't being scanned.

Another possibility is to actually scan the keyboard for a specific key by storing the keyboard row number (inverted) in location $DC00, and checking location $DC01 for the value of the desired key. The problem with this approach is that the key must be down when the scan takes place for it to register. To make sure that the key is detected, it would have to be scanned frequently and possibly in several places throughout the program. That can be time-consuming, and using the keyboard in this way also interferes with operation of the joysticks.

A good solution involves using the RESTORE key and NMI vector. Point the NMI vector to a routine which sets a flag (stores a nonzero value in some memory location), then jumps to the normal destination of the NMI vector. Whenever the RESTORE key is struck, it generates an NMI and this flag will be set. The machine language program can check the flag and exit if it is set. That way, no matter what the program is doing when the RESTORE key is hit, it will eventually find out about it when it gets around to checking the flag. When the flag is found to be set, it should be cleared (set to zero) before exiting to prepare for future runs.

Defaults in INPUT Statements

When using INPUT statements it's nice to provide the user with a reasonable default so that he/she can just press return in most cases. Here is a good way to do it:

```
10 input " Drive number   0]]] " ;dr
```

After the prompt message comes three spaces, the default, followed by three cursor–lefts. If the default is more than one character long, increase the number of spaces and cursor–lefts accordingly.

An added bonus of this technique is that you can reject invalid entries in a very nice way. For example:

```
20 if dr<0 or dr>1 then print " Q " ;:goto10
```

This will simply ignore any input other than 0 or 1, without any drama.

350800 And Its Relatives   Elizabeth Deal, Malvern PA

The "350800 poison number" mentioned in the BITS and PIECES of the Vol 5, Issue 5 Transactor is indeed a member of a class of neat numbers with high byte 137 (in fixed point format). This is due to a little bug in the PET, VIC and the C64 computers, as well as the APPLE. The bug does NOT exist in the RADIO SHACK computers, nor in the Commodore's B–128, Plus 4 and C–16.

Tracking the story down can be fun, and shows that the culprit is an intended error–exit from the routine that converts numeric characters in the BASIC text to a fixed point number. This routine is used for many things, one of them being BASIC line number entry. If you follow the PET code (below), beginning where it says 'START HERE', you'll see that when a number's high byte exceeds hex $19 (25 decimal) the intent is to abort. But . . . the PET code jumps into the middle of the ON routine. Now when, and only when, your entered number has a high byte of 137 (hex $89) we fall into the trap. We pull an item off the stack and crash on trying to execute the code. By now, the action–address has been mangled. In the upgrade PET, we end up in zero page, $C8 being the remaining byte on the stack. Good fun.

You can look up the details in the Butterfield's maps – the 'perform ON' routine can be your starting point for disassembly. After ON comes the 'get fixed point number' routine which contains the multiply–by–10 code which is pretty long, and not reproduced here. The story ends with the call to the CHRGET routine and the loopback. Here are two disassemblies, one from the Upgrade PET (problems) and one from the B–128 machine (all fixed).

```
;perform ON
c853  20 78 d6 jsr  $d678
c856  48       pha
c857  c9 8d    cmp  #$8d   ;gosub token?
c859  f0 04    beq  $c85f
c85b  c9 89    cmp  #$89   ;goto?. . .a trap into which we
                             fall
c85d  d0 91    bne  $c7f0
c85f  c6 62    dec  $62
c861  d0 04    bne  $c867
c863  68       pla         ;<— herein lies the stack
                             problem.
c864  4c 02 c7 jmp  $c702  ;dispatch command/rts
;
c867  20 70 00 jsr  $0070
c86a  20 73 c8 jsr  $c873
c86d  c9 2c    cmp  #$2c
c86f  f0 ee    beq  $c85f
c871  68       pla
c872  60       rts
;
;--->START HERE - get fixed point number
c873  a2 00    ldx  #$00
c875  86 11    stx  $11
c877  86 12    stx  $12
;big loop - do while characters are numeric
c879  b0 f7    bcs  $c872  ;all done - exit
c87b  e9 2f    sbc  #$2f
c87d  85 03    sta  $03
c87f  a5 12    lda  $12
c881  85 1f    sta  $1f
c883  c9 19    cmp  #$19   ;is the number over 63999?
c885  b0 d4    bcs  $c85b  ;yup. . .get out (into the trap!) **
c887  a5 11    lda  $11
;. . . etc - multiply by 10 routine
c8a7  20 70 00 jsr  $0070  ;chrget - from basic text
c8aa  4c 79 c8 jmp  $c879  ;and loop back
```

Here we have the same thinking, but this time in the B-128. An identical code is in the Plus 4 machine, see Butterfield's Plus 4 map in vol.5, issue 5. It's clear as daylight that the problem has been fixed. Entering a BASIC line: 350800 print "what's the point?" just blows off to SYNTAX ERROR. If you follow this disassembly, you'll see the little fix:

```
;perform ON
f8d2b 20 d6 b4 jsr  $b4d6
f8d2e 48       pha
f8d2f c9 8d    cmp  #$8d   ;gosub?
f8d31 f0 07    beq  $8d3a
f8d33 c9 89    cmp  #$89   ;goto?
f8d35 f0 03    beq  $8d3a
f8d37 4c 4f 97 jmp  $974f  ;nope. . .go to Syntax Error
f8d3a c6 75    dec  $75    ;yes
f8d3c d0 04    bne  $8d42
f8d3e 68       pla
f8d3f 4c aa 87 jmp  $87aa
;
f8d42 20 26 ba jsr  $ba26
f8d45 20 4e 8d jsr  $8d4e
f8d48 c9 2c    cmp  #$2c
f8d4a f0 ee    beq  $8d3a
f8d4c 68       pla
f8d4d 60       rts
;
;START HERE - get fixed point number
f8d4e a2 00    ldx  #$00
f8d50 86 1b    stx  $1b
f8d52 86 1c    stx  $1c
;big loop - while characters are numbers
f8d54 b0 f7    bcs  $8d4d  ;normal exit, not a number
f8d56 e9 2f    sbc  #$2f
f8d58 85 0c    sta  $0c
f8d5a a5 1c    lda  $1c
f8d5c 85 22    sta  $22
f8d5e c9 19    cmp  #$19   ;over 63999?
f8d60 b0 d5    bcs  $8d37  ;yes, jump out to Syntax Error
f8d62 a5 1b    lda  $1b
;. . . etc multiply by 10
f8d82 20 26 ba jsr  $ba26  ;chrget-next character
f8d85 4c 54 8d jmp  $8d54  ;loop back
```

Tickertape — Dave Smart, Russell, Ont.

Here's a good little ticker-tape routine — it can be used on any machine, but the 64 version has 'tick-tick' sound efects. The nice thing about this routine is that it can handle strings up to 255 characters long.

Usage notes: just put the string to be scrolled in Q$, and GOSUB 100; you can vary the speed by changing the '65' in line 160; line 105 must be changed for 80 or 22 column screens; the string Q$ is left altered by the routine.

```
100 rem tickertape subroutine-dave smart
105 ln=40: rem # of columns in screen
110 for l=1 to ln: q$=" "+q$: next
120 for l=1 to ln: q$=q$+" ": next
130 for l=1 to len(q$)-ln+1
150 print mid$(q$,l,ln)"Q";
160 for t=1 to 65:next t,l
170 return
```

Add the following lines for sound effects on the C-64:

```
106 poke 54273,70:poke 54278,249: poke54276,17
    : poke54276,16
140 poke 54296,15:poke 54296,0
```

Debugging Aid Update — R.C. Marcus, Agincourt, Ont.

Mr. Marcus writes,

"In the latest issue of Bits & Piecesm issue 5 vol. 05, was included a handy 'Built-in debugging aid" for BASIC 4.0 machines.'

I would like to add more information along this line. This is a BASIC routine and is in BASIC ROM of the VIC, 64 and by your listing of the 16/+4 memory map, page 25 of the same issue, it resides there as well.

It is referred to as Print 'IN' routine and resides in the following locations: VIC, 56770; 64, 48578; 16/+4, $A453.

A SYS to the appropriate location will provide this handy feature."

Easy Program UN-NEW After Reset

Last issue's Bits & Pieces presented "REGAIN" to restore your BASIC program after a system reset or a new. If you crash and reset, but don't have REGAIN in memory, you can use this method, sent in by Alan Clooney of Cranbourne, Australia:

```
poke 2050,1:sys 42291:poke 46,peek(35)
      :poke 45,peek(781)+2: clr
```

If this gives an error message then

```
poke 45,peek(781)-254:poke 46,peek(46)+1: clr
```

1541 DOS Crash With REL Files — John Menke, Mt Vernon IL

"Failure to properly close a relative file crashes the 1541's DOS. Subsequent disk operations will give unpredictable results, probably damaging other files, the directory, and BAM. Use of the initialize command 'I' only apparently and deceptively sets things right. The DOS will not work correctly after the initialize in this case. It must be reset with 'UJ' or 'U:' or by turning the drive off momentarily."

1541 DOS Wedge Tips — John Menke

"Most of the 1541 wedge commands work in program mode with a minor syntax change; whatever follows @, >, /, ↑ or ← must be in quotation marks. There appears to be a problem only with the % command when used in this way. These commands must be on their own on a separate program line; in some cases they'll work as the last statement on a line. Variables are not recognized as file names by the wedge commands. The following program lines illustrate the three most useful applications which I have found for these commands:

```
10 @ "$"
20 <
30 ↑ "program name"
```

Line 10 lists the directory, and as usual the space bar stops/continues the listing on the screen. Line 20 reads the disk error status and prints it to the screen. Line 30 is useful in loaders, or in chaining programs."

One–Line Decimal ⇋ Base B — A Hooyer, He Soest Holland

To convert from decimal value 'D' to base 'B' with output length 'L':

```
8 n$ = " ":fori = 1tol:h = d:d = int(h/b):h = h–d*b
  :n$ = chr$(h + 48–7*(h>9)) + n$:next:return
```

Result in 'N$'. To convert 'N$' from base 'B' to decimal, output in 'D':

```
9 d = 0:fori = 1tolen(n$):h = asc(mid$(n$,i))–48
  :d = d*b + h + 7*(h>9):next:return
```

Examples:
From decimal to hex: d = 4096:b = 16:l = 4:gosub 8:print n$
From binary to decimal: n$ = "1000101":b = 2:gosub 9:print d

Restore Key Fun — Scott MacLean, Georgetown, Ontario

Most programmers know that

```
poke 808, 205
```

will disable RUN STOP/RESTORE, but it has other minor side effects, such as disabling the LIST function etc. The RESTORE key is tied directly into the 6510 NMI (Non–Maskable Interrupt) line. When you press it, it jumps through a vector at 792–793 to wherever it goes, does its stuff, and has its fun. It is possible to replace the vector with say, 49152:

```
poke 792, 0: poke 793, 192
```

Ha, now when we press RESTORE, it jumps to location 49152, and starts doing things. Let's put an RTI (Return from Interrupt) instruction there with:

```
poke 49152, 64
```

Now press RESTORE! Neat! Wow! When you press it, nothing happens! Try RUN STOP/RESTORE. Still nothing. Try this:

```
poke 49152, 32: poke 49155, 64: poke 792, 0: poke 793, 192
X = 226: Y = 252
poke 49153,X: poke 49154,Y
```

Press RESTORE. Or this:

```
X = 234: Y = 232   Or:   X = 34: Y = 228
```

Quick Note: The 8250 Dual Drive records files in sectors spread 5 apart as opposed to 3 apart in the 4040, 8050, and 1541.

Screen Save Update — R.C. Marcus, Agincourt, Ont.

The handy method to save the screen which appeared in the recent issue under Bits and Pieces, titled "Put Mental Notes on Disk (or tape)!", can be indeed a useful tool. It can be used as well on the VIC with the appropriate changes to the Kernel SYS's.

Unfortunately, with the limited screen of the VIC this direct mode entry takes up three to four lines, depending on the length of the filename; plus another for the "SAVING . . . " message, so in all, a possible five lines are taken or 110 character positions. As the screen has only 506, this is a fair amount of space just for the saving instructions.

The attached short program is for the VIC with any memory configuration. It provides screen save with a single command; SYS 828. The program saves up to the second last line of the screen, so by placing the SYS command on the second last line, as instructed by the BASIC loader program, it does not scroll the screen or appear when the screen is recalled. This gives 462 screen positions for message characters.

This program places a machine language program in the tape buffer, so it can only be used in disk–based systems. The filename is built into the routine as "scr(shifted space)ml" and appears in the disk directory as "scr"ml; the ml is the reminder I use to indicate that the load command must include the ",1" to indicate a relocating load.

To recall a saved screen, LOAD it in with: LOAD"scr",8,1. Putting the LOAD on line 18 will cause the "READY." to appear near the bottom of the screen and not mess up the message.

The routine does the save without the SAVING message to conserve screen space, but will print error messages if they arise. It also saves with the replace option, so be sure to rename your old screen file if you don't want it wiped out by the next one you save.

```
100 rem basic loader for screen save
110 for p = 828 to 879 : read a : poke p,a : cs = cs + a : next
120 if cs<>6685 then print "error!..in data"spc(14)
     "statements." : end
130 print"'sys828'...on the 2nd"spc(11)"last line to"spc(10)
     "save screen."
140 data 169,  64,  32, 144, 255, 169,   0, 162
150 data   8, 160,   1,  32, 186, 255, 169,   9
160 data 162, 103, 160,   3,  32, 189, 255, 169
170 data   0, 133, 251, 173, 136,   2, 133, 252
180 data 164, 252, 200, 162, 205, 169, 251,  32
190 data 216, 255,  96,  64,  48,  58,  83,  67
200 data  82, 160,  77,  76
```

Auxiliary Bits

(for the +4/C16, B Series, 1541, and 8050)

Elizabeth Deal
Malvern, PA

+4 and C16 Bits

These computers are miracles. What follows are some notes I have which may well be unintelligible to the beginners, but can be of use to someone familiar with other Commodore machines. The User's manual is rather complete, so these are just additional comments:

Character strings are handled differently than in previous machines: there is no such thing as pointers into a program. All strings declared inside a program are copied to RAM (usually hidden under ROM). There are no garbage collection delays. Additionally, new functions can be done with the strings:

1. The first one is assignment statement with MID$ on the left. Yup, you're seeing it correctly. It's now OK to code:

```
MID$(a$,4,2) = "de"
```

This will change whatever was in positions 4 and 5 to be "de". Can you see why strings can't live inside a program? Would be a programming nightmare if they did.

2. INSTR function returns a position of one string within another, so

```
INSTR$("xyz","y")
```

returns 2. You can also specify a starting position for the search. The old code

```
forj=1tolen(a$):ifx$=mid$(a$,j,1)thennextj
```

is no longer needed, and the INSTR function is instantaneous! Hurray to Commodore.

Tape is incompatible with other CBM machines. The connector is different, but that's the small part. The timing is different. It seems that the writing goes at about half the speed of the previous units. The code to accomplish tape writing and reading is enormous. Reading is particularly difficult because the TED chip functions differently: there is no such thing as detecting a negative transition – all transitions have to and are being detected in software. The screen is turned off to permit 1.7 mhz operation. Still, it is a slow process.

Tape errors are funny. If you happen to position a tape to the very tippy–end of a program you don't want to load, the computer reports BREAK error (#30) and does not go on to look for the program you do want. Using error trapping (TRAP statement exists in the language!) is the way to go in program mode.

Generally, error numbers when used with tape are wrong. You may get a DEVICE NOT PRESENT ERROR, when you think it should be a FILE NOT FOUND ERROR. You invariably get BREAK ERROR when the end–of–tape header has been read in. This would be only a cosmetic nuisance, were it not for the fact that a STOP–key also causes a BREAK error. It's hard to tell one from another

There is lots of RAM in the machine, and one tends to play a lot of hide–and–seek games in finding things. Some clues:

1. Page 4 contains various indirect routines that permit taking bytes from ROM or RAM. These routines are used by BASIC.
2. In page 7, specifically at $7d7 is an equivalent routine for use by the machine code monitor.
3. BASIC PEEK returns a byte from RAM. Machine Language Monitor returns a byte from ROM. MLM normally saves only RAM, you can't save ROM. BASIC SAVE normally saves RAM. LOAD loads bytes into RAM, as you'd expect. Sometimes you may wish to change the defaults. It can be done:

(a) To PEEK ROM from BASIC: modify a routine in page 4 (at $0494) to ignore the store instruction:

```
poke1176,44 : now peek ROM : poke1176,141
```

This is fairly safe, so long as you DO NOT WORK ANY STRINGS BETWEEN THE TWO POKE1176 INSTRUCTIONS. This, for instance, is the only way you can get at the character generator ROM from BASIC, as far as I can tell.

(b) To peek/save ROM from the MLM, set bit 7 in the byte at $7f8. Incidentally, the monitor has a nice feature – ">7f8 80" is all you need to type in, the ">" sets bytes and displays just one line of memory.

The MONITOR is nice, it's almost like SUPERMON. But there is a serious bug – they chopped the TRANSFER command: T with overlapping addresses does not work in one direction – when the destination is higher in memory than the source. The bytes just write one over another and you lose all your work. A cure: first Transfer to another, non–overlapping area, then do a second transfer to where you wanted to go in the first place.

Colour memory, as in the C64, contains the colour codes for the 1000 screen bytes. One difference, bit 7 is the flashing bit. Funny things happen when you load the C64 colour map into, what's now called, screen attributes map in the Plus 4. You get flashing for nothing.

To change the colour attributes from the C64 POKEs into the COLOR statements, you'll need to add one to each value, as the Plus 4 colour numbers are from 1 to 16. POKE values are still 0–15, but there is little reason to use them.

The keyboard is a delight. Perhaps a bit too soft, but easy to use. The ESC sequences are a joy to use. There is even a pause–all–output key: two keys actually – CTL–S, with any other key restarting the output. There is only one problem: if you use CTL–S during a program run, and use a subsequent GET statement, the S will appear to the GET statement as a real input – I think it's a bug – the keyboard buffer isn't cleared, so you'll have to do it yourself.

Programmable function keys are useful. Unlike in the B machine, most of them have a carriage return at the end. I don't like this feature, but it can be easily changed. Unfortunately, the keys are active inside a running program. Watch out here: if you use GET, and the user pushes the DIRECTORY key, all the letters in 'DIRECTORY' including the carriage return will be delivered to the GET! If you don't want it to happen, there is a way to disable the function keys: either set them all to null (" ") inside a program and redefine at the end, or POKE their lengths to zero.

The default colours and luminances for the sixteen colour keys are in RAM. They are in a table in page 1 at $113. You can change them as you wish. Machine code people who love to POKE the stack (auto–run programs!) will have to stay away from this area.

Some of the structured BASIC statements are splendid. For instance, the DO WHILE construct permits you to code a loop that will never execute (FOR–NEXT loops always run at least once, unless you test and skip around). The interpreter seems to be looking ahead, almost like a compiler: it skips the loop and all the loops inside it. EXIT permits leaving a loop early. LOOP UNTIL tests a condition at the end of a loop. What more can we ask?

The character table is half the size of that in the C64. Reverse characters aren't in ROM, they are software–generated. You may have to take this into account if you convert programs from the C64 to the +4/C16 machines. Pointing the character base address is simple. It does require POKEs, a rare event in this machine: using the BASIC method (above) or the monitor transfer command, move the characters to any RAM. Then tell the TED chip about the move: tell location $ff13 the page number of the start of your character definitions, then clear bit 2 at location $ff12. That's all there is to it. Much simpler than in the C64. Incidentally, it is perfectly all right to speak in semi–hex to the BASIC interpreter, hence

```
POKE DEC("FF12"), DEC("71")
```

will tell the chip the character base is at $7000. What's that 1 doing in $71? No connection, nothing. Nothing is a one. I don't know why.

When you play with non–ROM character set, a nasty thing can happen: an exit from the monitor or any error in BASIC resets things only half way back to normal. So the screen becomes a mess. Several solutions: TRAP all errors in BASIC. Do not leave the monitor (guarantees learning machine code by the total immersion method). Hold STOP and push the little reset button. Define a function key to (blindly) type the reverse maneuver to set the character base to the default again. Enjoy the crazy screen sight and push some keys while you do so – it's actually an interesting display.

It is possible to move the screen memory anyplace in RAM. The TED chip needs to be told of the move, of course. $FF14 register is the place to use. However, I know of no way to print on the screen when it's not in the standard location. You can POKE it, you can flip it, you can do all sorts of things with the relocated screen, but no printing. The print command ($FFD2) delivers bytes to the default location and only there. It should be possible, if you must print on a relocated screen, to reroute output to your own routine (page 3 vectors) but I doubt that it's worth the trouble.

The GRAPHIC split screen always splits five lines from the bottom. The bottom five lines are in the text mode, the top is bit–mapped. The constant which controls the split raster line is coded in ROM, hence a bit rough to change. However, there is a link in the interrupt–service code which does permit you to modify the place of the split screen, if you must do it.

Disabling the STOP key is a favourite pastime of many people. It's quite easy on the Plus 4 computer – use a TRAP statement and trap error #30 to resume execution. I know, however, of one situation where the STOP cannot be TRAPped. That is in I/O. Tape LOAD illustrates it quite well, as things are slow: the STOP can be TRAPped after the message "LOADING" would appear, not before. While the computer is searching for a header, it uses another way to test the STOP. It looks directly at the keyboard register in the TED chip, and it never tells BASIC about it. The same is probably true with the serial disk, but it is a bit harder to catch, as things happen faster. A moral: to disable a STOP during I/O use a little machine code, especially if your program uses tape. The whole exercise is almost pointless anyway, as the little reset button lets anyone in. I like that.

B–128, 1541, and 8050 Bits

I wish Commodore would reconsider their decision to drop the B–machines. B–128 is a terrific machine. Sure it's hard to program, but it's fun. It has superb BASIC, superb keyboard, 2mhz clock, it's fast and pleasant to use!

There are an assortment of curiosities about the machine itself and some disk drives:

Happy news: On the B128 the files close themselves! When an error condition causes a disk file to remain open, editing a program line makes the disk whirr a bit and a file gets closed. It's incomplete, but it's not a * file anymore. Clever and useful – if you keep the drive door down, of course.

There is a RESTORE <line number> command in BASIC, just as in +4.

BLOAD "file name" drive,unit,bank,address loads program files and does not cause BASIC to run from the beginning. A splendid feature.

BASIC programs which have machine code tacked on to the end are difficult to manage. They can be run, but don't try to SAVE'em without first fixing the pointers up. LOADing such programs causes the end–of–program pointer to be set to the byte following the three zeros (ouch!). Editing a line (or just pushing a RETURN over a line) on the screen does the same thing.

There appears to be a bug in the screen editor which can unreverse reversed characters such as home, cursor directions, etc. in quotes. This only happens if you insert characters ANYWHERE on a line containing the control characters and only when you use the INST key. If you use ESC–A/C to enter/cancel the insert mode, then the line is not ruined. Something is wrong in the setting of the insert–flag but you can prevent trouble by pushing RETURN twice over such lines if you have used the INST key. Say it again Sam. . .

There is an "initialize the drive" command in BASIC–128. It is DCLEAR D1 (for drive #1). I don't know why, but I keep thinking it can NEW the disk. Funny name.

The DOS built into the 8050–drives that come in the Protecto package is a fairly advanced version number 2.7 as you can see in the sign–on message.

There has been a change in the way character string functions work. While ASC of a null byte still returns ILLEGAL QUANTITY instead of a zero (as in the PLUS 4), ASC of characters outside the string aren't ignored anymore: ASC(MID$(" ABC " ,4)) is now ILLEGAL.

Machine Language monitor is a bit rough to use. Some pointers:

1. You can enter the MONITOR by a call, bank15:sys14*4096 does it. You will never exit the monitor, even pressing the reset button doesn't work. This can be useful if you are messing with page zero and rather not exit to BASIC. The exit address is in $f03f8/9. It can be changed.

2. Normal entry is via a break; bank15;poke6,0:sys6 does the job. Exit is nasty, it clears the screen, among other things. Once again, you can change the reset vector to a better setup.

3. Probably the most annoying feature of the resident monitor is that all error commands default to loading and running machine language programs. A pest.

4. The G (go) command is dangerous: do not try to Go to another bank, the crashes are unreal.

5. Do not use the Z–command. It tries to work a co–processor, whatever that is, which isn't there. Consequently we crash.

6. There is a nice little monitor, called EXTRAMON, on the TPUG disk. It has a fairly clean exit to BASIC, as it does not clear the screen. However, do not use the B–exit if you have run a Go command. Most likely you will crash.

7. Crashing has a new twist. Much of the time you don't really crash. The cursor comes back, and things may appear normal. But a closer look reveals, for instance, that your BASIC text is mangled up, the transfer sequences may have funny bytes put in them, and so on. So – like in the old days, shut off, and start from scratch – even when you see the cursor.

If you store a byte in RAM that isn't there and try to read it back, do something between the two operations. A little bit of time (Jim Butterfield says 14 microseconds) are needed for the address to vanish and a real byte to come through. Otherwise the read operation gives a false result. I've been putting three NOP instructions, that's 12 cycles. That may be cutting it too close.

If you have a mismatch–type of error in READing DATA items, the B–machine reports an error about the DATA line itself, rather than the READ statement.

I suspect that there has been an undocumented change in the way IEEE devices function since the 4040. Things that are plugged in but not turned on cause a bus crash. For instance, a printer that is connected but not on will cause a crash if you try to LOAD or SAVE. Incidentally, the same is true in the Plus 4 machine – two 1541 disk drives, one not turned on, will also crash the system. Can anyone explain this?

Another change is that a 1541 and an 8050 drive must have the complete error message read off the drive. You can no longer look at the first value (first byte of the error message) and quit if it's good. The whole thing has to be read in to that last carriage return. Failure to do so causes strange problems which are hard to debug. In the case of relative files, the light continues to flash on the 1541, but not so on the 8050. Non–relative files and/or the 8050 give no clue. So read the whole message. Again, this change hasn't been documented by CBM, as far as I know, but I've seen the trouble ever since the 1541 was born, and now get the same behaviour in the 8050. Life sure was simpler with my old PET and 4040!

The MPS 8050 drive with DOS 2.7 has a bug: if you try to copy a file from one drive to another, and the file already exists on the destination drive, the disk crashes. BASIC COPY command and the monitor's wedge crash in this way. The only way out is to reset the disk (on/off switch) and then button–reset the computer. The fault doesn't seem to be in the B–machine, since it behaves correctly with the 4040 drive (FILE EXISTS DOS–error). My Upgrade PET also has trouble with the 8050, yet none with the 4040.

Do not trust the writeup of the KERNAL routines in the various guides to the B–computer. Some routines are described correctly, others are copies of the C64 guide and may not function the same. For instance, to read the ST, a major task on the B, you must set the carry bit. If you don't, you'll be setting ST to a value in the A–register. Not a very nice thing to do, when you're reading a file. . .

Due to the zero–page pointers, programs saved from the B 128 do not LIST very well on any other Commodore computer. If you need compatibility, put BASIC higher in memory. PET–type of a setup seems to be the best thing – BASIC at 1025 ($401 in bank 1). The pointer to start of BASIC is in bank 15 at $2c/2d.

Keyword token numbers have been shifting recently. You cannot count on the PRINT USING token, for instance, to be the same in the B machine as on the Plus 4. The standard command (PET vocabulary) numbers are the same, but there is no pattern with the expanded commands. A bit nasty for a program such as LISTER.

Letters

We appreciate feedback from our readers, and we read each and every one we receive. However, due to the volume of incoming mail it is sometimes impossible to reply personally, especially to technical questions and inquiries. Further, many inquiries have similar nature. Those questions which we feel would be of general interest will be included in the letters section. Also, we make every attempt to keep one step ahead of potential questions – keep a regular eye on our News BRK section for these and other important announcements.

Bbits: Two additional little pieces of information about the B–128 that's being closed out through Protecto.

1. The cassette read line is connected to the 6526 CIA at bit 4, $DC0D (56333), similarly to the the '64. The bit is set on a negative transition, and cleared by reading $DC0D. Whether it triggers an IRQ or not depends on how the interrupts are enabled.

This wee bit of info is neither in Transactor V4/5 (p.49) nor in Butterfield's Machine Language book.

2. The video chip in my B–128 is a 6845, not a 6545. The 6845 lacks the auto–increment register of the 6545, but permits interlaced scan as an option which the 6545 does not. So I tried it:

```
POKE 55296,8: POKE 55297,2
```

This puts the display into interlace mode, with each line being duplicated a little bit lower on the odd scans. The idea is to fill in between the lines so that a solid block looks solid rather than striped, and a vertical line looks like a solid line rather than dotted. Well, it works, except that on the monitor I'm using (a cheap BMC), there is a lot of jitter. I believe that most monitors don't like interlace, and I remember that the Ultraterm display card for the Apple (made by Videx) has the same problem; there are just a few monitors that are satisfactory with an interlaced display. But at least it's something to try, and to decide for oneself which mode is preferable.

Charles A. McCarthy St. Paul MN

Diskpleased: With surprise and disappointment I read volume 5, issue 5's news of the advent of the Transactor Disk.

Some other magazines offer their programs on magnetic media, but they are directed in large part or almost in whole toward non–programmers, such as Gazette, Run, and Ahoy!.

Your publication, however, promised only one issue earlier to remain high level. The idea is not just to get and run programs but to read the articles and TYPE in the listings in order to learn programming techniques and tricks. Distributing the completed product defeats the purpose: instant foods have a rightful place in grocery stores and in kitchens but not in schools of haute cuisine.

You have seriously undermined your position. I'll continue subscribing to Transactor in print, but no thank you for the magnetic version.

David W. Tamkin Chicago, Illinois

The choice is yours.

SIDioyncrasies: If you have a place for a HELP section or Questions I would be interested in some information on the audio input for the C64. Specifically; #1: What kind of signal is the port expecting; low like a mike or guitar or an amplified signal like that of a home stereo? #2: From what I read the only mention of the ability to merge is with the synthesizer sound and to run it through the envelope generator. If the envelope generator can modify the sound under computer software then is there somewhere to read the audio in 'in memory'? Bet there is, huh?

Rick Cronee, Jackson, TN

The letters section is as good a place as any for help, so here goes:

1) The SID chip audio input will accept a signal of around 200 millivolts peak to peak. This is typically the kind of level you'd get from an audio component's signal output, for example from the TAPE OUT jack of a stereo receiver. A guitar should work fine, but an unamplified microphone's signal would probably be too weak to compete with the backround noise and output from the SID chip itself.

2) The external input can be put through any of the three programmable filters (low pass, band pass or high pass), which are controlled from addresses $D415 to $D417. The only instantaneous audio signals that can be read are the signal level and amplitude envelope of SID voice 3. These values could theoretically be used as filter or volume values to modulate the external input, but unfortunately there's no way to read the external input itself. You might, however, try to interface your audio source to one of the paddle ports and read the paddle register.

DiskMod Notes: *Recently we have received quite a few letters stating that Jim Butterfield's BASIC DiskMod isn't too friendly with the C64/1541 combo. Inaccurate and partial readings from sectors plus really poor screen editing are the two major complaints. Well, by going through the code, and using it as per Karl's article, everything but line 140 is OK. This particular version was designed to be machine portable with but one minor dependency: the screen memory location. 32768, which is for the PET, should be changed to the correct start address of your machine.*

By reading through the complaints, a simple cure has been found for the problems described. When up and running, the program will ask you for the drive number of the drive in question. Answer it with an 's' for single drives (1541/2031), or 0/1 for the IEEE dual drives. This is very important due to the way Jim reads and modifies disk data. He accesses disk RAM direct, therefore he requires the correct RAM buffer start addess to do it properly. You will notice in the code that he assigns the string variable B$ a value of CHR$(17) as default, but changes it to CHR$(3) if the answer to the drive type is 's'. This is the high byte of the start address, translated into $1100 for the dual drives and $0300 for the single units. Without a correct answer at this point, you will find odd things happening in the read/write department. Perhaps a print statement should be incorporated into the program to state this fact. We leave this to you.

One final note. When the first half of the data has been displayed on the screen with the 40 column machines, you will notice a prompt of 'swap' suddenly appearing. If you want to see the balance of the data from that particular sector, press 's' to swap over to the second half of the sector. Simple, but another point a few of you didn't quite pick up on.

Lock Spurs: *A letter has arrived regarding Jim Butterfield's 'Lock Disk 64' program. The complaint was that it didn't work at all. We checked it out, and everything is fine. But, to use the program, you have to remember that Lock Disk for the 64 works in a rather odd manner. Once fired up it will ask you the name of the program to convert, then the name of the converted program. Given the correct responses, the new locked program is written to disk. After, if the locked program is loaded into the computer without ",8,1", it will just load, but the list will be a new load statement. Run it and the program will be booted correctly from disk. If loaded via ",8,1", it will automatically run once the load is complete. Another neat trick written into the code is a vector change to stop your program from ever ending. If ever your computer goes back to 'ready' mode, for whatever reason it might have, it will automatically re-run the program once again. Your problem was that your new locked program would constantly flash the intro message on the screen once loaded. Sounds like either an end statement in your code, or a syntax error encountered. Check your code and try again. It should work.*

Slipped Disk: I just finished reading your latest issue (Vol. 5, issue 05) and I found Michael Quigley's article on the 1541 informative, especially after my unhappy experiences this past weekend.

What with Xmas presents and after-Xmas sales, I found myself with quite a few programs so I spent quite a bit of time on Sunday making backups for my own use. Eventually, as I was copying one of my disks, my 1541 went bonkers. At least the flickering red light reminded me of Christmas. A call to the Commodore service centre in Pennsylvania revealed that they would repair/replace the unit, which would no longer read files, for $85.00 (prepaid, of course). So, I went home at noon for lunch, found a box and prepared to part with my money and drive. I had not intended to ship it with the cardboard shipping piece in the drive, but I decided to do so. As I inserted it into the drive, I felt something move at the back of the machine. Since hope springs eternal and since I really wasn't looking forward to three more weeks of down time, I re-connected it and VOILA! It was working properly. Exactly what happened or why I don't know, but that little piece of cardboard saved me a lot of money and unhappiness.

John D. Baird, Norwalk, OH

Apparently fate was smiling in your direction that day. In general though, it never hurts to have a hardware PEEK. Sometimes pushing an IC firmly back in its socket is just enough to cheat fate and render a problem harmless. Besides, after 3 months you no longer need to be concerned about voiding the warranty. But even prior to the three month grace period, if your disk spends three weeks at the shop, that's three weeks when a real problem could arise which might otherwise occur three weeks after your warranty expires. I'm not sure if Commodore extends the warranty period if warranty service is performed, but who needs to be without their equipment when the problem is but a lamb in wolf's clothing. This and other self-service tips will be appearing in a future Transactor.

Compu-Artists Unite!: Hi! I am an artist working in the new medium of computer graphics and I have noticed something very interesting about computer-generated art that I want to tell you about.

I noticed that there are a lot of programs, drawing pads, printers, and so forth for the creation of computer art. Computer graphics is now an easy thing to get into, unlike 2 years ago when I first got started.

What is frustrating is that there are very few opportunities for computer artists to display their work. Most art magazines ignore computer art and fewer galleries have showings of it.

I think what needs to be done is for us to organize ourselves into a group, for the purpose of educating the public about what computer art can represent and bring pressure on art competitions to open categories for computer graphics. Also, galleries aren't going to pay attention to computer art until a group comes along and starts beating at the door.

So could you do me a favour and please print my name and address so I can get into contact with others working in this medium? Thank you.

George Bailey, 6474 Highway 11, Deleon Springs, FL, 32028

Perhaps you should contact Wayne Schmidt at 41 East 1st Street, Apt. 2W, New York, NY, 10003. Wayne has done several pieces for Ahoy! magazines and I'm sure there are several more that haven't yet been seen. I'm also sure Wayne would love to hear from you.

Point 11 AWOL: In Volume 5, Issue 06 (Programming Aids and Utilities), there is an article on aligning the 1541 disk drive. Point #11 is missing and nothing is said about tightening the second screw.

If there is something left out, I would appreciate knowing about it as we intend to do this from time to time.

Harvey B. Herman, Professor and Head, University of North Carolina

You're right, point number 11 got left out and should have stated the next logical step in the re-assembly process, tightening the remaining stepper motor screw.

Transbloopers

Dynamic Expression Evaluator: Volume 5, Issue 04

The DATA statements are fine, but there's a glitch in the BASIC loader portion. In line 155, "IF B♁0 . . . " should read "IF B♁≐0 . . . ", and the checksum in line 190 should be 10928 instead of 1330. We didn't catch this error until now because it seems the majority of you found it and corrected it yourself.

List Scroller: Volume 5, Issue 06, page 52

The first line of the BASIC Loader program should be line 10, not 0. This won't effect the operation in any way, but the "Verifizer" code will show as incorrect for the first line unless you make it a 10.

STP: Volume 5, Issue 06, page 55

The article stated that a version of STP that resides in the cassette buffer would appear on the Transactor disk for Volume 5 Issue 06, but alas, that version never made it to the disk. We'll try to squeeze it onto the next one. Also, the first line of the source listing should be line 1000, not 00.

Aligning the 1541: Volume 5, Issue 06, page 65

Point number 11 was left out; it should have said to tighten the remaining stepper motor screw. (See the Letters section.)

TransBASIC: Volume 5, Issue 06, page 20

The command 'CURSOR' and the function 'CLOC' were mentioned, but omitted from the program listings. They can be found in this issue's TransBASIC column.

The MANAGER Column

Don Bell
Scotland, Ont.

Letters to The Manager

Transferring or Archiving Records from One Disk to Another

Carrol W. Dauenhauer (Gretna, Lousisiana) wants to know if you can transfer records from one file to another and continue to dump records to that file from the orginal file.

If your original data file is getting almost full, you may want to store or 'archive' some records on another disk e.g. orders that have been completed or filled. Bear in mind that once separated from the orginal file, these records can only be manipulated and accessed through their own separate file. Also, as far as I know, you can only copy records one group at a time, i.e. you cannot keep dumping records, a few at a time to 1 archive file. Rather, the pattern will be to create several archive files each according to some criteria you set, disk 90% full, end of time period or say, reached a certain quota.

You can copy 'active' records from one file to a new file on another disk.

The best trick here is to set aside the first field in your record as a 'flag' for indicating whether a record is to be transferred or not. I would just make it a 1 character alphanumeric field in which you would place an 'a' for an active record and a 't' if you want it to be transferred.

(1) Choose the MANIPULATE FILES option then select 'Copy a Data File'. COPY ACTIVE RECORDS OR ENTIRE FILE? Choose 'A' (for active). This will copy all 'active' records not flagged as deleted, i.e. any record that has a blank or 't' in the first field, but not a '&'.

You now have a second file that has both active records and records to be transferred, but not 'inactive' or deleted records.

(2) Now you can delete the 'transferred' records in the original file as you now have a backup or archive copy of them. Do this by using the 'Global Change' function on field 1, changing 't' to '&'. This frees up space for more 'active' records in your original file.

(3) In order to purge your new archive file of 'active' records, you can also use the 'Global Change' function on field 1, only this time changing 'a' to '&'.

If you are conscious of saving disk space, you may want to go one step further and create a third file. You could make a copy of only the 'active' records in this file. This will leave you with a file containing only records that are completed or orders that have been filled with no blank records. You may then erase file two, the intermediate file.

All of this is useful for creating single 'archive' files in one shot. The problem is that you cannot keep dumping more records to this archive file. You will just keep making a bunch of little archive files every time you want more disk space. Within 'THE MANAGER' itself, there is no way of appending records from another file or concatenating files. If someone out there wants a good challenge, there's one to work on!

TransBASIC Installment #3

Nick Sullivan
Scarborough, Ont.

To this point, Nick has introduced the concept of TransBASIC (a method for building custom commands to be incorporated into BASIC) and the structure of a TransBASIC module was discussed in part 2. To take advantage of new TransBASIC command listings, one must first obtain a copy of the TransBASIC Kernel. The Kernel is only about 500 bytes long, but the source listing of the Kernel is quite long and can't be printed each time. Volume 5, Issue 05 (Hardware & Peripherals) contains the printed listing, however The Transactor Disk for every issue will include this file, plus files from the current and all previous TransBASIC articles.

Note: The CURSOR command and the CLOC function were described in part 2, but the source listings were omitted. They will be included here.

A TransBASIC ROMp

The routines listed here are useful in composing TransBASIC statements and functions; some are indispensable and are used time and again. Many other routines and bits of routines may be found in the BASIC and Kernal ROMs. Aids in discovering them include: Jim Butterfield's memory maps, Sheldon Leemon's "Mapping the Commodore 64" (COMPUTE! Books, 1984), a machine language monitor (like Supermon), and lots of time to browse.

CHRGET AND CHRGOT

These two routines are the means by which BASIC fetches bytes from the input stream, which can be either program text or a direct mode command line. A pointer at $7A/7B generally points to the byte most recently fetched. The byte is returned in the accumulator; .X and .Y are not affected. The carry flag will be returned set if and only if the byte is not an ASCII numeral. The zero flag will be returned set if and only if the byte is a statement terminator (0 or $3A, the colon).

CHRGET ($73) is visited en route to the execution routine for BASIC and TransBASIC statements; therefore such routines begin with the accumulator holding the first byte following the statement token. The same byte can subsequently be re-obtained, and the flags reset, by calling CHRGOT ($79). The above also applies to function execution routines in TransBASIC, but not in BASIC itself, which handles function calls in a slightly different manner.

TYPE AND SYNTAX CHECKING

The first two routines affect the status register only:

44429 $AD8D Ensure that last expression evaluated was of numeric type; or show a TYPE MISMATCH ERROR.

44431 $AD8F Ensure that last expression evaluated was of string type.

The next routines test the current byte in the input stream against a specified value. If the test fails a SYNTAX ERROR is shown. The routines exit via CHRGET, putting the next byte from the input stream in the accumulator.

44791 $AEF7 Test for right parenthesis.
44794 $AEFA Test for left parenthesis.
44797 $AEFD Test for comma.
44799 $AEFF Test for character in .A

EXPRESSION EVALUATION

Most of these routines expect the CHRGET pointer to be already pointing to the first byte of the expression to be evaluated. Only one, at $B79B, begins with a call to CHRGET to get the first byte of the expression on its own. After these routines the CHRGET pointer points to the first byte beyond the evaluated expression.

44446 $AD9E The workhorse of the evaluation routines. It decides whether the expression is of string or numeric type, then evaluates it and sets the expression–type flags in $0D and $0E appropriately. If the expression is of string type, a pointer is left at $64/$65 to the descriptor of the resultant string. If the expression is of numeric type the result is left in Floating Point Accumulator #1.

44426 $AD8A This routine calls $AD9E to evaluate the expression, then calls $AD8D to make sure that it was of numeric type.

44785 $AEF1 Check for opening parenthesis, evaluate the enclosed expression with $AD9E, then check for closing parenthesis.

44788 $AEF4 Same as $AEF1 but without the check for opening parenthesis. This can be used when the opening parenthesis is part of a function keyword.

47003 $B79B This routine calls CHRGET to get the first byte of a numeric expression between 0 and 255. The result is left in .X. An ILLEGAL QUANTITY ERROR is shown if the expression evaluates to an out of range result.

47006 $B79E The above routine should be entered here if CHRGET has already been called.

47083 $B7EB Get two parameters of the type used by the POKE statement. The first, an unsigned integer, is left in $14/15. The second is an integer between 0 and 255; it is left in .X. The parameters must be separated by a comma.

NUMERIC CONVERSION

47009 $B7A1 Convert the number in FPA #1 to an integer in .X.

45482 $B1AA Convert the number in FPA #1 to a signed integer in .A (high byte) and .Y (low byte).

45969 $B391 Convert the signed integer in .A (high byte) and .Y (low byte) to a floating point number in FPA #1.

47095 $B7F7 Convert the number in FPA #1 to an unsigned integer in .Y (low byte) and .A (high byte), and in $14/15.

45986 $B3A2 Convert the number in .Y to a floating point number in FPA #1.

STRING HANDLING

These routines can be used in various ways to develop statements and functions for manipulating strings.

46324 $B4F4 The bottom–of–string–memory pointer at $33/34 is moved down by the number of bytes specified by .A. A garbage collection is done if necessary. The new pointer value is also put into $35/36, and is returned in .X (low) and .Y (high). The original contents of .A is retained there.

46205 $B47D This routine calls the above one at $B4F4, but also saves the address of the reserved space at $62/63, and the number of reserved bytes at $61.

46282 $B4CA This routine converts the data left in $61/62/63 by the previous routine into a descriptor on the temporary descriptor stack. It leaves a pointer to the descriptor in $64/65.

46755 $B6A3 This routine checks that the most recent expression was of string type. Then it uses the pointer in $64/65 to create a pointer to the string data in $22/23. The length of the string is left in .A. The address in $22/23 is also contained in .X/.Y after the call.

46758 $B6A6 This is like $B6A3, but without the check for string type.

PRINTING CHARACTERS AND STRINGS

43847 $AB47 Print the character in .A to the current output device. This routine just calls the usual Kernal output routine ($FFD2) but includes i/o error–checking.

43735 $AAD7 Print a carriage return and, if the current output device is not the screen, a line feed as well.

43839 $AB3F Print a space.

43842 $AB42 Print a cursor right.

43845 $AB45 Print a question mark.

43806 $AB1E Print the string whose address is contained in .A (low byte) and .Y (high byte). The string must end in zero.

43809 $AB21 Set up a string using the routine at $B6A6 (see page 5), then print it.

43812 $AB24 The same, assuming the call to $B6A6 already to have been made.

ERROR MESSAGES

Jumping to any of the following routines will print the appropriate message, clear the stack, and restore direct mode. There are many others besides those given here. They may be easily found if needed. The same thing may be accomplished at the expense of two extra bytes by loading .X with the error message number and jumping through the BASIC error routine vector at $0300.

42037 $A435: OUT OF MEMORY ERROR
42353 $A571: STRING TOO LONG ERROR
44808 $AF08: SYNTAX ERROR
45640 $B248: ILLEGAL QUANTITY ERROR

TWO USEFUL TESTS

43052 $A82C Check STOP key, and execute STOP statement if it has been pressed.

45990 $B3A6 Show ILLEGAL DIRECT error if in direct mode.

Next issue, we'll try using some of these routines to create an actual TransBASIC command.

New Commands

This part of the TransBASIC column is devoted to describing the new commands that will be added each issue. The descriptions follow a standard format:

The first line gives the command keyword, the type (statement or function), and a three digit serial number.

The second line gives the line range allotted to the execution routine for the command.

The third line gives the module in which the command is included.

The fourth line (and the following lines, if necessary) demonstrate the command syntax.

The remaining lines describe the command.

CURSOR (Type: Statement Cat #: 004)
Line Range: 2574–2604
Module: CURSOR POSITION
Example: CURSOR 11
Example: CURSOR ROW,COL
Moves the cursor to specified row (0–24) and column (0–39). Column zero is assumed if no second parameter is present.

CLOC (Type: Function Cat #: 005)
Line Range: 2606–2618
Module: CURSOR POSITION
Example: IF PEEK(CLOC)<>32 GOTO 100
A quasi–variable that returns the actual memory location of the cursor.

COLSPR (Type: Statement Cat #: 031)
Line Range: 3530–3548
Module: SET SPRITES
Example: COLSPR 1,0: REM MAKE SPRITE #1 BLACK
The specified sprite (0 to 7) is set to the specified colour (0 to 255).

SSPR (Type: Statement Cat #: 032)
Line Range: 3550–3558
In Module(s): SET SPRITES
Example: SSPR 4
The specified sprite is switched on.

CSPR (Type: Statement Cat #: 033)
Line Range: 3560–3572
In Module(s): SET SPRITES
Example: CSPR 1
The specified sprite is switched off.

XSPR (Type: Statement Cat #: 034)
Line Range: 3574–3626
In Module(s): SET SPRITES
Example: XSPR 1,312
The specified sprite is positioned at the specified horizontal location (0 to 511).

YSPR (Type: Statement Cat #: 035)
Line Range: 3628–3654
In Module(s): SET SPRITES
Example: YSPR 5,59
The specified sprite is positioned at the specified vertical location (0 to 255).

XYSPR (Type: Statement Cat #: 036)
Line Range: 3656–3662
In Module(s): SET SPRITES
Example: XYSPR 0,H+1,V–1
The specified sprite is positioned at the specified co–ordinates.

WITHIN((Type: Function Cat #: 040)
Line Range: 3698–3784
In Module(s): WITHIN
Example: IF WITHIN(13;A,16) THEN PRINT "CLOSE ENOUGH!"
WITHIN(takes three numeric arguments. If the second argument is greater than the first, and the third greater than the second, it returns 'true' (–1); otherwise it returns 'false' (0). The comma used as a separator is equivalent to 'less than'; the semicolon is equivalent to 'less than or equal to'.

XLOC((Type: Function Cat #: 041)
Line Range: 3786–3810
In Module(s): READ SPRITES
Example: XSPR 3,XLOC(1)
Returns the horizontal position (0–511) of the specified sprite (0–7).

YLOC((Type: Function Cat #: 042)
Line Range: 3812–3828
In Module(s): READ SPRITES
Example: IF YLOC(6)>190 GOTO 200
Returns the vertical position (0–255) of the specified sprite.

Program 1

```
IL  0 rem cursor position (sept 4/84)   :
FH  1 :
JI  2 rem  1 statement,  1 function
HH  3 :
BE  4 rem keyword characters: 10
JH  5 :
NJ  6 rem keyword      routine    line    ser #
GD  7 rem s/cursor     csr        2574    004
LP  8 rem f/cloc       csrloc     2606    005
NH  9 :
ME  10 rem u/usfp (2620/006)
PH  11 :
KD  12 rem ================================
BI  13 :
DC  101 .asc "cursoR"
EI  600 .asc "cloC"
GI  1101 .word csr-1
MP  1600 .word csrloc-1
BE  2574 csr     jsr  $b79e    ;get first parameter
MG  2576         cpx  #$19     ;must be under 25
NL  2578         bcs  cs2      ;save on stack
IM  2580         txa
CJ  2582         pha
EM  2583         ldy  #0       ;assume column 0
CP  2584         jsr  $79      ;branch if no
EN  2585         beq  cs1      ; second parameter
LO  2586         cmp #","      ;has to be comma
PB  2587         bne  cs3
NN  2588         jsr  $b79b    ;get parameter
AH  2589         cpx  #$28     ;must be under 40
GD  2590         bcs  cs2
BD  2592         txa           ;move it to .y
BP  2594         tay
JK  2596 cs1     pla           ;recover row param
BP  2598         tax
NA  2599         clc           ;jump to kernal
BM  2600         jmp  $fff0    ; plot routine
KN  2602 cs2     jmp  $b248    ;illegal quantity
DD  2603 cs3     jmp  $af08    ;syntax error
CK  2604 ;
KH  2606 csrloc  lda  $d1      ;$d1 and $d2
ON  2608         clc           ; contain the
MM  2610         adc  $d3      ; start of row
FO  2612         tay           ; location. $d3
CO  2614         lda  $d2      ; contains the
GJ  2616         adc  #0       ; column.
AL  2618 ;
KK  2620 usfp    ldx  #0       ;convert .a (high)
FM  2622         stx  $0d      ; and .y (low)
DN  2624         sta  $62      ; from unsigned
PB  2626         sty  $63      ; integer to
JG  2628         ldx  #$90     ; floating point
FA  2630         sec           ; number in
FO  2632         jmp  $bc49    ; fac #1
AM  2634 ;
```

Program 2

```
LG  0 rem set sprites (aug 25/84)        :
FH  1 :
PH  2 rem 6 statements, 0 functions
HH  3 :
AF  4 rem keyword characters: 27
JH  5 :
NJ  6 rem keyword    routine    line    ser #
CB  7 rem s/colspr   colsp      3530    031
HE  8 rem s/sspr     ssp        3550    032
MC  9 rem s/cspr     csp        3560    033
HG  10 rem s/xspr    xsp        3574    034
PG  11 rem s/yspr    ysp        3628    035
OK  12 rem s/xyspr   xysp       3656    036
BI  13 :
IP  14 rem u/chkspr (3664/037)
HM  15 rem u/raschk (3676/038)
ON  16 rem d/powers (3694/039)
FI  17 :
AE  18 rem ================================
HI  19 :
PE  108 .asc " colspRsspRcspR "
JG  109 .asc " xspRyspRxyspR "
EH  1108 .word colsp-1,ssp-1,csp-1
ML  1109 .word xsp-1,ysp-1,xysp-1
BF  3530 colsp  jsr  chs1       ;get sprite number
AI  3532        txa
IN  3534        pha             ;save it
FN  3536        jsr  $b7f1      ;check comma and
NA  3538        pla             ; get colour
AO  3540        tay             ;sprite # is index
KI  3542        txa
BD  3544        sta  $d027,y    ;poke colour
GM  3546        rts
CF  3548 ;
OK  3550 ssp    jsr  chs1       ;get sprite number
HN  3552        lda  powers,x   ;set the bit
KO  3554        ora  $d015      ;or sprite enable
GP  3556        bne  csp1       ; rgstr, turn on
MF  3558 ;
IJ  3560 csp    jsr  chs1       ;get sprite number
BO  3562        lda  powers,x   ;set the bit
KP  3564        eor  #$ff       ;mask it out
BF  3566        and  $d015      ;and sprite enable
NL  3568 csp1   sta  $d015      ; rgstr, turn off
ON  3570        rts
KG  3572 ;
AN  3574 xsp    jsr  chs1       ;get sprite number
GK  3576        stx  t3         ;save it
JN  3578        jsr  $aefd      ;get comma
JB  3580        jsr  $ad8a      ;get x position
KI  3582        jsr  $b7f7      ;conv to integer
NA  3584        lda  $15        ;get high byte
HB  3586        cmp  #2         ;branch if
FD  3588        bcs  xs3        ; too high
KM  3590        ldx  t3         ;get sprite number
IB  3592        ror             ;put msb in carry
BA  3594        lda  powers,x   ;set the bit
CM  3596        bcc  xs1        ;branch on 0 msb
IK  3598        ora  $d010      ;or msb register
NC  3600        bcs  xs2        ;skip
MP  3602 xs1    eor  #$ff       ;mask the bit
AG  3604        and  $d010      ;clear the bit
DI  3606 xs2    tay  ;save msb
IP  3608        txa             ;sprite number
KO  3610        asl             ;double it
JK  3612        tax             ;use as index
AI  3614        lda  $14        ;get x low byte
KE  3616        jsr  raschk     ;wait for raster
GO  3618        sty  $d010      ;write msb
CJ  3620        sta  $d000,x    ;write low byte
CB  3622        rts
GA  3624 xs3    jmp  $b248      ;illegal quantity
AK  3626 ;
IA  3628 ysp    jsr  chs1       ;get sprite number
KD  3630 ys1    txa             ;double it
LN  3632        asl
OE  3634        pha             ;set it aside
LA  3636        jsr  $b7f1      ;comma, y-position
HE  3638        txa             ;move it to .y
HA  3640        tay
AA  3642        pla             ;get 2*(sprite #)
JM  3644        tax             ;use as index
HN  3646        tya             ;y-position
KG  3648        jsr  raschk     ;wait for raster
EN  3650        sta  $d001,x    ;write position
AD  3652        rts
ML  3654 ;
OF  3656 xysp   jsr  xsp        ;write x-position
OA  3658        ldx  t3         ;get sprite number
AI  3660        bpl  ys1        ;write y-position
EM  3662 ;
JO  3664 chksprjsr   $73        ;bump chrget ptr
CB  3666 chs1   jsr  $b79e      ;get sprite number
LB  3668        cpx  #8         ;must be under 8
GI  3670        bcs  xs3
EE  3672        rts
AN  3674 ;
HI  3676 raschkpha              ;store accumulator
IG  3678 ras1   lda  $d012      ;read raster line
BK  3680        sbc  $d001,x    ;subtract sprite-y
AF  3682        bcc  ras2
NH  3684        cmp  #$2b       ;wait till
ON  3686        bcc  ras1       ; clear of sprite
FN  3688 ras2   pla
GF  3690        rts
CO  3692 ;
FJ  3694 powers .byte 1,2,4,8,16,32,64,128
GO  3696 ;
```

Program 3

```
AE  0 rem within (aug 25/84)          :
FH  1 :
EC  2 rem 0 statements,  1 function
HH  3 :
HO  4 rem keyword characters: 7
JH  5 :
NJ  6 rem keyword    routine    line    ser #
GK  7 rem f/within(  within     3698    028
MH  8 :
DI  9 rem u/pshfp1 (3270/063)
HI  10 rem u/pul57 (3308/064)
PH  11 :
KD  12 rem ================================
BI  13 :
KA  607 .asc " within " : .byte $a8
GC  1607 .word within-1
BE  3270 pshfp1lda   #3         ;fac#1 to stack
```

```
JG  3272          jsr  $a3fb      ;check stack room
FC  3274          pla             ;save return addr
HJ  3276          sta  $71
GF  3278          pla
OJ  3280          sta  $72
NI  3282          jsr  $bbca      ;fac#1 to $57-$5b
GN  3284          ldx  #0         ;clear index
JI  3286 phf1     lda  $57,x      ;copy area
IM  3288          pha             ; to stack
GL  3290          inx
PD  3292          cpx  #5
GN  3294          bne  phf1
GM  3296 phf2     lda  $72        ;restore return
IB  3298          pha             ; address
BH  3300          lda  $71
CG  3302          pha
EN  3304          rts
AG  3306 ;
KG  3308 pul57    pla             ;$57. . . from stack
OH  3310          sta  $71        ;save return address
IH  3312          pla
AM  3314          sta  $72
KP  3316          ldx  #4         ;initialize index
OG  3318 pl57     pla             ;copy to area
PL  3320          sta  $57,x      ; starting at $57
BL  3322          dex
DA  3324          bpl  pl57
HN  3326          bmi  phf2       ;restore return address
GH  3328 ;
IL  3698 within   jsr  $ad8a      ;read num expr
IK  3700          jsr  comtst     ;sec if val2 within
EI  3702          php             ; bounds, and save
OP  3704          jsr  comtst     ;ditto
OB  3706          bcc  wth1       ;‘false’ on clear
JM  3708          plp             ;recover flag
BJ  3710          pha             ;dec stack ptr
EC  3712          bcc  wth1       ;‘false’ on clear
HN  3714          lda  #$ff       ;-1 = ‘true’
JJ  3716 .byte $2c                ;‘bit’ to hide lda
AB  3718 wth1     lda  #0         ;0 = ‘false’
IJ  3720          jsr  $bc3c      ;convert sign to fp
OM  3722          pla             ;inc stack ptr
NC  3724          jmp  $aef7      ;skip close bracket
EA  3726 ;
OC  3728 comtst   jsr  pshfp1     ;fac#1 to stack
ML  3730          jsr  $79        ;reread separator
NN  3732          pha             ; and save it
FF  3734          jsr  $73        ;bump chrget pointer
BL  3736          jsr  $ad8a      ;read num expr
JM  3738          pla             ;check separator
MO  3740          cmp  #","       ;comma (<) is ok
DL  3742          beq  ct1
KL  3744          cmp  #";"       ;semicolon (<=) is ok
EP  3746          bne  ct4        ;anything else is wrong
HO  3748          clc             ;clear for semicolon
LL  3750 .byte $24                ;‘bit’ to hide sec
KJ  3752 ct1      sec             ;set for comma
DK  3754          ror  t3         ;save flag as high bit
HD  3756          jsr  pul57      ;stack to $57 area
IA  3758          lda  #$57       ;compare area with fac
LP  3760          ldy  #0
OA  3762          jsr  $bc5b
CC  3764          bmi  ct3        ;val1>val2, false
JA  3766          bit  t3         ;comma if n flag set
JJ  3768          bpl  ct2        ; val1<=val2, true
FI  3770          tax
FD  3772          beq  ct3        ;val1=val2, false
PF  3774 ct2      sec             ;return true
MK  3776          rts
KI  3778 ct3      clc             ;return false
AL  3780          rts
FN  3782 ct4      jmp  $af08      ;syntax error
OD  3784 ;
```

Program 4

```
BO  0 rem read sprites (aug 25/84)       :
FH  1 :
DH  2 rem 0 statements, 2 functions
HH  3 :
BE  4 rem keyword characters: 10
JH  5 :
NJ  6 rem keyword     routine    line    ser #
IL  7 rem f/xloc(     xloc       3786    041
DK  8 rem f/yloc(     yloc       3812    042
NH  9 :
EP  10 rem u/chkspr (3664/037)
JN  11 rem d/powers (3694/039)
AI  12 :
LC  13 rem  this module also contains one
MO  14 rem  line from set sprites — 3624
DI  15 :
OD  16 rem ================================
FI  17 :
JA  608 .asc "xloc" : .byte $a8 : .asc "yloc" : .byte $a8
          ;a8 = shifted (
AF  1608 .word xloc-1,yloc-1
GA  3624 xs3    jmp  $b248       ;illegal quantity
JO  3664 chkspr jsr  $73         ;bump chrget ptr
CB  3666 chs1   jsr  $b79e       ;get sprite number
LB  3668        cpx  #8          ;must be under 8
GI  3670        bcs  xs3
EE  3672        rts
AN  3674 ;
FJ  3694 powers .byte 1,2,4,8,16,32,64,128
GO  3696 ;
ML  3786 xloc   jsr  chs1        ;get sprite number
HA  3788        jsr  $aef7       ;skip bracket
FC  3790        txa              ;double sprite #
LH  3792        asl
DG  3794        tay              ;use as index
DK  3796        lda  $d000,y     ;get x low byte
AO  3798        tay              ;
OF  3800        lda  powers,x    ;
OP  3802        and  $d010       ;get msb
PL  3804        beq  xl1         ;zero, msb clear
OK  3806        lda  #1          ;non-zero, mbs set
JJ  3808 xl1    jmp  $b391       ;.a/.y to fac#1
IF  3810 ;
IN  3812 yloc   jsr  chs1        ;get sprite number
BC  3814        jsr  $aef7       ;skip bracket
PD  3816        txa              ;double sprite #
FJ  3818        asl
LL  3820        tay
NA  3822        lda  $d001,y     ;get y position
PL  3824        tay
MA  3826        jmp  $b3a2       ;.y to fac#1
KG  3828 ;
```

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are two versions of VERIFIZER on this page; one is for the PET, the other for the VIC or 64. Enter the applicable program and RUN it. If you get the message, "***** data error *****", re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, enter NEW, then turn VERIFIZER on with:

```
   SYS 828  to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version       (turn it off with SYS 637)
```

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors (eg. POKE 52381,0 instead of POKE 53281,0), but ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

Listing 1a: VERIFIZER for C64 and VIC-20

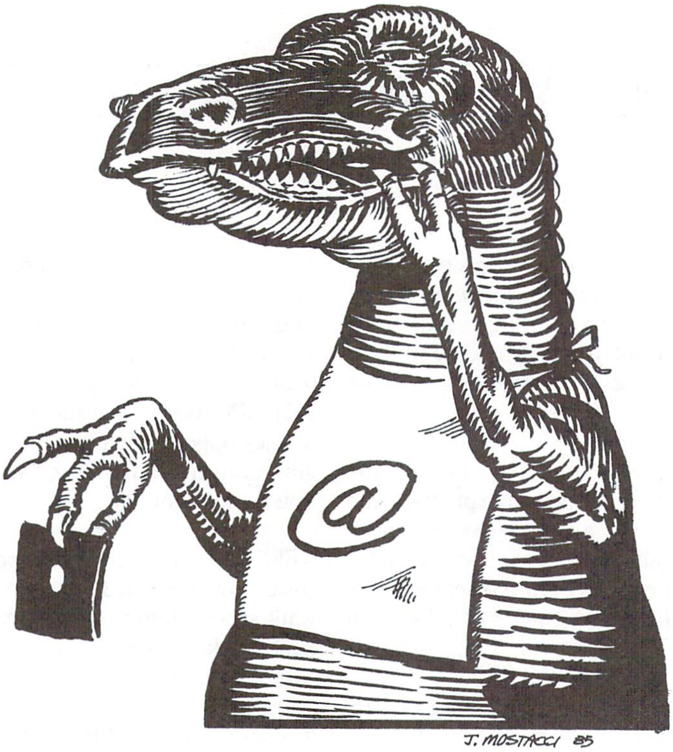
```
KE  10 rem* data loader for "verifizer" *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print "***** data error *****": end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI  10 rem* data loader for "verifizer 4.0" *
CF  15 rem pet version
LI  20 cs=0
HC  30 for i=634 to 754:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
OG  60 if cs<>15580 then print "***** data error *****": end
JO  70 rem sys 634
AF  80 end
IN  100 :
ON  1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB  1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK  1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB  1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE  1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI  1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB  1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA  1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE  1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL  1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA  1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI  1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB  1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM  1130 data 251, 133, 251,  96
```

SAVE with Replace Exposed!!

Charles H. Whittern
Hudson, MI



Or, SAVE @ Your Own Risk!

For years there has been unresolved controversy over the reliability of the Commodore "SAVE with Replace" (SAVE@) command. Warnings against its use have often appeared in print. Rumours first arose during the early days of the 4040 disk drive, and have continued with the 1541 drive. I became involved, not long after I got my 64, when one of my SpeedScript files was corrupted while I was using "SAVE with Replace". To be safe, I began using SCRATCH before each SAVE. Although this required extra time and effort, I felt much more secure.

A few months ago I became aware that two long–standing prizes were being offered to anyone who could show that SAVE @ was unreliable. One was a three–year old offer of a case of beer from Harry Broomhall of England. The other was a bottle of champagne from the Transactor magazine.

About the same time, I read a magazine column which emphatically proclaimed the soundness of "SAVE with Replace", and invoked the names of several well–known Commodore experts in its support. Since I had only experienced trouble once with "SAVE@", I decided to try using it again. I was putting together a disk of small to medium sized programs for our local computer club. Many of them needed some touching up, so I began my work, merrily using the convenient "SAVE@". Late in the afternoon I reLOADed one of these programs, but much to my surprise it wasn't there! It had been replaced by another program. On the directory everything looked fine. The program name and block count were O.K., but the program was gone!

I became somewhat miffed with the 'experts' opinions, and determined to unmask this program thief. The result is "SAVE@ EXPOSED!!!". I have tested it on three different 64s, with four different 1541s (including the new lever-operated model) and on my SX-64 portable. It has demonstrated the improper replacement of disk files on every system so far.

In order to meet the objections of the experts, before using "SAVE@", one must be sure there is space on the disk for the program to be SAVEd, that no improperly closed files have been SCRATCHed, and that the drive number has been specified as in SAVE " @0:PROGRAM NAME " ,8. My test disks all had over 300 blocks free. No improperly closed files had been SCRATCHed, and the correct syntax was used.

"SAVE@ EXPOSED!!!" selects randomly from five of the programs on the disk whose names have been placed in its data statements. Then using the 'Dynamic Keyboard Technique', it LOADs the selected program and immediately SAVEs it with replace. Another program is then LOADed and SAVEd with replace. The cycle continues until stopped by the operator. Sometimes a faulty replace occurs in just one cycle. Often it takes longer. The LOADing and SAVEing@ shows on the screen as the program RUNs. On paper, I make tally marks by each name as it comes on the screen for LOADing and SAVEing@. When each name has occurred at least five times I stop the program and examine the disk directory. Usually two or three of the programs will have been improperly replaced and one or two will have been lost.

Some Observations:

1. The block count is not always changed, but usually changes when the replacement file is larger, and often even when it is smaller. "SAVE@" is especially deceptive when the block count is not changed since there is no evidence of trouble on the disk directory. You are left to be rudely awakened later when you try to LOAD the victimized program.

2. A replacement program which has been linked to the wrong name is usually one which had been recently LOADed.

3. If "SAVE@ EXPOSED!!!" RUNs for a long time, more of the programs are replaced by the same one.

4. Disks which have not had any SCRATCHing done on them seem to be more immune to this bug.

5. Only programs that have been LOADed and SAVEd@ during the same session seem to be endangered.

6. Sometimes a program is mixed with parts of another rather than being completely replaced.

The debate is over. There is no longer any doubt about "SAVE@". It is unreliable. I suggest to the experts that they look again at "SAVE@" and some of the other related DOS routines for the bug that is causing trouble with the links used by "SAVE@". Good Hunting...

To Use "SAVE@ EXPOSED!!!" yourself, find or make up a test disk about half full of small to medium length programs, then do some SCRATCHing and SAVEing on it. Now type in "SAVE@ EXPOSED!!!" and SAVE a copy of it to another disk for safekeeping. Then LIST the program and change the DATA in line 240 to the names of any five programs on your test disk. Next, SAVE your version of "SAVE@ EXPOSED!!!" to your test disk. If you want to keep an uncorrupted backup copy of this test disk for study or comparison, make it now before RUNning "SAVE@ EXPOSED!!!".

SAVE @ EXPOSED!!!

```
HA  100 rem "save@ exposed!!!"
MB  110 rem by c.h.whittern,box 215,hudson,mich 49247
ON  120 cs$=chr$(147): qt$=chr$(34)
KO  130 d3$=chr$(17)+chr$(17)+chr$(17)
        : d4$=d3$+chr$(17)
HJ  140 for i=1 to 5: read a$(i): next
IL  150 i=int(rnd(0)*5)+1
JD  160 print cs$"load"qt$;a$(i);qt$",8"
NH  170 print d4$"save"qt$"@0:"a$(i);qt$",8"
KA  180 j=int(rnd(0)*5)+1: if i=j then 180
NN  190 print d3$"load"qt$;a$(j);qt$",8"
OJ  200 print d4$"save"qt$"@0:"a$(j);qt$",8"
GO  210 print d3$"load"qt$"save@ exposed!!!"qt$",8
IG  220 poke 631,19: for i=1 to 5: poke 631+i,13: next
OO  230 poke 637,82: poke 638,117: poke 639,13
        : poke 198,9: end
JE  240 data recover ram,check disk drive,quadra,
        performance test,disk log
```

Get a scratch pad and write down the names of the programs you placed into the data statements, along with their block counts. LOAD and RUN "save@ exposed". Each time a program is LOADed and SAVEd with replace, make a tally by its name. When each of the five programs has been LOADed and SAVEd with replace at least five times, stop the computer with the RUN-STOP key while it is LOADing a program.

LOAD and LIST the directory. Check the block counts first. If they are wrong, you can be sure the program has been replaced by another. Then LOAD and LIST each of the five programs to discover exactly what mischief has been done!

Editors Note:

Although many will scoff at the statements made above, they have been tested and found to be true, at least on the 1541 and dual 4040 drives. We didn't have a 2031 but it's hard to exclude it, being so closely related. Oddly enough it didn't upset the 8050 or 8250 at all. The original DOS source listings for the 8050 have comments that lead one to believe that the engineers did indeed look for a problem with SAVE @, but apparently nothing was changed.

For our test purposes, 10 files were SAVEd to diskette, labelled PRG #1 through PRG #10. Initially, each had a file size of 11 blocks, and each was written to be easily identified. The program "SAVE@ EXPOSED!!!" was then modified to include the 10 filenames, and this was SAVEd to diskette. From here, the program was fired up and allowed to RUN for about ten minutes.

After ten minutes, BASIC AID was LOADed in, and with it each file was FLISTed (LISTed directly from disk) to the screen. One file, PRG #2, was found to be posing as PRG #4. By FLISTing PRG #4, it was found to be OK. Although PRG #2 still had the same filename, it was now a true clone with no further semblance of its original identity.

Though this was a form of proof, we decided it was time to get really mean. Without belabouring the point, some of the files were increased in size, then SAVEd @ back to disk. This insured that a good assortment of block counts were showing in the directory, and the sector distribution of the files would be good and mixed up.

After 30 minutes of operation, the program was stopped and the results were tabulated. YECHH! Of the 10 files, 5 were corrupted badly. PRG #1 became PRG #4. PRG #2 became PRG #7. PRG #4 became PRG #5. PRG #7 became PRG #9. And PRG #9 became PRG #1. Of these, only PRG #4 had its directory links mixed up with another, PRG #5. The others, though still posing under the same file, were true clones. It seems we have a problem.

At this time, one bottle of champagne should be winging its way to a certain mail box in Hudson, Michigan. We'll be informing Mr. Broomhall about this development, but wether his criteria have been met is not known. Our criteria was simply to prove a problem exists and that has obviously been shown. Although the program offered here demonstrates this quite clearly, the sequence of events leading up to its often unexpected occurence in day to day activity is still a mystery. And why does SAVE with replace seem to work on most occasions? A look through DOS ROM has shown that SAVE with replace is merely performing a SCRATCH after a SAVE followed by some very simple directory updates. Nevertheless, there is no longer any doubt about the potential danger which means there must be something going unnoticed. We'll be taking another very careful examination of the code but we don't underestimate this bug, probably the most elusive of them all. Perhaps those others who so adamantly claim the bug is a myth might care to join in our search for the answer to the next logical question: Why? (and that includes you, Commodore).

Disk Tricks

Scott MacLean
Georgetown, ONT

The 1541 disk drive, contrary to popular belief, is a very intelligent disk drive. It can be told to go off and do something on its own, and it will whirr happily away to itself, without any intervention from the computer. This is contrary to an APPLE disk drive, where the computer is tied up when the disk is in use. This is because the 1541 is actually a computer, almost as smart as your 64! It has a 6502 CPU in it, I/O chips, ROM, and RAM, not to mention interfacing circuitry for the motors and read/write head. There is a way to directly write and read to and from the 1541's memory. This can be useful in controlling the disk drive without intervention from the 1541's parser, which is sort of like a syntax error checker. The way a user accesses this memory is through 2 commands: Memory Write (abbreviated M–W) and Memory Read (abbreviated M–R). The syntax for these commands is as below:

```
print#15, " m–w " ;chr$(memlo);chr$(memhi);chr$(len);chr$(data);chr$(data)
```

memlo, memhi: location in the disk drive memory to do the write, in lo–hi byte format
len: number of bytes in data
data: series of bytes to be loaded in starting at memlo,memhi.

```
print#15, " m–r " ;chr$(memlo);chr$(memhi);chr$(len)
```

You may send as many as 34 bytes of data at a time to the disk drive's memory. Only one byte may be received from the memory at a time. The memory location we will be concerned with today is $1C00, or 7168 decimal. This location is one of the main control locations in the 1541. Its layout is as follows:

Bit Value Use For $1C00

7	128	* See below
6	64	Density: high bit
5	32	Density: low bit
4	16	* See below
3	8	Red drive light
2	4	Drive motor control
1	2	Stepper motor: high bit
0	1	Stepper motor: low bit

* Bits marked with a * won't be described here.

6–5. Record/playback density:

The tracks on a diskette are concentric, thereby making the inner tracks smaller than the outer tracks. To tell the disk drive, this location is adjusted when a track and sector is selected. It is adjusted as follows:

Track	5	6
1–17	1	1
18–24	1	0
25–30	0	1
31–35	0	0

3. Red drive light:

This bit controls whether the red light on the front of your drive is on or off. You can M–W this to turn on, but the 1541's interrupts will turn it back off, producing only a brief flicker.

2. Drive motor control:

This bit turns the drive motor on and off. This location is not touched by the drive's interrupts, so you are free to turn it on and off at will. In order to move the head, a safety interlock ensures that the drive motor is enabled before, or else the head movement instruction is ignored.

1–0. Stepper motor control:

The stepper motor is the motor which moves the head back and forth across the disk. It will remain inactive unless the drive motor is enabled (see bit 2). The stepper motor consists of 4 internal coils, and a magnet mounted on a shaft. When the coils are sequenced on and off around in a circle, it drags the magnet arm around, which produces a very high accuracy movement. The movement is much too accurate to be used by the drive, so two movements is considered one track on the disk. Half tracking is achieved by moving only one movement, and writing in between tracks where most copiers don't look. Commodore could have doubled the 1541's capacity with this feature. . .but wait!! Before you rush off to double all your disks capacity, listen: The read write head is too wide. If you write to track 4.5, you will destroy track 4 and track 5. Similarly, if you write to track 7, you would destroy track 6.5 and 7.5. This motor is useful for other tricks, however. Try typing in the following program before proceeding furthur.

```
10 open 1,8,15
20 x=0
30 if peek(197)=7 then x=x-1
40 if peek(197)=2 then x=x+1
50 if x=4 then x=0
60 if x=-1 then x=3
70 print#1,"m-w";chr$(0)chr$(28)chr$(1)chr$(4+96+x)
80 goto 30
```

You may find it helpful to remove your drive cover before running this program, in case you make a mistake, in fact, I would recommend removing it so that you may get a better idea of what is going on *(Editor's Note: This will void your warranty!)*. This is done by removing the 4 screws in the bottom of the drive, carefully flipping the drive over, and removing the cover. Then run the program. Press and hold down the CRSR UP/DOWN key for about a second and watch the read write head. Press and hold down the CRSR LEFT/RIGHT key for about a second and watch. Try holding the UP/DOWN key until the head clicks. This is what happens when you initialize a disk. Hold down the LEFT/RIGHT key until the head clicks. Release it IMMEDIATELY!! This is track 37. Return the head to about center with the UP/DOWN key. If the head won't move, give it a gentle push with your finger.

You may have a disk that you borrowed from your friend that doesn't "agree" with your 1541. If the red light flashes rapidly while it loads, try the following program:

```
10 open 1,8,15
20 input"<i>n or <o>ut of alignment";io$
30 print#1,"i";"m-r"chr$(0)chr$(28): get#1,a$
    : a=asc(a$+chr$(0))
40 v=(aand254): if io$="o" then v=(aor1)
50 print#1,"m-w";chr$(0)chr$(28)chr$(1)chr$(v)
60 close1: end
```

This program moves your head up 1/2 track and ends, or if you select <I>n alignment, it resets the head to normal.

If at any time you find you have trouble reading, writing, or accessing your drive, one of the following should have an effect:

```
load"#",8
open 1,8,15,"i0": close1
```

Or use the program above and select <I>n alignment.

The 1541 disk drive is full of goodies and tricks, only one of which we have tapped here. One note is the fact that if you move the head manually as we have here, the 1541's DOS (Disk Operating System) is unaware that you did so, and will assume that it is still where it started out. If it is at track 3, and you manually move it to track 30, then try to LOAD"$",8, it will try to move to track 18, and the head will jam, requiring you to open the disk up and reset the head. The safest way to exit is to execute the following line several times:

```
open 1,8,15,"i0": close1
```

Editor's Note

Now that DOS Memory maps have been published for the 4040, 1541, and 8050 (inside The Complete Commodore Inner Space Anthology, see back cover) it will be somewhat easier for us to examine other "disk tricks". Watch for more articles like this one in future Transactors.

DiskBusters!

Michael Quigley
Vancouver, BC

Reviews of some current disk copying packages

Although The Transactor does not agree with the idea of "copy protection unlockers", we can not ignore their existence either. Had Michael chosen to review a single package we would not have accepted it, however the comparison that follows is in the true Transactor tradition and we hope that any readers who might take offense to the undue publicity will also consider the fact that the information here will help the decided avoid yet another less than satisfactory package. – M.Ed.

One of the biggest rackets facing Commodore 64 owners is an ever-increasing number of "disk copy" programs. Every month's issue of major magazines features large ads for utilities with claims like "Backs up virtually all existing disks for Commodore 64 including Copy Protected Software", "The ultimate bit by bit disk duplicator", "No better disk copier at any price", and "Fastest and most advanced copier you can buy."

In reality, these programs are part of a vicious circle highly reminiscent of Biblical "begats". As soon as one method of breaking copy protection is introduced, a new protection system is quickly developed by the software houses, which brings another generation of pirate programs, and on and on. The last year has seen, in many commercial programs, the demise of the familiar errors which cause the 1541 to perform its unpleasant knock–knock noise as a method of disk protection. In their place have come half–tracking (moving the read head to a space between the extant 35 tracks), writing beyond track 35 (up to track 44), varying the number of sectors in a manner inconsistent with normal DOS functions, and the use of fast–load techniques.

Many of the pirate pack programs try to justify their existence by claiming that a person has a right to back up their software, which I agree with. A person should also be able to modify their software to their own purposes, especially if it doesn't meet their expectations.

However, some of these breaking packages are less than subtle about their real intentions. One of them, The Software Protection Handbook, was originally to be called "The Software Pirate's Handbook II". The Authors try to justify this name by saying that the word "pirate" in the title was "intended as a light–hearted reference to any copying process, and to inspire a certain tendency [sic] of humankind; the attraction to things mysterious or secret."

What follows are reviews of a representative sample of disk–buster type programs. Not surprisingly, several other companies which I contacted refused to send me their products.

DI–SECTOR
Starpoint Software, Star Route, Gazelle, CA 96034. $39.95

Di–Sector is a slickly designed program which is relatively easy to use. It comes in the form of a master disk from which you are allowed to make three copies. Each of these is encoded with your name and serial number. It features a 3–minute copy program, a quick format (around 16 seconds), a public domain–style disk doctor and a machine language monitor with typical commands which additionally allows you to transfer code to and from the 1541's memory.

Di–Sector also contains a bit copier, a file copier (which will read the file names from "invisible" directories) and an error maker/checker. There is a Sector Editor, sort of like a Disk Doctor, which I found confusing because some characters in the sector did not appear on the screen. An "Arts Backup" creates unprotect backups of Electronic Arts disks.

About the only negative feature of Di–Sector is that you're not allowed to return to the main menu from half of the program's six sub–sections or exit the program without turning off the computer. The company's ads are also in error when they claim that "None of our copy routines ever make [sic] the drive head 'kick'." Formatting a disk not only kicks the head, but does so several times faster than normal.

Most parts of the program load in very quickly using special DOS techniques, and devices like printers should not be connected to the serial port. (Try disconnecting your printer after booting De–Sector for some unusual and very harmful noises.)

The manual with Di–Sector is not bad, though there are names referring to various parts of the program which do not tally with the names in the main menu.

Di–Sector will not copy itself, nor will it copy recent programs which contain methods of protection other than errors. As such, its uses may be somewhat limited to legitimate purposes like copying and performing various housekeeping tasks on your own disks, which it does extremely well.

MASTER COPY
Digital Wizardry, 3662A South 15th Street, Milwaukee, WI 53221. $19.95

This is a home–grown kind of production which claims to be "the most effective, yet still the most inexpensive copy utility. . .for. . .the 64." This claim is debatable, since virtually everything available in it is available in a public domain equivalent.

The program is divided into 7 sections. One of these is a Disk Doctor which allows you to copy a block from one disk to another. It also allows you to scan back and forth within tracks — that is, it will jump from sector 1 to 2 to 3 and so on. Another section catalogs a disk, which just means reading the directory. And another copies a disk with a variant on the familiar 4–minute backup program, complete with head knocks at uncalled–for locations. If you want to format a disk, this can be done in 21 seconds. Errors 20, 21, 22, 23, 27, 29 can be located and created.

Probably the least satisfactory part of the program is the one which copies sequential and program files. After you insert the disk you want to copy from, a prompt — "OUTPUT *" — appears on screen. There is nothing to tell you, without looking in the manual, that you are supposed to input either D for Disk or T for Tape (the latter an unusual touch) at this point. You can copy a total of about 110 blocks or 27K at a time (some public domain programs allow up to 51K), and once you have copied certain files, you have to scan through those file names in the directory on your subsequent passes through it before copying new ones. You cannot exit from this part of the program to the main menu.

Virtually all these utilities are available for nothing from user group libraries. About the only thing I found interesting about Master Copy was its method of protection which involved numerous errors and "secret passwords". As you might have guessed, you can't copy the disk with itself.

PROGRAM PROTECTION MANUAL FOR THE C–64
C.S.M. Software, P.O. Box 563, Crown Point, IN 46307.
$29.95 plus $2 shipping.

I like this book, because it seems, unlike most computer literature, to be written by an intelligent person. This is not to say that it's free of grammatical errors or published in a slick format. It outlines methods of defeating various methods of protection and also tackles the ever–changing area of software law (U.S. variety, of course).

The book is written in a clear, concise and easy–to–follow manner. I had little trouble employing some of its methods to change several older commercial programs so their errors would not be detected.

The book comes with a disk of "public domain" software, including the Disk Doctor written by Canadian Don Lekei. The book's author, one T.N. Simstad, gets himself so entangled in various statements of liability that he describes this disk as "copyrighted". The disk, by the way, contains various features such as an invisible directory and other little challenges, all of which can be explored with methods described in the book.

Among these public domain programs are an early version of the 4–minute copy program (that sure gets around, does't it?) and another to determine if there are any errors on disks, both of which cause the 1541 to do its knock–knock routine, a major cause of drive failure. Is it no coincidence that C.S.M. Software also sells a 1541 disk alignment program for $39.95?

C.S.M. also sells expansion boards to aid in cracking cartridges and publishes a monthly Program Protection Newsletter for $35.00 a year, which details in each issue how to break 4 or 5 programs.

Unfortunately, the kind of approach exemplified by this company is all too susceptible to the "vicious circle". Much of its information is already obsolete, though it may be of interest to people planning to protect their own programs. Or perhaps it will just discourage them from even bothering to write any.

SUPER CLONE, also known as THE CLONE MACHINE
Micro–W Distributing Inc., P.O. Box 113,
Pompton Plains, N.J. 07444. $49.95

This program, issued in a revised version in September 1984, was one of the earliest copy utilities. It consists of three major sections. One of these is a further variant on the 4–minute backup without the head–knocking at the beginning of a transfer. Another is something called Tough Nuts Utility, which allows you to break complicated new methods of protection like those varying the number of sectors per track in a manner inconsistent with DOS. In order to find out about Tough Nuts, however, you have to subscribe to a newsletter from Micro–W devoted to these methods, at an additional cost.

The major part of the program consists of Super Clone and the original Clone Machine, the latter being a slower version of the former. Super Clone does have one good feature – a bit copier which, like most, takes an eternity, but it copies most normally–created disks, warts (errors) and all.

Parts of the program leave a lot be be desired. The file copier, for example, lets you choose several files and then proceeds to copy them – one at a time! Although the program is supposed to be "self–documenting," there are sections which utilize the function keys for various purposes. There are no prompts for these on screen – you have to look them up in the manual.

The manual itself is not bad, but suffers from a certain kind of disorganization caused by the complexity of the programs themselves. The back of the manual contains several pages devoted to errors which may occur!

The Software Protection Handbook
PSIDAC, 7326 N. Atlantic, Portland, Oregon 97217. $19.95
disk of programs $16.95, or both for $29.95

According to its ads, this book will help you "blow the locks off protected software!". At the same time, it claims it "does not condone piracy." Sure, sure. . . On its second last page is an ad for "Software Pirates' T–shirt White Skull and Crossbones on jet–black Shirt."

Co–authored by David Thom and Vic Numbers (yes, that's correct), the book, for the most part, lives up to the first claim above. Methods on how to break into disk, tape and cartridge programs are all covered. The last two can be done with the help of hardware which PSIDAC will be glad to sell you at additional cost. There are numerous programs listed throughout the book for examining disks, checking for and creating errors, duplicating disks and individual files, creating auto–boots, and so forth. All these programs can be purchased on disk from PSIDAC, again for more money. (I was not supplied with the disk).

Probably the most interesting part of the book is its first chapter, which examines the legal aspects of copying software, and in doing so, gives new dimensions to the phrase "rambling discourse." It does have a few points which I agree with, such as the fact that most software is grossly overpriced. However, what is one to think of opinions like this when placed in the context of fuzzy thinking like the following passage: "Copying for sale, distribution or other non–personal uses is Piracy. . . . Loaning your original to another person for temporary use is not piracy. . . . However, copying an original you do not own is unethical."

The book gives the appearance of being well–made with a plastic spiral–type binding and glossy cover. The typography inside leaves a great deal to be desired, however, using an IBM style of typewriter and listings from what seems to be a Commodore printer. The book is full of unbearable illiteracies on practically every page, ranging from three–letter words like "its" on up.

DOS File Executor

Program and concept by Chris Johnsen, Clearbrook, BC

Execute Machine Language programs inside your 1541

Automatic Disk Drive Load and Execute

There's a useful feature hidden in the 1541's ROM that Commodore never told anyone about. Now that more disk drive mysteries are being revealed with books like "Inside Commodore DOS" from Datamost Inc., and Transactor's "Complete Commodore Inner Space Anthology", we're learning how to get the most out of the drive's undocumented features.

What we're talking about here is the automatic load and execute feature, accessed by simply sending the filename of a specially formatted USR file to the drive's command channel. The filename must be prefixed with the characters "&:" (ampersand, colon), both in the disk directory and the command sent to the drive. On receiving the magic word, the drive will load the machine code contained in the file into the specified RAM address and execute it. This feature (UTLODR — residing at $E7FF to $E852 in the ROM) was probably put there for drive ROM development purposes, to quickly boot up a new DOS version. In this article it will be referred to as the "DOS exec" routine, and the files it uses as "DOS exec files". The DOS exec capability apparently exists on 8050/8250 and new 2031 drives as well as the 1541.

This article explains the format of DOS exec files, and presents the program in Listing 1, "DOS exec filer" for the C64, which creates a DOS exec file from a machine code object file. We'll also be using the DOS exec technique to implement two short drive-executable routines: (1) to change the disk drive's device number to 9, and (2) to disable the horrid head-knock which occurs while LOADing some commercial protected software. These routines may be created from object form by "DOS exec filer", or created directly from the short BASIC programs in Listings 4 and 5.

Since the 1541 contains a 6500-family CPU and the DOS allows for execution of user routines, the drive can be programmed using 6502 machine code. The traditional way to execute routines within the drive is to use the memory-write (M-W) and memory-execute (M-E) commands or the block-execute (B-E) command, all of which are documented in the drive manual. Programming the drive using these commands usually requires a program to be run on the host computer, which means loading the program and disturbing any current program in memory. With the block-execute command, you have to know exactly which track and sector on the disk contains the machine code to be executed — not very convenient. Using the DOS exec method only involves sending a single command to the disk drive, which then gets all its instructions from a special DOS exec file on disk.

When writing programs to be executed by the disk drive, there are a few things to keep in mind. The main constraints are the lack of RAM available and the volatility of that RAM. Fortunately, we can still use some pretty useful routines, since simply changing the contents of certain RAM locations can go a long way towards customizing the behaviour of the DOS. Listing 2 shows the machine language routine which changes the drive's device number to 9, and Listing 3 is the anti-knock remedy. These tiny weapons need only be executed once to have their intended effect, and are perfect candidates for DOS exec files.

Remember, once a DOS exec file is on disk, you just have to send the name of that file to the drive command channel, and it gets loaded into drive memory and executed. As an example, if you create the "&:dev 9" file using the program in Listing 4, you'll be able to change your drive's device number just by entering:

```
                open 15,8,15: print#15, "&:dev 9"
    or simply  "@&:dev 9"  using the DOS wedge
```

You won't have to remember any special numbers or commands, and you won't have to LOAD anything into your computer to disturb it. After realizing the tremendous power of this capability you'll probably want to create your own DOS exec files. "DOS exec filer" in Listing 1 (explained later) will create DOS exec files for you, but an explanation of such a file's format follows to appease your curiosity.

First of all, a DOS exec file is a USR file. All that means is that the format is user–defined and the file appears in the directory with USR next to it instead of SEQ or PRG.

Assuming that you know about tracks and sectors, let's get right into the format of a DOS exec file sector; a diagram appears below for clarification. The first two bytes, 0 and 1, are set automatically by DOS and point to the next track and sector (if any) as usual. After that we're on our own. Bytes 2 and 3 must be set to the address in disk RAM — in low, high format — where the user machine language routine will load and execute. Byte 4 contains the length of the machine language program on the current sector. The machine language program follows in the subsequent bytes on the sector, not going further than byte 254. After the ML program, there is a checksum byte, which would occupy byte 255 if the ML program filled the sector. The checksum byte must be supplied by the user, and represents the sum of bytes 2 through the end of the program on the sector. The low and high bytes of this sum are added to yield the checksum byte. This diagram should clarify the format.

When OPENing the DOS exec file to initially create it, you have to use the drive number in the filename to ensure that the ampersand and colon get included. A valid DOS exec file OPEN would be:

```
          open 8,8,12, "0:&:filename,u,w"
```

As you can see, creating a DOS exec file involves quite a few steps: writing the load/execute address, the program length, copying the program itself, and calculating and writing the checksum at the end. And if the program is longer than 250 bytes, all this must be done for each subsequent sector.

You can save yourself all this work by using the BASIC program in Listing 1, "DOS exec filer". With DOS exec filer, just write the machine language program for the drive to execute and put the object file on disk (the origin address is unimportant). DOS exec filer will ask the filename of the program and the start address in drive memory (after printing a table of available addresses), and create a DOS–executable USR file for you with a filename that you specify. Simple. After that, whenever you have the exec file in the drive, you've got instant access to your favorite drive utility.

The examples used here, "&:dev 9" and "&:anti–knock", are a good start to your disk drive exec file collection. Dev 9 comes in handy when you wish to access two disk drives, and anti–knock can keep the poor 1541's head from rattling itself out of alignment by hitting read errors. If you want either of these files, you can create them without typing in DOS exec filer — just use Listings 3 or 4 to create them directly.

Now that this obscure disk drive feature has been publicized, we hope to generate a flurry of activity out there in hackerland. If anyone comes up with a good DOS exec file for the 1541, 8050, 8250 or new 2031, send it in for the bits & pieces column. Maybe we can work the drive as hard as the computer for a change!

0 1	2 3	4	5...............................	end+1
Track, Sector Links	Low, High Address	Length	machine language routine	checksum

Listing 1: DOS exec filer for the C64

```
KG | 10 rem * 1541 dos execute filer :
   |    – print#15, "&:dos utility"
II | 20 :
ML | 30 rem ***** set up screen *****
MF | 40 gosub5010:printgr$:poke53280,11:poke53281,0
   |    :gosub2020
GK | 50 :
NB | 60 rem ***** m.l. program read *****
FK | 70 open15,8,15:input#15,er$,em$,et$,es$
   |    :print#15, "u;":printd$
NK | 80 input"  name of program file ]]]]]]]]] ";p$
GJ | 90 ifp$= " "thenprintu$u$u$:goto80
HE | 100 iflen(p$)>16thenprint"  not more than 16
   |      characters":printu$u$u$u$:goto80
CJ | 110 pn$=a$+p$+b$
KA | 120 print#15, "i0:":open1,8,3,pn$:close1:printu$;
   |      :gosub3010:open1,8,3,pn$
DH | 130 get#1,ip$:ip$=ip$+z$:lo=asc(ip$):get#1,ip$
   |      :ip$=ip$+z$:hi=asc(ip$)
MD | 140 printd$"  finding length of m.l. routine"
JI | 150 ra=hi*256+lo
JN | 160 printd$"  computer ram address  ";ra
IB | 170 printd$"  reading byte #  "
CF | 180 nu=nu+1:get#1,ip$:printtab(16)u$;nu:
   |      en=st and64:re=st and2
JD | 190 ifenthen210
EF | 200 ifnotrethen180
EA | 210 close1:gosub3010
JC | 220 printu$"   "q$p$q$"file contains";
   |      nu;l$;"  bytes  ";d$
PD | 230 printd$d$"  press shift to continue":wait653,1
   |      :gosub2020
```

```
EG  240 :
CC  250 rem**** write usr execute file ****
ML  260 print:input" name of execute file ]]]]]]]] ";e$
PO  270 ife$ = " "thenprintu$u$u$:goto260
OK  280 iflen(e$)>14thenprint" not more than 14
          characters":printu$u$u$u$:goto260
DO  290 en$ = a$ + m$ + e$ + ur$:open1,8,3,en$:close1
          :w = 9:gosub3010
DP  300 ifer$ = "00"thenprint" sorry that file
          exists":printu$u$u$u$:w = 0:goto260
HK  310 ifer$<>"62"thengosub3040
GD  320 print" dos ram address (768 - 2048)
JI  330 printd$" buffer 0   - 768    $0300"
EI  340 print" buffer 1   - 1024   $0400"
EJ  350 print" buffer 2   - 1280   $0500"
NK  360 print" buffer 3   - 1536   $0600"
IB  370 print" bam buffer - 1792   $0700"d$
OL  380 input" decimal address of exec file ]]]]] ";dm$
LE  390 ifval(dm$)<768thenprintd$d$" not lower
          than 768 ";printu$u$u$u$u$:goto380
ED  400 ifval(dm$)>2048thenprintd$d$" not higher
          than 2048":printu$u$u$u$u$:goto380
NI  405 en$ = a$ + an$ + e$ + c$
ME  410 open1,8,3,pn$
BF  420 open2,8,4,en$
BJ  430 dm = val(dm$):hi = int(dm/256):lo = dm-hi*256
OM  440 bl = int(nu/250):ef = nu-250*bl
HP  450 get#1,ip$:get#1,ip$:printd$d$" writing
          byte # ":ifbl = 0then510
BA  460 fort = 1tobl
GP  470 print#2,chr$(lo);:ip = lo:gosub1020:print#2,chr$(hi);
          :ip = hi:gosub1020
HB  480 print#2,chr$(250);:ip = 250:gosub1020
AA  490 forz = 1to250:gosub4010:nextz
AC  500 print#2,chr$(ck);:ck = 0:nextt
OB  510 print#2,chr$(lo);:ip = lo:gosub1020:print#2,chr$(hi);
          :ip = hi:gosub1020
BM  520 print#2,chr$(ef);:ip = ef:gosub1020
LC  530 forz = 1toef:gosub4010:nextz:print#2,chr$(ck);
DK  540 close1:close2:gosub3010
OP  550 gosub5070:printgr$:gosub3010
FP  560 printd$d$" usr file complete"
KJ  570 print:input" compile another file ]]]] ";v$
KO  580 ifleft$(v$,1) = "y"thenclose15:run
JL  590 ifleft$(v$,1) = "n"thenprint#15,"u;":close15
          :printcl$:end
MD  600 printu$u$u$:goto570
GN  610 :
DM  620 — subroutines:
KO  630 :
AA  1000 rem ***** checksum of block *****
GC  1020 ck = ck + ip:ifck>255thenck = ck-255
CC  1030 return
EI  1040 :
HB  2000 rem ***** title *****
AA  2020 printcl$d$d$"1541 dos  execute filer"
LK  2030 print"        "
EH  2040 printdn$"print#15,"q$"&:dos exec"q$
OB  2050 return
AI  2060 :
DJ  3000 rem *** check error channel ***
LB  3010 input#15,er$,em$,et$,es$
IE  3020 ifer$ = "00"thenreturn
KO  3030 ifw = 9thenreturn
AA  3040 print#15,"i0:"
BC  3050 printd$" error "er$" "em$" t "et$" s "es$
FI  3060 printd$d$d$:close15:end
CH  3070 :
CM  4000 rem**** read byte/write byte ****
OE  4010 get#1,ip$:ip = asc(ip$ + z$):gosub1020
GO  4020 by = by + 1:printtab(16)u$;by:print#2,chr$(ip);:return
CD  4030 :
IK  5000 rem  **** declarations ****
JG  5010 u$ = chr$(145):l$ = chr$(157):q$ = chr$(34)
HP  5015 cl$ = chr$(147):d$ = chr$(17):gr$ = chr$(30)
DG  5020 z$ = chr$(0):p$ = "":pn$ = "":e$ = "":en$ = ""
          :ip$ = "":dm$ = "":rn$ = "":er$ = "":em$ = ""
CH  5030 et$ = "":es$ = "":a$ = "0:":an$ = "&:"
          :b$ = ",p,r":c$ = ",u,w":r$ = "r0:":eq$ = " = "
EF  5040 pq$ = "pR15,":ur$ = ",u,r":m$ = "??"
FF  5050 ck = 0:nu = 0:ra = 0:hi = 0:lo = 0:bl = 0:w = 0:ef = 0
          :ip = 0:z = 0:t = 0:by = 0:dm = 0
AO  5060 return
FI  5070 u$ = chr$(145):cl$ = chr$(147):d$ = chr$(17)
          :gr$ = chr$(30)
EP  5080 return
```

Listing 2: Assembler code for "dev 9" 1541 disk drive program

```
; **** DEVICE 9 ***
LDA  #$29  ;device number $09 plus $20
STA  $77   ;LSNADR - listener address
LDA  #$49  ;device number $09 plus $40
STA  $78   ;TLKADR - talker address
RTS        ;end.
```

Listing 3: Anti-knock 1541 program to stop head knocks

```
; *** ANTI-KNOCK ***
LDA  #$C5  ;set bits 0, 2, 6, and 7
STA  $6A   ;REVCNT - error recovery count
RTS        ;end.
```

Listing 4: BASIC program to create "&:dev 9" DOS exec file

```
GK  10 rem device 9 dos exec file-run this,
OI  20 rem then to set drive to device 9:
LD  30 rem open1,8,15:print#1,"&:dev 9"
MJ  40 :
FP  50 open 8,8,12,"0:&:dev 9,u,w"
JA  60 fori = 1to13:reada:print#8,chr$(a);:next:close 8
OM  90 :
IE  100 data 0, 3, 9, 169, 41, 133, 119, 169, 73, 133,
        120, 96, 45
```

Listing 5: Used to create "&:anti-knock" file

```
MH  10 rem anti-knock exec file -run this,
NN  20 rem then to disable head knock:
KI  30 rem open1,8,15:print#1,"&:anti-knock"
MJ  40 :
CI  50 open 8,8,12,"0:&:anti-knock,u,w"
LG  60 fori = 1to9:reada:print#8,chr$(a);:next:close 8
OM  90 :
PM  100 data 0, 3, 5, 169, 197, 133, 106, 96, 199
```

Alphabetize Your Disk Directory

Stacy McInnis
Upland, CA

Here is a useful utility for your software tool box. Eventually your disk will become filled and it will seem like a good idea to remove some of your no longer needed files. At other times you can not remember the name of a file but you are sure you would recognize the name if you saw it. Possibly you would like the entries in your disk directory to correspond to those in your file cabinet. When any of these times come you may feel your task would be a lot easier if your directory list were in alphabetical order. Alpha is a program for the Commodore 64 and Plus/4 that will alphabetize the directory list in memory.

To use Alpha. . .

Type in the BASIC program "Alpha" and RUN it. The BASIC program POKES the machine language program Alpha into memory starting at location 30000 and ending at location 30263. Alpha also uses locations 29520 to 29994 for temporary storage.

Now that you have Alpha in place, load your directory as you normally would:

```
LOAD "$",8
```

To perform the alphabetization, type:

```
SYS 30000
```

A somewhat typical directory of 20 entries will alphabetize instantaneously. A directory of 144 entries and many similar names takes about 15 seconds. When you see READY on the screen with the blinking cursor your alphabetization is complete. Now LIST your directory and you will find that all the entries are in alphabetical order.

How Alphabetization Is Performed

When your directory is loaded it is stored in memory in a set format at the start of BASIC text. The start of BASIC text is not the same for the Plus/4 and the 64. However,the address of the start of BASIC text for both machines is stored in locations 43 and 44. Alpha looks at the file name within each directory entry. If any two consecutive file names are not in alphabetical order, their directory entries are exchanged. Alpha repeats this exchange operation until all entries are alphabetized. The alphabetizing is performed on the directory list as it appears in memory. Alpha in no way alters the directory as it appears on the disk or the order of the files on the disk.

Listing 1: BASIC loader for the alphabetize program

```
AN  10 rem* data loader for ''Alpha'' *
LI  20 cs=0
KA  30 for i=30000 to 30262:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
AI  60 if cs<>29860 then print" ***** data error
       ***** ": end
II  70 print" sys 30000 to sort directory"
AF  80 end
IN  100 :
EE  1000 data 160,   3, 185,   3,   0, 153,  80, 115
MC  1010 data 136,  16, 247, 160,   0, 140,  84, 115
IA  1020 data 177,  43, 133,   3, 141,   8, 116, 200
NH  1030 data 177,  43, 133,   4, 141, 153, 116, 160
LI  1040 data   4, 177,   3, 201,  34, 240,   7, 200
BE  1050 data 192,   8, 240,  35, 208, 243, 174,  84
BJ  1060 data 115, 152, 157,  87, 115, 238,  84, 115
LF  1070 data 232,  24, 165,   3, 105,  32, 133,   3
JD  1080 data 157,   8, 116, 165,   4, 105,   0, 133
JG  1090 data   4, 157, 153, 116,  76,  79, 117, 173
OJ  1100 data  84, 115, 201,   2, 176,   1,  96, 169
BG  1110 data   0, 141,  85, 115, 169,   1, 141,  86
CC  1120 data 115, 172,  86, 115, 174,  85, 115,  24
PE  1130 data 189,   8, 116, 125,  87, 115, 133,   3
HE  1140 data 189, 153, 116, 105,   0, 133,   4,  24
AH  1150 data 189,   9, 116, 125,  88, 115, 133,   5
OJ  1160 data 189, 154, 116, 105,   0, 133,   6, 177
FA  1170 data   3, 209,   5, 240,  12, 144,  20,  32
BE  1180 data 238, 117, 206,  85, 115,  48, 192,  16
LB  1190 data 195, 238,  86, 115, 172,  86, 115, 192
HH  1200 data  15, 208, 190, 238,  85, 115, 174,  84
JP  1210 data 115, 202, 236,  85, 115, 240,   3,  76
FC  1220 data 140, 117, 160,   3, 185,  80, 115, 153
CI  1230 data   3,   0, 136,  16, 247,  96, 174,  85
PI  1240 data 115, 189,  87, 115, 168, 189,  88, 115
HH  1250 data 157,  87, 115, 152, 157,  88, 115, 189
HA  1260 data   8, 116, 133,   3, 189, 153, 116, 133
GC  1270 data   4, 189,   9, 116, 133,   5, 189, 154
PM  1280 data 116, 133,   6, 160,  31, 177,   3, 153
EB  1290 data 232, 115, 136, 192,   1, 208, 246, 160
AO  1300 data  31, 177,   5, 145,   3, 136, 192,   1
ND  1310 data 208, 247, 160,  31, 185, 232, 115, 145
FD  1320 data   5, 136, 192,   1, 208, 246,  96
```

Listing 2: The PAL source code

```
FK  100 sys700 ;assembled on''PAL 64''
HO  110 .opt n
JL  120 ;alpha by stacy mcinnis
PN  130 ;alpha produces an alphabetized image of
BA  140 ;the disk directory that was loaded into
PL  150 ;the computer. alpha in no way alters
KE  160 ;the disk directory that is stored
LG  170 ;on disk.
HE  180 ;-------------------
PM  190 ;to use alpha:
OI  200 ; 1. load the alpha program
JO  210 ;      load "alpha",8,1
HI  220 ; 2. load the disk directory
JG  230 ;      load "$",8
MF  240 ; 3. sys to alpha to alphabetize
GP  250 ;      sys 30000
GH  260 ; 4. now list or print your directory
GP  270 ;   as you normally would
GF  280 ;---------------------
BB  290 ;data definitions
KG  300 ;---------------------
JG  310 adz1    =   $03    ;zero page locations
JE  320 adz2    =   $04    ;for indirect addressing
BI  330 adz3    =   $05    ;zero page locations
BG  340 adz4    =   $06    ;for indirect addressing
II  350 dirl    =   $2b    ;low byte of address of
IC  360 ;beginning of directory list
JN  370 dirh    =   $2c    ;high byte of address of
MD  380 ;beginning of directory list
PC  390 *       =   $7350  ;decimal 29520
DP  400 zsav    .byt 0     ;save contents of
DB  410 .byt 0             ;zero page locations that are
IH  420 .byt 0             ;used in indirect addressing.
OG  430 .byt 0             ;these are restored at
AL  440 ;program completion
EC  450 tolent  .byt 0     ;count of file names
KB  460 ;in directory
PB  470 dirent  .byt 0     ;entry in directory entry
PD  480 ;address list being considered
KA  490 testch  .byt 0     ;pointer to character
IK  500 ;in name being considered
IG  510 offset  =   *      ;filename offset from
KF  520 ;beginning of directory entry
MK  530 *       =   *+145
KG  540 savnam  =   *      ;area to save the directory
GB  550 *       =   *+32   ;entry in while it is
DM  560 ;being exchanged
DE  570 adentl  =   *      ;low byte of address of
EO  580 *       =   *+145  ;directory entry
LI  590 adenth  =   *      ;high byte of address of
IP  600 *       =   *+145  ;directory entry
JE  610 ;-------------------
EB  620 ;entry for alphabetizing
CH  630 ;  sys 30000
HG  640 ;-------------------
CI  650 *=$7530
LH  660 ;-------------------
GF  670 ;save locations 3,4,5,6. restore
JM  680 ;these locations at completion of
BD  690 ;program. these are used for
NL  700 ;indirect addressing.
NK  710 ;-------------------
```

```
BC  720         ldy  #3
IM  730 lsav    lda  adz1,y   ;save contents of
OI  740         sta  zsav,y   ;of 3,4,5,6 in zsav
JK  750         dey
NO  760         bpl  lsav     ;loop until all saved
JO  770 ;-------------------
KB  780 ;fill array adentl with the low byte
IB  790 ;of the address of the entry in
PD  800 ;in the directory.
AE  810 ;fill array adenth with the high byte
GD  820 ;of the address of the entry in
EH  830 ;the directory.
ME  840 ;set tolent to the total number of
KF  850 ;entries in the directory.
DE  860 ;-------------------
JH  870         ldy  #0       ;initialize the count of the
DB  880         sty  tolent   ;number of directory entries
FN  890         lda  (dirl),y ;low byte of the beginning
LA  900         sta  adz1     ;of the first directory entry
OO  910         sta  adentl   ;save entry address
IH  920         iny
CL  930         lda  (dirl),y ;high byte of the beginning
HD  940         sta  adz2     ;of the first directory entry
OA  950         sta  adenth   ;save entry address
LD  960 lc      ldy  #4       ;scan for " that begins
CH  970 llc     lda  (adz1),y ;the filename
FG  980         cmp #$22      ;$22 is a "
PI  990         beq setpt     ;branch if have a $22
JN  1000        iny           ;search on for a $22
GL  1010        cpy  #8       ;if not in 8 are finished
MO  1020        beq  outlpc   ;is not a filename
FD  1030        bne  llc
BC  1040setpt   ldx  tolent   ;entry in table
BN  1050        tya
DI  1060        sta  offset,x ;save offset to name
HP  1070        inc  tolent   ;count entries
AA  1080        inx           ;entry index
AM  1090        clc           ;increment pointer to names
BD  1100        lda  adz1     ;in directory
NB  1110        adc  #$20     ;distance between names
BK  1120        sta  adz1
AP  1130        sta  adentl,x ;save entry address
LH  1140        lda  adz2
LF  1150        adc  #0
NM  1160        sta  adz2
AB  1170        sta  adenth,x;save entry address
AN  1180        jmp  lc       ;loop until all addresses stored
KM  1190outlpc  lda  tolent   ;there must be at
FD  1200        cmp #2        ;least two entries in order
LN  1210        bcs  l1       ;to alphabetize
BD  1220        rts           ;if not return
FL  1230;-------------------
PC  1240;compare each entry with the entry
BC  1250;following it.
CC  1260;if the entries are in alphabetical
LJ  1270;order go on to test next entry.
LI  1280;if the entries are not in alphabetical
DE  1290;order exchange them in the directory
HM  1300;list  and restart alphabetizing
GH  1310;with the preceeding entry
LL  1320;-------------------
MM  1330;begin loop with first character
LC  1340;first directory entry
JN  1350;-------------------
```

```
EI  1360l1        lda  #0        ;point to first
GK  1370          sta  dirent    ;entry to test
HP  1380;---------------------
IA  1390;begin loop with first character
LA  1400;---------------------
OC  1410l2        lda  #1        ;point to first character
HG  1420          sta  testch    ;to alphabetize by
JC  1430;---------------------
GG  1440;begin loop with character and
MF  1450;directory  entry incrementing
HE  1460;---------------------
KB  1470l3        ldy  testch    ;index for character
LJ  1480          ldx  dirent    ;point to address of name
IE  1490          clc
PE  1500          lda  adentl,x ;add offset to
GA  1510          adc  offset,x  ;beginning of entry
NN  1520          sta  adz1      ;to point to
JO  1530          lda  adenth,x;file name
BO  1540          adc  #0
CE  1550          sta  adz2      ;file name
IK  1560          clc            ;add offset to beginning of
CK  1570          lda  adentl + 1,x  ;entry to point to
ON  1580          adc  offset + 1,x  ;filename
PH  1590          sta  adz3
KP  1600          lda  adenth + 1,x
HC  1610          adc  #0
BK  1620          sta  adz4
LA  1630          lda  (adz1),y ;characters to
OB  1640          cmp (adz3),y ;compare
JC  1650          beq  nomov    ;same character
HA  1660          bcc  next     ;branch if already in order
CA  1670          jsr  change   ;else go interchange
PH  1680          dec  dirent   ;point to preceeding
NK  1690;to test if must also
PJ  1700;exchange preceeding pair
EP  1710          bmi  l1       ;if minus start at beginning
EJ  1720          bpl  l2       ;start with preceeding
ND  1730nomov     inc  testch   ;preceeding characters ok
LC  1740          ldy  testch   ;now compare next character
BJ  1750          cpy  #15      ;only 16 characters in
NF  1760          bne  l3       ;name (0-15)
PJ  1770next      inc  dirent   ;point to next entry
AM  1780          ldx  tolent   ;to alphabetize
GD  1790          dex           ;tolent begins count at 1
EL  1800          cpx  dirent
JN  1810          beq  endlis   ;out if all finished
KN  1820          jmp  l2       ;go to compare with next entry
BO  1830endlis    ldy  #3       ;restore saved zero
GK  1840lres      lda  zsav,y   ;page contents
JF  1850          sta  adz1,y
PP  1860          dey
NP  1870          bpl  lres
HK  1880          rts           ;alphabetization is complete
JB  1890;have reached the end of
MC  1900;list so return to basic
NF  1910;---------------------
PH  1920;enternal routine to exchange
IC  1930;two directory names
LH  1940;---------------------
IA  1950change    ldx  dirent   ;entry to lower
FH  1960          lda  offset,x ;exchange offsets
BI  1970          tay
IK  1980          lda  offset + 1,x
GD  1990          sta  offset,x
```

```
HI  2000          tya
EA  2010          sta  offset + 1,x
MD  2020          lda  adentl,x ;move entry address
JI  2030          sta  adz1      ;to zero page
IP  2040          lda  adenth,x
HE  2050          sta  adz2
PO  2060          lda  adentl + 1,x;entry to raise
PF  2070          sta  adz3
KN  2080          lda  adenth + 1,x
HH  2090          sta  adz4
EJ  2100          ldy  #31       ;index for character
CI  2110;exchange
DB  2120lpsav     lda  (adz1),y ;get first name
PB  2130          sta  savnam,y;and save
HB  2140          dey
FM  2150          cpy  #1
BD  2160          bne  lpsav
OF  2170          ldy  #31
HP  2180lpmv1     lda  (adz3),y ;move 2nd name
AM  2190          sta  (adz1),y ;to first name position
DF  2200          dey
BA  2210          cpy  #1
KI  2220          bne  lpmv1
BD  2230          ldy  #31       ;moved saved name to
EF  2240lpmv2     lda  savnam,y;2nd name position
LL  2250          sta  (adz3),y
PI  2260          dey
ND  2270          cpy  #1
HM  2280          bne  lpmv2
ON  2290          rts
IN  2300.end
```

Auto Default For The Commodore 64

Tony Doty
Sandy, UT

With its declining prices and more sophisticated software, many one–time VIC-20 owners are changing to the Commodore 64. Sprites and 64k of RAM make the 64 one of the most popular personal computers on today's market. The 40 column screen adds to the professionalism of this work–horse. However, one is left with the opinion that Commodore overlooked two very important items during the transition between the machines. These are the default screen colours and the choice of the cassette as the primary program storage device.

The most notorious transition between the VIC-20 and the Commodore 64 is the new screen colours for the 64. The light blue characters on dark blue background scheme that Commodore chose for the 64 lacks the needed contrast between the background and characters for long hours of programming. In fact, with the introduction of Commodore's portable 64, the SX-64, and its five–inch colour monitor, the staff at Commodore must have recognized this problem. The default of the SX-64 video screen (blue characters on a white background), dramatically improved the video images displayed on the five inch colour monitor. The biggest disappointment with the changes to the SX-64 ROM was the fact that after all the time and expense Commodore went through to engineer the changes, the SX-64 reflects only a partial upgrade of the ROM over its predecessor. A new power up message and screen colours were included, but the default to the disk drive was not implemented (even though there is no provision for a cassette unit on the SX-64), though these changes could have been easily added.

After working with the VIC-20 and the Commodore 64, one starts to believe that Commodore used virtually the same operating system for both machines. The most immediate draw back of this decision is apparent in the choice of LOAD and SAVE default devices. Using the cassette as the default device was understandable with the VIC-20. The software writers seemed to target their wares toward the 3.5k unexpanded version of the VIC with a cassette. With the 3.5k of user memory, the LOAD time for programs was not a critical factor when compared to the cost of the disk drive. With the placement of the 64 in a market where more and more of its primary software is offered on floppy disk format, it seems only fitting that the LOAD and SAVE device defaults to the disk drive.

Making The Changes

I was quite dismayed with these two aspects of the Commodore 64. It wasn't long before investigation of the machine's internal code was my top priority. A method of setting things right had to be found. The areas I wanted to change were (1) The screen colours that the 64 powered up with (to dark blue characters on a white background), and (2) Set the I/O default device to 8. The programs provided in this article are the result of the tests required to ensure that the change would execute without problems on the computer.

Hardware or Software

For those who, like myself, feel that these changes should be made a permanent part of the operating system, I submit a list of the memory locations that were changed for the final hardware EPROM version.

Address	Old	New	Effect
$E1DA	$01	$08	Set Default Device To Disk Drive
$E535	$0D	$06	Set Character Colour To Dark Blue
$ECDA	$06	$01	Set Background Colour To White

Also changed was the 'LOAD/RUN' message displayed when the shifted RUN/STOP is pressed. The finished version will LOAD " * " and RUN the first program on disk when the shifted RUN/STOP key is pressed. Also included in the changes was the addition of my name in the power up message denoting my ownership of the system. Several other recommended changes have been published in various articles which could also be implemented. The 8k by 8 Bit device used for the hardware version listed here is the MCM 68764 C. The MCM 68764 C was purchased from a local Hamilton Avnet distributor for under $20.00. The new EPROM will replace the chip #901227-02 in the 64.(Editors Note: The chip specified is available in Canada for $34.50 as priced through Future Electronics - try the C2764-30 28 pin chip ($13.50 Cdn.) with a 28 to 24 pin adapter socket if you can find one)

A hardware change!! For many owners the thought of opening their computer and changing the chips is a horrifying ordeal to reckon with. Others may lack the needed equipment or knowledge to make the hardware changes. Thus, the purpose of the software program "Auto-Default". When loaded, the program will execute, change the default screen colours and device #, then reset the machine. The changes will remain a part of the operating system until a hardware reset is activated.

The Auto-Default Program

The Auto-Default program was derived from the desire to make the operating system of the Commodore 64 interface better with outside peripheral devices. Once the program is in place in memory, the user will find that the LOAD and SAVE commands are geared toward the disk drive. The 64 will now default to the disk drive when a device is not specified. To LOAD the directory, a LOAD"$" with a (return) is all that will be needed. To SAVE a program, type SAVE"name" then (return). The use of the cassette unit will still be supported by the Auto-Default program, although the user will now have to use a device number of one for tape LOADs and SAVEs. In addition to the LOAD and SAVE changes, the new screen colours and white background with blue characters should enhance the video images. Different colours or default device number can be altered by changing the values listed in the program.

BASIC vs. Machine Language

The Auto-Default program reflects the advantages of learning and using machine language routines to speed up the execution of programs. If written in BASIC, the two major FOR/NEXT loops used to move the 16864 bytes of ROM into RAM would have exceeded 1 minute and 20 seconds to execute. The excessive amount of time to execute such a utility program would have overshadowed its usefulness, making it impractical to use each time the computer was turned on. Compare this with the time of 10 seconds it will take to LOAD and execute the machine language version of Auto-Default. The advantages are overwhelming. For those who are new to machine language programming, it is advisable to type in the Auto-Default program from its BASIC listing to help you better understand the logic of this program in respect to the machine language source listing provided.

Designing The Program

When writing a program such as this, there are several questions the programmer must ask. First, where in memory would the program best fit and interfere the least? The answer this time was found in the memory from $0200-$025B. The 88 bytes of RAM located here are called the 'system input buffer'. No conflicts with the operating system were encountered here.

The second question, how to make the program as user friendly as possible. What does user friendly mean? To most, the term is one of many sales pitches given each time they want to know if a piece of software or hardware is difficult to operate. To the Auto-Default program, user friendly means no SYS calls to remember, no RUN to type, and no instruction book to memorize. To accomplish all of the above criteria, the program must be able to take control of the system, make the necessary changes, and return the control of the system to normal operation without the need of excessive user intervention. This was managed in the Auto-Default program with the help of the 'auto start' method of executing the program.

Using The Auto Start

What comes to mind when an auto start program is referred to? For most, their thoughts render a cartridge that is plugged into the back of the computer that holds a favourite game or utility program. This method of starting programs using a ROM cartridge is very effective, but due to the cost of EPROM burners, printed circuit boards, and EPROMS, this method is out of reach of most home computer enthusiasts.

However, there is a second method of creating an 'auto-start' program, using only software and altering the return address of the LOAD routine. To understand the principles of the 'auto-start', first we must examine the LOAD statement, and its machine language routines. When the statement

```
LOAD"AUTO-DEFAULT",8,1
```

is entered, the operating system must interpret the statement, set the parameters, and execute the routine associated with the BASIC statement (if the statement is entered with the correct syntax). An address to the machine language routine which is the starting point of the BASIC LOAD statement is loaded from a table of BASIC commands at $A00C. To access the LOAD routines at $E168, a JSR (Jump Save Return or more commonly called Jump to SubRoutine) will be used to call the machine language routine. The JSR will take the contents of the program counter, add 2 to the count, and store the sum of the addition on the 'stack' for later use. The following two bytes after the JSR are loaded into the program counter, and program execution continues at this new address. At the completion of the LOAD, an RTS (ReTurn from Subroutine) will be encountered. This instructs the processor to pull the first two bytes off the stack, (remember that JSR stored them), and return to that address. Next, the program counter is adjusted to reflect the next byte of the program by adding 1 to the return address. The 6510 uses this address to continue processing the program.

With that information, it's time to make an auto run routine. The program will be loaded; LOAD"AUTO-DEFAULT",8,1 (return). The return address will be stored by the JSR onto the stack. The program will start loading at $0101, and as such will overwrite the stack. At the end of the LOAD routine, the RTS will pull the next two bytes off the stack, now set at $02, add 1 to the address, then load the sum into the program counter. The program will resume processing at the address $0203, which marks the beginning of the Auto-Default program in memory.

The 6510 Processor

The heart of the Auto-Default program centers around the copying of the BASIC ROMs into RAM which is located under their resident addresses. RAM under ROM, how can this be? Due to the design of the 6510 processor (and its onboard 8 Bit I/O port), more than the standard 65535 bytes of memory can be accessed by the processor. By switching blocks of memory in and out, the 64 can control the 64k RAM plus 20k of ROM used in its operating system. The onboard I/O port of the 6510 is controlled by the eight bits of memory in location $01. Bits 0,1,2 control the internal memory and determine if RAM or ROM will appear in a given location. Bits 3,4,5 are assigned the task of controlling the cassette unit. Bit 3 controls the cassette write line, bit 4 the cassette sense switch (senses if the play key is pressed), and bit 5 the cassette motor control line. At this time the bits 6 and 7 are not used by the operating system, leaving them vacant for future use. The chart in Figure #1 will illustrate the changes that occur in the 64's memory when the I/O port is reconfigured by the first three bits of location $01.

Figure 1: C64 Location $01, Bits 0–2

Bit	Status	Comments
0	Set 1	ROM Memory $A000–$BFFF (BASIC ROM)
0	Clear 0	RAM Memory $A000–$BFFF
1	Set 1	ROM Memory $E000–$FFFF (Kernal ROM)
1	Clear 0	RAM Memory $A000–BFFF RAM Mem $E000–$FFFF (No Operating System)
2	Set 1	I/O Devices Appear in Processor Area
2	Clear 0	Character ROM Appears in Processor Area

Problems, Problems, Problems

Two unexpected problems were encountered in the testing of the software. Both were overcome, but I feel they are worth mentioning. Much to my amazement, the first time the program was powered up, the BASIC bytes had increased from 38911 to well over 51000. This was due to the routine that sets the top of BASIC memory finding RAM at the address of $A000, instead of the usual ROM. To correct the amount of memory that BASIC can use, the machine code at $FD88-$FD8B was changed to LDX #$A0 and LDY #$00 to force the top of the BASIC memory to $A000. This change was necessary to keep BASIC from storing its variables in the RAM above $A000 and overwriting the modified BASIC operating system now in RAM.

In past, the only method used to change the screen colours and device default was to POKE in the new values into the corresponding memory locations, only to run the risk of losing everything due to a RUN/STOP RESTORE. The correction of this problem was the factor that made this utility program truly useful. When the reset routine is called at location $FCE2, a second subroutine is called to set the I/O default values for the 6510 I/O port. This routine sets the system to read the ROM that is located in the memory of $A000-$BFFF and also $E000-$FFFF. By changing the contents of $FDD6 to $E5 (11100101 binary), the system, upon a reset call, will use the new memory that has been switched in (refer to the chart in Figure #1 for further explanation), $A000-BFFF and $E000-$FFFF as the operating system, keeping our defaults intact!

Understanding The Program

The program file writer does just what the name implies. It writes a program file to the diskette which the 64 can directly LOAD and execute. What denotes a program file? When the DOS (Disk Operating System) is instructed to write a program file, by the statement OPEN 5,8,5,"NAME,P,W" or OPEN 5,8,1,"NAME", the first two bytes that are sent to the disk drive will be the address, in low byte/high byte order, of the starting point in the 64's memory the program will be located. Lines 120 and 130 OPEN the file and send this address to the disk. Next, line 140 sends the disk 258 $02's that, when loaded by the program file, will overwrite the stack insuring the return address of $0203 at the end of LOAD. This will be the start address of the Auto-Default program. Line 150 READs the data statements and sends the program data to the disk. This is the actual program that will be located at $0203. Line 80 closes the program file.

Using The Program

After typing in the program file writer, save it on a separate disk. To make a working copy of the program, LOAD"PROGRAM FILE",8. Next, insert a new, formatted disk and type RUN (return). The program will be written out to disk and executed. Save the Program File Writer to disk to make additional copies of the Auto-Default program. To LOAD the program type

```
LOAD"AUTO-DEFAULT",8,1
```

Clearly, the best way to make these changes is to replace the operating system ROM with a custom burned EPROM. However, if making hardware changes doesn't appeal to your interest, the program provided in this article can still be of significant help. At last, there is a way to select the screen colours that work best with your equipment and have them stay even after pressing RUN/STOP RESTORE. Even nicer is not having to specify the disk drive for LOADs and SAVEs. Who knows, maybe you will like it enough to take the time to learn about making your own custom EPROMs.

I hope that this program will be of some use to you. The screen colours and default device are of my preference. With a little study, the program should be easily modified to fit each programmer's needs.

```
DN  100 rem ** auto-default basic listing **
CO  110 :
JC  120 rem move basic rom into ram memory
KO  130 for a=40960 to 49151: poke a,peek(a): next a
AA  140 :
OM  150 rem move kernal rom into ram memory
DA  160 for a=57344 to 65535: poke a,peek(a): next a
OB  170 :
MM  180 poke 57818,8:  rem change i/o default to disk drive
OA  190 poke 58677,6:  rem change cursor to dark blue
NM  200 poke 60634,1:  rem change background color to white
HA  210 poke 64982,229: rem change i/o reset table
NO  240 poke 64904,162: rem force the top of basic memory
           to $a000
CL  250 poke 64905,0
ML  260 poke 64906,160
JM  270 poke 64907,160
LH  280 poke 1,53:    rem switch in ram memory at $a000
JH  290 sys64738:     rem system reset call
```

```
NK  100 rem ** writes 'auto-default' auto-run code to diskette
FE  110 rem ** written by tony doty
KD  120 gosub 170: open 5,8,5,"auto-default,p,w"
MC  130 print#5,chr$(0);:print#5,chr$(1);
PP  140 for l=0 to 258: print#5,chr$(2);: next l
LC  150 for l=0 to 89: read a: print#5,chr$(a);: print a;: next l
DP  160 close 5: load "auto-default",8,1
HP  170 open 15,8,15,"s0:auto-default": close15: return
FP  180 data 169, 160, 133, 254, 169, 224, 133, 252
NL  190 data 169,   0, 133, 251, 133, 253, 168, 177
CA  200 data 253, 145, 253, 177, 251, 145, 251, 200
DA  210 data 208, 245, 230, 254, 230, 252, 165, 252
NF  220 data 208, 237, 169
BE  230 data   8: rem ** default device nunber
LF  240 data 141, 218, 225, 169
EH  250 data   6: rem ** default cursor color
PH  260 data 141,  53, 229, 169
NG  270 data  14: rem ** default bordor color
LJ  280 data 141, 217
ML  290 data 236, 169
AA  300 data   1: rem ** default background color
JG  310 data 141, 218, 236, 169,  53
FE  320 data 133,   1, 169, 229, 141, 214, 253, 169
HG  330 data 162, 141, 136, 253, 169,   0, 141, 137
FK  340 data 253, 169, 160, 141, 138, 253, 169, 160
HH  350 data 141, 139, 253,  76, 226, 252,   0,   0
ON  360 data   0,   0
```

PAL Source for Auto-Default

```
IK  100 rem ** auto default 64 by tony m. doty  sandy, utah **
CO  110 :
AJ  120 open 4,8,1,"@0:auto default.obj"
NB  130 sys(700)
DK  140 .opt o4
LA  150 *         =    $0100     ;the stack for auto-run
GB  160 ;
CH  170 hitemp    =    $fb
MJ  180 lotemp    =    $fd
FH  190 savdef    =    $e1da
JE  200 chrcol    =    $e535
AI  210 bkgrnd    =    $ecda
LC  220 ioport    =    $01
LM  230 bdrcol    =    $ecd9
IB  240 iovect    =    $fdd6
PK  250 reset     =    $fce2
KH  260 ;
CD  270 ; ** stack to be filled with $02's for start at $0202
KC  280 .byt $02, $02, $02
IJ  290 ;
OA  300 *=$0200
IE  310 .byt $02, $02, $02
GL  320 ;
MF  330 start     =     *           ;start the code at address $0203
KH  340           lda   #$a0        ;get addr for basic routines
FK  350           sta   lotemp+1;and store the hi byte in ptr
NC  360           lda   #$e0        ;get the addr for the operating
DH  370           sta   hitemp+1;system and store it
HO  380           lda   #0          ;get low order byte
CI  390           sta   hitemp      ;for both and store
FF  400           sta   lotemp      ;them in the temp pointer
KJ  410           tay               ;clear a counter for the move
KB  420 ;
NL  430 nxtpag    =     *
FA  440           lda   (lotemp),y ;read the data from
KM  450           sta   (lotemp),y ;rom and then write it to
IJ  460           lda   (hitemp),y ;ram for both the high and
FI  470           sta   (hitemp),y ;low rom chips
AM  480           iny
FH  490           bne   nxtpag
FC  500           inc   lotemp+1
PB  510           inc   hitemp+1
JA  520           lda   hitemp+1
NJ  530           bne   nxtpag
CJ  540 ;
FP  550           lda   #8          ;set default i/o to 8
FA  560           sta   savdef
DI  570           lda   #6          ;character colour blue
GK  580           sta   chrcol
FD  590           lda   #$0e        ;border light blue
BL  600           sta   bdrcol
PL  610           lda   #1          ;background white
LK  620           sta   bkgrnd
JP  630           lda   #$35        ;select configuration of
EP  640           sta   ioport      ;6510's pia
CA  650           lda   #$e5        ;pia value for reset routine
MP  660           sta   $fdd6
CI  670           lda   #$a2        ;force top of basic memory
FM  680           sta   $fd88       ;to $a000
NJ  690           lda   #0
HP  700           sta   $fd89
MB  710           lda   #$a0
DB  720           sta   $fd8a
AD  730           lda   #$a0
IC  740           sta   $fd8b
GO  750           jmp   reset
OG  760 ;
ON  770 .end
```

File Pursuit

Richard Evers, Editor

A File Trace and Size Utility For All Drives

A simple routine to perform a useful task. File Pursuit will allow you to perform a trace of any file held on diskette, excepting Relative files and files that have disk protection tricks standing in their way. As an added bonus, you will also get the directory track, sector, and index into the sector the file was found on, the index into the last sector consumed, and the total number of bytes occupied by the file. The byte count also includes the first two bytes found in the file, which could be the start address if the file type is PRG. To make everything fall together, output can also be directed either to the screen or printer.

The concept is simple. Once the file chosen has been found on diskette, the directory sector and index into the sector is found by looking at location DL+(256*DH) for the sector, and DL+DI+(256*DH) for the index. The table below will show you the values for each variable depending on the drive type.

Drive	Buffer	DT	DL	DH	DI	HY	DL+256*DH		DL+DI+256*DH	
1541	$0300	18	144	2	4	3	$0290	656	$0294	660
2031	$0300	18	144	2	4	3	$0290	656	$0294	660
4040	$1100	18	96	67	4	17	$4360	17248	$4364	17252
8050	$1100	39	96	67	8	17	$4380	17248	$4368	17256
8250	$1100	39	96	67	8	17	$4360	17248	$4368	17256

As can be deduced from the above, the 1541 and 2031 appear identical, as do the 8050 and 8250, as far we are concerned today. Now, for a greater shot at understanding, let's break down the variables.

DT = The directory track

DL/DH = Lo/Hi address in drive RAM where the directory sector of the file opened can be found.

DI = The index from DL+256*DH that the index into the directory sector can be found.

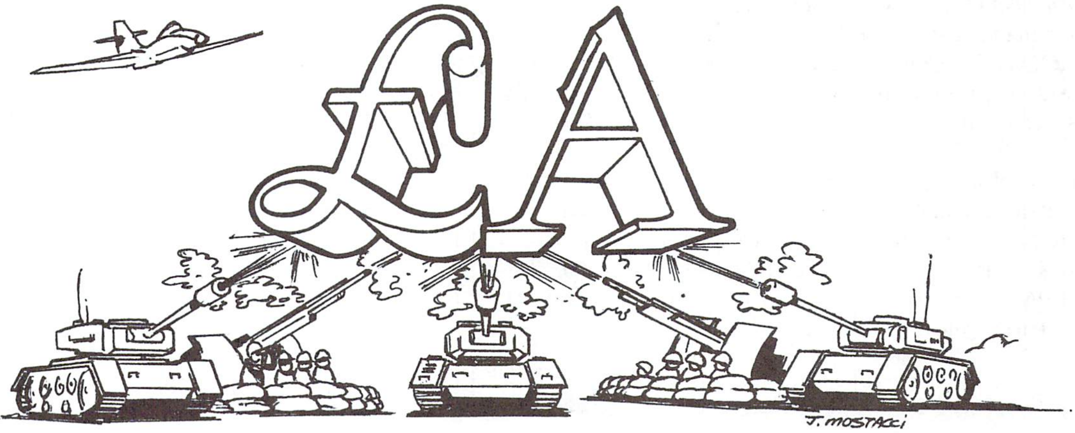
HY = High byte of the RAM buffer in use for this program.

Without going into a lot of unecessary detail, once the directory sector and index into the sector is determined, reading that location tells us the first data block in use. From there, the first two bytes are read from each data block to determine if more sectors are in use. If the track is greater than zero, then the answer is yes, and the pursuit continues. If not, then the last block has been reached, and the sector indicator points to the last byte of the file in the block. Once this has been determined, the total number of all bytes read are displayed, and the program terminates. And, due to the method of finding the information within the drive, the trace is fast. Not bad for a few minutes keying in time. The great pursuit, a Transactor specialty.

```
AJ | 100 rem save " @0:file pursuit " ,8
FB | 110 rem ** rte/84 – file trace + file size (includes
   |        first 2 bytes) **
MO | 120 :
HE | 130 print " ** file pursuit: trace and size ** "
PN | 140 z$ = chr$(0): ttl = 0
KA | 150 :
HH | 160 input " (s) screen or (p) printer " ;sp$: dv = 0
PN | 170 if sp$ = " s " then dv = 3
PN | 180 if sp$ = " p " then dv = 4
JF | 190 if dv = 0 then 160
MD | 200 :
LM | 210 print " (1) 1541/2031; (2) 4040; (3) 8050/8250  "
KB | 220 input dt$: if dt$< " 1 "  or dt$> " 3 "  then 220
CH | 230 dt = 39: dl = 96: dh = 67: di = 8: hy = 17: if dt$ =  " 2 "
   |        then dt = 18: di = 4
KM | 240 if dt$ =  " 1 "  then dt = 18: dl = 144: dh = 2
   |        : di = 4: hy = 3
OG | 250 :
PD | 260 input " dr#,filename " ;d$,f$
MM | 270 open 15,8,15
JH | 280 open 8,8,8,(d$) +  " : "  + (f$): get#8,a$ : if st
   |         then print " not found " : stop
GJ | 290 :
BN | 300 open 1,(dv): rem open screen/printer channel
KK | 310 :
CG | 320 rem ** file exists – get sector + index into dir trk **
EG | 330 print#15, " m–r " chr$(dl)chr$(dh)chr$(1)
FN | 340 get#15,s$: sec = asc(s$ + z$)
FH | 350 print#15, " m–r " chr$(dl + di)chr$(dh)chr$(1)
OK | 360 get#15,i$: ind = asc(i$ + z$)
KF | 370 close 8
AP | 380 :
DL | 390 print#1, " filename:  " f$ "  on dr# " d$
EN | 400 print#1, " found: dir track " dt " sector " sec " index " ind
OA | 410 :
CK | 420 open 8,8,8, " #0 "
IO | 430 print#15, " u1 " ;8;val(d$);dt;sec: rem read in block
AF | 440 print#15, " m–r " chr$(ind + 1)chr$(hy)chr$(2)
   |        : rem get track, sector links
IB | 450 get#15,tr$,sc$: trk = asc(tr$ + z$): skt = asc(sc$ + z$)
FF | 460 if trk then 520: rem track = 0 if last block
LP | 470 print#1, " index into last sector  =  " skt
LL | 480 ttl = ttl + skt
PO | 490 print#1, " total file size  =  " ttl–255 " bytes
GD | 500 close1: close8: close15: end
CH | 510 :
KG | 520 print#1, " link : track " trk " sector " skt
GJ | 530 print#15, " u1 " ;8;val(d$);trk;skt
IF | 540 print#15, " m–r " chr$(0)chr$(hy)chr$(2)
OL | 550 ttl = ttl + 254:rem ** add up file size
CO | 560 goto 450
```

Supernumbers For The Commodore 64

John R. Bennett
Ann Arbor, MI



Finally — Indestructible Variables!

When I got my Commodore 64, I was amazed at its power and versatility. However, I had one minor gripe – editing a single line of a program causes all the variables to be cleared. I knew why this happens; variables are stored at the end of the program and when the program changes size, this location moves. To keep its memory pointers internally consistent, BASIC clears all variables whenever a program line is entered.

However, not all microcomputers do this. In particular, the Sharp PC–1500 stores variables at the top of memory in reverse order. In addition, it has permanent memory locations for the 26 single letter variables. Thus, you can run a fairly long program to calculate something, and then modify the display stage to present the answer in different ways without clearing the variables (as long as you restart the program with a GOTO statement rather than a RUN).

The permanent storage locations for the single letter variables has a nice side effect – programs run faster because the BASIC interpreter does not have to search through the variables looking for the one you want. One day, while I was killing time by disassembling BASIC, it occurred to me that Commodore had left enough holes (indirect jumps through RAM addresses) to allow me to remedy this problem. The resulting patch to BASIC, which I call 'SUPERNUMBERS' has all the advantages of the reverse, top of memory storage. In addition, because they are a new class of variables which are not touched by a normal CLR, Supernumbers are invulnerable to almost all modes of program failure. They survive RUNSTOP/RESTORE, LOADing programs, and even hitting a reset button. Only a power outage, a runaway program which pokes the memory locations where they are hidden, or Kryptonite can destroy them. Supernumbers also provide an excellent vehicle for passing floating point variables between machine language and a BASIC program.

To understand how Supernumbers are created you have to study the way BASIC handles the simple replacement statement:

```
X = Y
```

The BASIC interpreter examines the statement from left to right one character at a time. When it encounters the X, it stores it and then looks for the next character. Since the next character is an equal sign, it knows that it needs to evaluate an expression for a real number and store the number in the location reserved for X. At this point the interpreter calls a routine which finds the address of the variable and stores a pointer to it in $49–$4A. It then calls a routine we can call 'formula evaluator', whose job it is to evaluate the right hand side of the equation and return a number. Thus, to define a new variable type one is required to modify two portions of BASIC, the part which finds the address of a variable and the part which computes the value of a numeric expression.

To make Supernumbers instantly recognizable to BASIC, they are prefaced with the £ character (The English Pound symbol, between the minus and CLR/HOME keys). Although you could make any number and type of Supernumbers, I chose to make them real and to have only 26 corresponding to the 26 single letter variables A–Z.

When Supernumbers are used, the above assignment is done slightly differently. First of all, the assignment would appear as:

```
£X = £Y
```

When BASIC parses this statement, The first character it finds is the £ on the left hand side. It immediately signals the dreaded 'SYNTAX ERROR'. However, before anything nasty happens, the Supernumber wedge in the normal error routine checks to see if a £ caused the error. If it did, it checks the code up to the equal sign to see if it can handle it. If so, it puts the address of £X in the pointer $49–$4A, and returns to the BASIC interpreter at the proper place. It may seem like extra work to let a syntax error happen and then cancel it, but in fact it is much simpler and faster to do this than to put a wedge into the CHRGET routine like many methods do. The code in the CHRGET wedge is executed every time BASIC looks for another character, but the error wedge code is only executed when the wedge character is found. (*Editor's note: See "A New Wedge For The Commodore 64" by Brian Munshaw in the last issue – Vol 5 issue 06*). The Supernumber routine more than makes up for the small extra work by finding the address of £X very quickly. BASIC would have to look through the whole table of variables until it found the right one. In large programs this process can use up considerable time.

The occurence of a Supernumber on the right hand side of the equation is easier to handle because Commodore had the wisdom to put an indirect jump through RAM at a convenient spot. The Commodore 64 Programmer's Reference Guide calls the vector IEVAL; it is stored at $030A–$030B. This vector is used in the BASIC interpreter at $AE83, where it executes the instruction:

```
JMP ($030A)
```

At power-up the KERNAL puts the address of the next statement, $AE86, in this vector. The Supernumber routine writes its address there instead and looks for the use of the £ character. If a Supernumber is required, it calls another BASIC routine to move it to the floating point accumulator before returning control to the BASIC ROM.

The entire program and memory space for these 26 Supernumbers is less than 300 bytes. When assembled at $C000, a 'cold' start of Supernumbers, which sets them to zero, is invoked by the command:

```
SYS 49152
```

A 'warm' start, which keeps the old values, is invoked with:

```
SYS 49155
```

You can use Supernumbers anywhere you can use a standard floating point number, except as the counter in a FOR–NEXT loop. The execution speed gained by using them depends on the program. In some simple tests, I found a 30% increase in speed. Programs with a lot of variables may benefit more; those programs with a lot of string manipulation or I/O will benefit less. The biggest advantages of Supernumbers, however, are their invulnerability and their fixed memory locations. They can be easily passed to new or modified BASIC programs and can easily be accessed by machine language programs.

Editor's Note: "Supernumbers" is a great little utility; handy, fast, and short. The only drawback is that there's no such thing as Supernumber arrays. It would be nice, for example, to PRINT £A(I). No problem. You can index supernumbers and transfer them to normal arrays with a BASIC program. The short program in listing 3 uses a "dynamic keyboard" technique to transfer the 26 Supernumbers £a to £z into the array a(1) to a(26). It then prints the array out as a visual check.

One more thing: There appears to be more than 26 Supernumbers available; £ and ££, among others. How many can you find?*

Listing 1: Supernumbers BASIC loader

```
BI   10 rem* data loader for "supernumbers" *
LI   20 cs=0
KG   30 for i=49152 to 49447:read a:poke i,a
DH   40 cs=cs+a:next i
GK   50 :
HH   60 if cs<>36424 then print"***** data error *****": end
AH   70 print" sys 49152 for cold start"
AF   80 end
IN   100 :
CO   1000 data  32, 102, 192, 169,  61, 141,  10,   3
GC   1010 data 169, 192, 141,  11,   3, 169,  24, 141
HA   1020 data   0,   3, 169, 192, 141,   1,   3,  96
ID   1030 data 224,  11, 208,   4, 201,  92, 240,   3
MH   1040 data  76, 139, 227,  32, 115,   0,  32,  19
MO   1050 data 177, 233,  65,  10, 170,  32, 115,   0
NH   1060 data 189, 113, 192, 188, 114, 192, 162,   0
LB   1070 data 134,  13, 134,  14,  96, 169,   0, 133
OK   1080 data  13,  32, 115,   0, 176,   3,  76, 243
NJ   1090 data 188, 201,  92, 240,   3,  76, 146, 174
AM   1100 data  32, 115,   0,  32,  19, 177, 233,  65
DH   1110 data  10, 170,  32, 115,   0, 189, 113, 192
DL   1120 data 188, 114, 192,  76, 162, 187, 162, 130
MF   1130 data 169,   0, 157, 164, 192, 202, 208, 250
EP   1140 data  96, 165, 192, 170, 192, 175, 192, 180
BB   1150 data 192, 185, 192, 190, 192, 195, 192, 200
MM   1160 data 192, 205, 192, 210, 192, 215, 192, 220
GP   1170 data 192, 225, 192, 230, 192, 235, 192, 240
OB   1180 data 192, 245, 192, 250, 192, 255, 192,   4
GL   1190 data 193,   9, 193,  14, 193,  19, 193,  24
FB   1200 data 193,  29, 193,  34, 193, 157, 167,   2
MP   1210 data 165,   2, 197,   2, 240, 193, 169,   0
NM   1220 data 141, 253,   3, 133,   2, 162,   6, 202
FA   1230 data 189, 174,   2, 164,   2, 192,   0, 254
PM   1240 data   1, 193,   0, 208, 242, 134,   2, 134
PI   1250 data  52,   3, 134, 212,   3, 166, 245,   3
DP   1260 data  16,  65,  16, 149, 119,   2, 224, 134
DP   1270 data 245,   3, 166, 212,   3, 224,   1, 208
LB   1280 data 214, 165,  52,   3, 133, 198,  68,  49
II   1290 data 226,  68, 192, 193, 134,  54,   3, 165
BP   1300 data 197,  65,   6, 208,   4, 193,   3, 208
HD   1310 data  35,  80, 177,  17, 149,  20, 111, 227
AJ   1320 data  54,   3, 161, 226, 133,  21,   3,  80
NH   1330 data 165, 197, 193,   3, 208,  19, 134,  54
DG   1340 data   3, 238, 164,   2, 181, 194,   2,   0
GH   1350 data 210, 215, 192, 192,   6, 208, 245, 208
PJ   1360 data   7,  68,  51, 193, 193,   5, 208,   0
```

Listing 3: Transfer Supernumbers to regular BASIC array, A( )

```
100 rem supernumber arrays
110 rem this program puts "supernumbers"
120 rem £a-£z into the array a(1)-a(26)
130 :
140 dim a(26):print"qqq"
150 fori=1to26
160 poke 631,13: poke 198,1
170 l$=chr$(64+i)
180 i$=right$("  "+mid$(str$(i),2),2)
190 print"Qa("i$")=£"l$":contQQ";:end
200 next: rem now print out array
210 fori=1to26:printa(i),:next
```

Listing 2: Supernumbers PAL source

```
OH  10  :
KL  20  open2,8,2,"@0:supernumber.obj,p,w"
CJ  30  :
PB  40  rem   activate pal
GK  50  :
NA  60  sys700
ML  70  ;
BG  80  .opt o2
AN  90  ;
KC  100 ; supernumbers for the commodore 64
EO  110 ;
FP  120 ;     by john r. bennett
IP  130 ;
NP  140 ;     no rights reserved
MA  150 ;
HE  160 chrget    =    $0073  ;get next byte
AC  170 ;
CH  180 valtyp    =    $0d    ;data type (0=number)
HC  190 intflg    =    $0e    ;data type (0=floating point)
OD  200 ;
FK  210 ierror    =    $0300  ;the vectors to
KO  220 ieval     =    $030a  ;be modified
MF  230 ;
FI  240 olderr    =    $e38b  ;the old values
IA  250 oldeval   =    $ae86  ;in the vectors
KH  260 ;
PN  270 ; routine which returns with carry
EK  280 ; set if accumulator is a letter
LE  290 ckalph    =    $b113
CK  300 ;
MP  310 ; routine to load floating point
LL  320 ; accumulator with number pointed
LH  330 ; to by (a,y)
OO  340 memto1    =    $bba2
EN  350 ;
DB  360 ; routine which changes ascii to
PO  370 ; floating point - normally called
IK  380 ; by oldeval
IK  390 ascflt    =    $bcf3
GA  400 ;
LI  410 *         =    $c000
KB  420 ;
IB  430 cold      =    *                ; cold start
OC  440 ;
PC  450           jsr   clrram          ; set ram to zero
CE  460 ;
NI  470 warm      =    *                ; warm start
GF  480 ;
ON  490           lda   #<neweval       ; replace vectors
ML  500           sta   ieval
KK  510           lda   #>neweval
JK  520           sta   ieval+1
DP  530           lda   #<newerr
DI  540           sta   ierror
DA  550           lda   #>newerr
OH  560           sta   ierror+1
GC  570           rts
KL  580 ;
EM  590 ;
GA  600 newerr    cpx  #11              ;only forgive
AH  610           bne  realerr          ;syntax errors
CO  620 ;
```

```
PO  630           cmp #"£"          ;make sure a '£'
EI  640           beq  found        ;caused it
AA  650 ;
CA  660 realerr   jmp  olderr       ;not supernumber
EB  670 ;
CG  680 found     jsr   chrget      ;find out which
DF  690           jsr   ckalph      ;letter is next
CD  700 ;
IN  710           sbc   #"a"        ;compute position
OK  720           asl               ;in address table
FK  730           tax
CF  740           jsr   chrget      ;point to next byte
AF  750           lda   numtab,x    ;get address in
FC  760           ldy   numtab+1,x  ;(a,y) registers
JE  770           ldx   #0
FF  780           stx   valtyp      ;real number
JF  790           stx   intflg
MA  800           rts
AK  810 ;
OK  820 neweval   lda   #0
MJ  830           sta   valtyp
KG  840           jsr   chrget
MJ  850           bcs   nodigit
CN  860 ;
CK  870           jmp   ascflt      ;not supernumber
GO  880 ;
OK  890 nodigit   cmp #"£"
MN  900           beq  found1
EA  910 ;
JI  920           jmp  oldeval+12   ;not supernumber
IB  930 ;
NP  940 found1    jsr   chrget      ;find out which
HF  950           jsr   ckalph      ;letter is next
CN  960           sbc   #"a"        ;compute position
IK  970           asl               ;in address table
PJ  980           tax
EE  990           jsr   chrget      ;point to next
OF  1000 ;
AE  1010            lda   numtab,x    ;put supernumber
JB  1020            ldy   numtab+1,x  ;into floating
FP  1030            jmp  memto1       ;point acc.#1
GI  1040 ;
LP  1050 clrram     ldx   #130        ;set all super-
DO  1060            lda   #0          ;numbers to zero
CO  1070 morex      sta   ram-1,x
PO  1080            dex
AK  1090            bne  morex
ID  1100            rts
MM  1110 ;
MO  1120 ; address table for supernumbers
AO  1130 ;
JB  1140 numtab .word ram,ram+5,ram+10
ML  1150 .word ram+15,ram+20,ram+25,ram+30
KN  1160 .word ram+35,ram+40,ram+45,ram+50
IP  1170 .word ram+55,ram+60,ram+65,ram+70
GB  1180 .word ram+75,ram+80,ram+85,ram+90
IN  1190 .word ram+95,ram+100,ram+105
CH  1200 .word ram+110,ram+115,ram+120
AD  1210 .word ram+125
KD  1220 ;
ME  1230 ram*      =    * + 130
OE  1240 ;
OL  1250 .end
```

VARPTR: Creation of a new BASIC Function

Anthony Bryant
Winnipeg, MAN.

The VARPTR function, which returns the address of a given variable in memory, is standard in many BASICs but not, alas, Commodore BASIC. There are times, such as when executing a short machine language routine stored within a string, when it is desirable to know where the darned thing is in memory.!

Just change the USR vector (at 785–786 on the C64, 1–2 on others) to point to this little routine:

```
LDA $48    ;($45 for PET)
LDY $47    ;($44 for PET)
JMP $B391 ;C64 fix–float conversion
; ($D391  for  VIC–20,
$C4BC for BASIC 4.0,
$D26D for 2.0)
```

then, either in direct or program mode, USR(variable) will give the address of the variable, whether integer, real, or string. Note that with a string variable it points to the first byte of the string descriptor, and with real or integer variables it points to the data (not the name).

Thus, if you use a statement like

```
V = USR(ML$)
```

then

```
PRINT PEEK(V)
```

will show the length of ML$, and

```
PRINT PEEK(V + 1) + 256*PEEK(V)
```

will give the starting address of the string itself. Using this technique, you could put a short (less than 256 bytes) machine language program into the variable ML$, then SYS to the string's start address to execute the program. A clean solution, precluding the need to protect memory for a temporary program.

Here's a bit of BASIC to put the VARPTR program (for the C–64) in the cassette buffer and set up the USR vector accordingly.

```
5 rem varptr: " USR(var) "
10 fori = 828to834:read a:pokei, a:next
20 poke 785, 60: poke 786, 3
30 data 165, 72, 164, 71, 76, 145, 179
```

Editor's Note: This has to get the award for "Best 7–Byte Program"!

Extra C64 Editing Commands Using the Function Keys

While programming, and particularly when doing disk operations using filenames from a directory on the screen, it is handy to be able to clear the screen of a small portion – say one or two lines below the cursor or everything below the cursor, to allow room for disk messages.

With the installation of the following pre–interrupt routine, the function keys become:

F1 – cursor to lower left corner
F3 – clear one line below cursor line
F5 – clear five lines below cursor line
F7 – clear to bottom from below cursor line

The routine which follows (printed in both assemble source and BASIC loader formats) is relocateable anywhere, taking up only 75 bytes. In addition it can be used with the screen in any bank position as it uses the C–64's clear screen line routine in ROM (which according to Mr. Butterfield hasn't changed with the ROM updates – one lives in hope!).

There are some personal preference changes which can be made to the routine:

1) If you want F5 to clear other than five lines then modify the number of INX instructions.

2) As it stands, F5 won't clear five lines unless there at least five lines remaining to the bottom of the screen. Change the BCS EXIT instruction to BCS BOT and it will clear five lines or less, until the bottom of the screen is reached.

```
AI   10 rem* data loader for "screen edit" *
LI   20 cs=0
BG   30 for i=49152 to 49226:read a:poke i,a
DH   40 cs=cs+a:next i
GK   50 :
JA   60 if cs<>9168 then print"***** data error *****": end
DD   70 sys 49152
AF   80 end
IN   100 :
KB   1000 data 120, 169,  13, 141,  20,   3, 169, 192
JD   1010 data 141,  21,   3,  88,  96, 166, 214, 134
MN   1020 data   2, 165, 197,  41, 127, 201,   4, 208
JA   1030 data   4, 162,  24, 208,  38, 201,   5, 208
II   1040 data   3, 232, 208,  17, 201,   3, 208,   4
NA   1050 data 162,  24, 208,  17, 201,   6, 208,  24
FC   1060 data 232, 232, 232, 232, 232, 224,  25, 176
NA   1070 data   8,  32, 255, 233, 202, 228,   2, 208
KE   1080 data 244, 166,   2, 160,   0,  32,  12, 229
IO   1090 data  76,  49, 234
```

```
LN   100 sys700  ;to start "PAL" assembler
GN   110 .opt oo
JG   120 *       =    $c000
MK   130 ;screen editing routines:
PF   140 ; "F1" - cursor to lower left corner
KO   150 ; "F3" - clear one line
IB   160 ; "F5" - clear five lines
KK   170 ; "F7" - clear to bottom
ED   180 setup   =    *
HF   190         sei
HL   200         lda  #<newirq
GE   210         sta  $0314    ;irq vector
HM   220         lda  #>newirq
CK   230         sta  $0315
OH   240         cli
GO   250         rts
KH   260 ;
OC   270 newirq  =    *
KC   280         ldx  $d6
AH   290         stx  $02      ;save current cursor line
NN   300         lda  $c5
NA   310         and  #$7f     ;test current key pressed
KF   320         cmp  #4       ;test for key "F1"
IH   330         bne  f3
EP   340         ldx  #$18     ;bottom screen line
GG   350         bne  move     ;branch always to move cursor rtn
BA   360 f3      =    *
FH   370         cmp  #5       ;key "F3"
CL   380         bne  f7
LB   390         inx           ;point to line below cursor line
EC   400         bne  doit     ;clear line routine
PD   410 f7      =    *
HK   420         cmp  #3       ;key "F7"
AO   430         bne  f5
DI   440 bot     =    *
JP   450         ldx  #$18     ;clear from bottom to cursor
PC   460         bne  ckit     ;branch always
FH   470 f5      =    *
HO   480         cmp  #6       ;key "F5"
HD   490         bne  oldirq
KK   500         inx           ;point to five lines
OJ   510         inx           ;below cursor line
EO   520         inx
OO   530         inx
IP   540         inx
LH   550 doit    =    *
DE   560         cpx  #$19     ;check max screen line limit
DP   570         bcs  exit
JB   580         jsr  $e9ff    ;clear screen line routine
FA   590         dex
PJ   600 ckit    =    *
BM   610         cpx  $02      ;reached cursor line"?
PL   620         bne  doit     ;no
IO   630 exit    =    *
OB   640         ldx  $02      ;restore cursor line position
GB   650 move    =    *
JA   660         ldy  #0       ;left column position
AG   670         jsr  $e50c    ;pos'n cursor, fix line/color mem
OI   680 oldirq  =    *
LN   690         jmp  $ea31
IJ   700 .end
```

Bootmaker II

Jeff Goebel
Georgetown, Ont.

The program listed here is a special type of program. When you RUN it, it writes a BASIC program to your specifications. It allows people with no programming experience to design and create personalized "BOOT" programs in seconds, simply by answering a few simple questions.

A "BOOT" is a program that automatically loads another program. Its main purpose is to eliminate memory work. Not computer memory, but *your* memory. Some programs, like many machine language routines, need to be loaded using ",8,1" instead of the normal LOAD"NAME",8. If you have a lot of these programs on assorted disks all over, it sometimes becomes hard to remember which need the ",1" and which ones don't. Additionally, many programs require specific SYS commands to activate them. Since there are many locations in memory where a machine language program can sit, there are many different SYS commands to remember.

Simply running Bootmaker II will show you basically how it works — It asks the appropriate questions. When entering the program, note that many of the statements require that they be typed in EXACTLY as they appear.

```
EC  0 rem "  bootmaker ii – magazine version
FI  1 rem "
HP  2 rem "      transactor magazine
HI  3 rem "
NI  4 rem "    written by jeff goebel 1984
JI  5 rem "
KI  6 rem "
PG  10 poke53280,0:poke53281,0:printchr$(147)chr$(14)
EB  110 print" Input name of program to boot " :inputc$
AG  150 input" Screen colour of boot (1–16) " ;a$:b = val(a$)
MD  170 input" Border colour of boot (1–16) " ;a$:a = val(a$)
CK  175 input" Cursor colour of boot (1–16) " ;a$:c = val(a$)
GO  180 print" Screen message #1 while loading " :inputs1$
FP  190 print" Screen message #2 while loading " :inputs2$
PN  210 input" Activate command (sys or run) " ;ac$
JF  220 print" Save @: " c$ "  ? "
OD  230 geta$:ifa$ =  " " then230
KE  240 ifa$ =  " n " then1000
OH  245 b$ = c$:print " Boot name:  " ;b$:c$ = b$ +  " .1 " :print " It loads:  " ;c$
JP  246 print " ok? "
BK  247 geta$:ifa$<> " y " anda$<> " n " then247
EM  248 ifa$ =  " n " theninput " Input name to load " ;c$
IL  250 open15,8,15
OA  260 print#15, " c0: "  + c$ +  "  =  "  + b$ +  " "
IL  270 print#15, " s0: "  + b$ +  " "
PL  280 close15
AO  290 open15,8,15
IP  300 input#15,a$,a1$,a2$,a3$
LC  310 printa$; "   " a1$; "   " ;a3$; "   " ;a4$:goto1007
HN  1000 ifa$ =  " n " theninput " name of boot program " ;b$
HM  1001 print " Boot:  " ;b$:print " Loads:  " c$:print " OK? "
ON  1002 geta$:ifa$<> " y " anda$<> " n " then1002
HH  1003 ifa$ =  " n " thenrun
MN  1007 printchr$(147); "  Creating boot program.  BOOTMAKER II "
DK  1008 print "  Transactor Magazine 1985    Jeff Goebel "
EH  1010 print " sPqqqq 10 rem  boot by bootmaker ii "
GC  1015 print " 20 rem  transactor magazine "
OB  1020 print " 30 rem   jeff goebel – 1984
KN  1030 print " 40 poke53280, " a " :poke53281, " b " :? " chr$(34) " Sn " ;
          chr$(34); " :poke646, " ;c
OH  1040 print " 50 ? " chr$(34);s1$
CD  1050 print " 55 ? " chr$(34);s2$;chr$(34); " :poke646, " b "
BA  1060 print " 60 ? " chr$(34) " qload " chr$(34) " chr$(34) " chr$(34);
          c$;chr$(34) " chr$(34) " ;
AK  1061 printchr$(34); " ,8,1 " chr$(34)
OE  1070 print " 70 ? " chr$(34); " qqqq " ;ac$
BL  1080 print " 80 poke198,2:poke631,13:poke632,13
GH  1090 print " 90 ? " chr$(34); " sq " ;chr$(34); " :new "
GJ  1100 print " esave " chr$(34); " @0: " b$;chr$(34); " ,8 "
IM  1200 poke198,10:fort = 631to641:poket,13:next:print " s " :new
```

Datapoke Aid

Daniel P. Chernoff
Portland, OR

The Easy Way to Enter BASIC Loader Programs

If you're a hunt-and-peck typist like myself you'll appreciate this handy utility to facilitate the typing in of machine language BASIC loader programs.

As you know, these programs consist of a series of DATA statements which, when the program is run, are POKEd into memory locations to create a machine language program that can then be accessed with a SYS command. Usually a checksum is provided to check the total of the numerical data entered as a method of spotting an entry error.

Datapoke Aid is a BASIC program using Dynamic Keyboard, a program line eraser to remove it after it has done its job, a numeric keyboard and error-trapping routines to create in a fast manner the lines of DATA statements for the loader program.

The program resides at lines 59999 and above to ensure that it doesn't interfere with the program lines to be entered. When first run the program prompts for the following inputs:

(1) Right or Left-hand keypad (Being a rightie I assume, but don't know, that a pad on the left side of the keyboard would be a convenience for our sinister friends);

(2) The starting program line for the data statements to begin at;

(3) The number of data statements per program line (usually an even number in the range 6–20); and the

(4) Checksum, if any (This does not check each data statement line but only the grand total – most long loader programs provide one).

Once this information is entered it need not be entered again, even if the typing in of the program is interrupted. In fact, because each data statement line entered becomes a program line, no work is lost if the program breaks or is otherwise halted. (Even in the unlikely event of a crash (keyboard freeze-up) a reset, if you have that facility, and an UNNEW program will restore all your work. The program can be saved as a BASIC program at any point following the entry of a line and then reloaded at a later time to continue the entering of subsequent data statement lines.

The key to the program is an error-trapping routine that, in conjunction with the numeric keypad, allows only numerics to be entered and only numbers between 0 and 255. After each number is inputted from the keyboard a comma is inserted by pressing the spacebar as the program checks to see that the number lies in the prescribed range. (The 'DEL' key permits corrections to be made beforehand.) A valid number yields a 'right' ding tone and the program proceeds to the next data item, whereas an invalid number entry provokes a 'wrong' buzzer and blanks the item for re-entry.

After each line of data statements is entered the program then jumps to the Dynamic Keyboard routine at 60230–60240 which enters the line into the program, updates the running sum of the data items and resets for entry of the next line. The program entry may be halted at the end of the data entry or at any intermediate point (at the end of a data statement line) by pressing the F1 function key.

If entry is not completed when halted then the entire program, consisting of the lines entered thus far and the Datapoke Aid utility, can be saved to disk. Conversely, when all entry is finished then, in response to the prompt, the utility can be deleted by the routine at 59999 (A useful little 'Electric Eraser' you can find use for elsewhere, it essentially tells the computer that the top of BASIC memory is below the portion at lines 59999 onwards where this utility program resides.) However, before deleting the utility, the program forces you to save the file, just in case.

After the utility has been deleted, the remaining BASIC loader program can be edited or augmented as any program. If the checksum is wrong after your entry (quite likely, and the buzzer will tell you so), the program can be proofread against the original and errors corrected.

Note that once the program is run, it modifies itself by adding lines 60020 and 60030 to intialize important variables and skip the initial start-up questions. To re-run the program for a 'cold start', change these lines back to the way they appear in the listing.

I find the real value of this program, for a non-typist, is the ability it provides to enter the program lines completely with one hand while the other traces along the program listing being copied, and without the necessity of either looking at the monitor screen to verify where you are or moving your hand from a single position poised above the keypad.

```
GE  59999 a=peek(61)+256*peek(62)+3:poke786,int(a/256):poke785,a-256*peek(786)
DH  60000 rem: " DATAPOKE AID       10/24/84 (c)1984  Daniel P. Chernoff
EF  60002 ifer=0then60020
II  60004 printchr$(31)"sA"chr$(34)"@0:"f$chr$(34)",8
PP  60005 print" qqqq vE"chr$(34)f$chr$(34)",8e
CM  60006 poke198,4:poke631,19:form=1to3:poke631+m,13:next
GG  60008 iferthenpokea-2,0:pokea-1,0:poke45,peek(785):poke46,peek(786):clr:run
IO  60020 :
CC  60030 poke 2,0
GJ  60040 poke53280,6:poke53281,6:printchr$(14)
JI  60050 print" heS "tab(14)" DataPoke Aid ":printtab(14)"
HO  60060 print" qq Left r l R or right r r R hand keypad: r]] ";:inputk$:print" Q "
MH  60070 print" q Starting line of data: ";:inputb:print" Q "tab(22)"    "
ND  60080 print" q Program line increments: 10]] ";:inputin:print" Q "tab(24)"    "
EE  60090 print" q No. of data items per line: 8]] ";:inputd:print" Q "tab(27)"    "
OP  60100 print" q Checksum (if any): ";:inputck:print" Q "tab(18)"    ":goto60710
GE  60110 poke214,21:print:print" her F1-end [data sum = "cs" ]]checksum is "ck" R ";:return
FL  60120 c=b+(in*peek(2)):print" S "chr$(14):poke53281,6:poke53280,6:gosub60320
JI  60130 dimȧ(30):k=1
HD  60140 print"s";:forj=1to7:print"[40 spaces]";:next
FJ  60150 gosub60110
GB  60160 l=l+1:gosub60750:print" se "c"data  ";:fori=0tod-1
NA  60170 gosub60480:ifa$=""then60170
MM  60180 ifa$=chr$(20)andc$<>""thenc$=left$(c$,len(c$)-1):print" ]  ] ";:goto60170
OJ  60190 ifa$=" E "then60530
NN  60200 gosub60790
DO  60210 ifa$=chr$(13)thena(k)=val(c$):cs=cs+a(k):c$="":k=k+1
MG  60220 ifa$=chr$(13)thenifi<d-1thenprint",";:gosub60225:next
KN  60222 goto60230
AG  60225 s=54272:pokes+24,15:pokes+1,110
PP  60227 pokes+5,9:pokes+6,9:pokes+4,17:pokes+4,16:return
NP  60230 ifa$=chr$(13)thenpoke2,peek(2)+1:print:printchr$(31)"60020cs="cs:print"rU
AN  60240 ifa$=chr$(13)thenpoke198,4:poke631,19:form=0to3:poke632+m,13:next:end
AA  60250 ifasc(a$)<45orasc(a$)>57thengosub60440:a$="":goto60170
AB  60260 printa$;:c$=c$+a$:a(k)=val(c$)
JL  60270 ifa(k)>256thengosub60440:print"  r Error! - reenter R ";:gosub60290:next
LB  60280 goto60170
GN  60290 forj=1to300:next:ifa(k)>1000thenl$=" ] "
EA  60300 print" ]]]]]]]]]] "+l$+"[22 spaces]]]]]]]]]]]] ";
CP  60310 c$="":l$="":a(k)=0:i=i-1:return
PN  60320 poke214,6:print:printspc(16)" e Keypad ":printspc(16)"
LJ  60330 ifk$="l"then60390
CN  60340 printspc(14)" < u R  r i R  r o R ":printspc(11)"      7     8     9
LK  60350 printspc(14)" < j R  r k R  r l R ":printspc(11)"     4     5     6
DN  60360 printspc(10)" < n R  r m R  r , R  r . R ":printspc(11)"0     1     2     3
AA  60370 printspc(14)" < space bar R ":printspc(15)", [comma]
IP  60380 return
NP  60390 printspc(14)" < e R  r r R  r t R ":printspc(11)"      7     8     9
CL  60400 printspc(14)" < d R  r f R  r g R ":printspc(11)"      4     5     6
ML  60410 printspc(10)" < z R  r x R  r c R  r v R ":printspc(11)"0     1     2     3
CD  60420 printspc(14)" < space bar R ":printspc(15)", [comma]
KC  60430 return
IE  60440 rem: 'wrong buzzer'
DJ  60450 poke54296,15:poke54273,5
CI  60460 poke54277,0:poke54278,240:poke54276,33:fort=1to500:next
HG  60465 poke54276,32:goto60470
JM  60470 fort=54272to54296:poket,0:next:return
```

```
FG  60480 rem:flashes char cursor to screen awaiting a get. change char as desired
OO  60490 print"  ] ";
BL  60500 poke204,0:geta$:ifa$=""then60500
AG  60510 poke204,1:print"  ] ";
EI  60520 return
BN  60530 ifcs<>ckthenprint" S ":gosub60440:gosub60110:print" sq "tab(8)"rError in data statementsR
IP  60540 f1$="testfile
JH  60550 print"qq   Do you wish to erase the Datapoke
HA  60560 print"   program lines (64000-) prior to
LF  60570 print"   saving the data statements you have":print"   just entered? y]";
EA  60580 gosub60500:q$=a$
OJ  60590 ifq$="n"thenprint:printtab(15)"Q . rNoR.":gosub60750:gosub60620:goto60650
ID  60600 ifq$="y"orq$=chr$(13)orq$="   "thener=1:print:gosub60620:goto59999
GI  60610 print"QQ":goto60570
PK  60620 print"qq   Enter filename: "f1$"Q";:print:printspc(18);
NB  60630 inputf$:iff$=""thenprint"QQQ";:goto60620
MP  60640 return
CL  60650 printchr$(31)"S60540f1$="chr$(34)f$
LA  60655 print"qq60025pO2,"(c-b)/in:print"f$="chr$(34)f$
CC  60660 print"qqgO60670":poke198,7:poke631,19:form=1to6:poke631+m,13:next:end
GG  60670 print"eSqq":f$="@0:"+f$:savef$,8
OK  60680 print:print:verifyf$,8
HJ  60690 ifst<>64thenprint"q   Resave y]]";:inputr$:ifr$="y"then60670
MB  60700 end
CF  60710 rem put variables in line 1 and return to 120
HP  60720 printchr$(31)"S60030b="b":in="in":d="d":ck="ck":k$="chr$(34)k$chr$(34);
IJ  60725 print":gO60120":print"rU
PH  60725 print"gO60120":print"rU
NN  60730 gosub60320
FO  60740 poke198,3:poke631,19:poke632,13:poke633,13:end
DB  60750 rem: 'right ding'
DA  60760 tn=54273:poketn+23,15:poketn,40
IO  60770 poketn+4,9:poketn+5,0:poketn+3,17:fort=1to500:next:poketn+3,16
HK  60780 poketn,0:poketn+3,0:poketn+4,0:poketn+5,0:poketn+23,0:return
DH  60790 ifk$="l"then60810
PH  60800 kp=-(a$="m")-2*(a$=",")-3*(a$=".")-4*(a$="j")-5*(a$="k")-6*(a$="l"):goto60820
DN  60810 kp=-(a$="x")-2*(a$="c")-3*(a$="v")-4*(a$="d")-5*(a$="f")-6*(a$="g"):goto60840
IG  60820 kp=kp-7*(a$="u")-8*(a$="i")-9*(a$="o"):ifa$="n"ora$="h"thena$="0"
EE  60830 goto60850
GI  60840 kp=kp-7*(a$="e")-8*(a$="r")-9*(a$="t"):ifa$="z"ora$="s"thena$="0"
LK  60850 ifa$=chr$(32)thena$=chr$(13)
KO  60860 ifkp>0thena$=chr$(48+kp)
CG  60870 kp=0:return
```

Load & Run

Thomas Henry
Mankato, MN

Start Up Machine Language Programs Just Like BASIC!

Recently I wrote a program for the CBM–8032 in pure machine language. In order to run the program I had to first load it and then type SYS25978. Needless to say, this ghastly number is hardly my favorite, and as a consequence, every time I wanted to run the program I had to look back to my notes to find the proper SYS address. And things get even worse on the VIC–20 or Commodore 64. Consider that you not only have to recall the proper SYS address, but you must also remember to load the program with a LOAD"filename",8,**1**. Surely there must be an easier way to get machine language programs up and running on a computer without all the hassles of special load instructions and forgettable SYS addresses!

Introducing The "LOAD & RUN" Utility

Good news; there IS a better way! Described in this article is a utility, called "LOAD & RUN", which you can apply to any programs of your own. This utility transforms a machine language program into a form which can be LOADed and RUN just like a BASIC program. No special load instructions are needed, and there's no need to concern yourself with the SYS address. If you deal with machine language as much as I do, I'm sure you will find this to be a real timesaver as well as brainsaver. Best of all, "LOAD & RUN" can be made to work on any model of PET/CBM, the VIC–20 or the Commodore 64 and any machine language programs you have can be easily retrofitted to include this new feature.

Let's get a general idea of the problem and how to solve it. Usually large machine language programs sit high up in memory far away from the start of BASIC. Somehow we have to load the program like BASIC and then transfer it up to its proper place. Finally some sort of automatic SYS should occur which will start execution of the code. Hence, we will transform the code so that it sits low in memory (where it can be loaded like BASIC), and include a small machine program which transfers the code back up to where it belongs and then executes it.

How "LOAD & RUN" Works

The assembler listing shows the program which will accomplish all of these tasks. The listing looks complex, but this is because many comments have been included. (Skilled machine language practitioners will find everything they need to know in the listing alone and may skip ahead, but beginners should keep reading here for more details.) Actually the code is a mere 42 bytes long! Since the program is so short and yet useful, machine language tyros will find this to be a great learning experience as well.

Let's get an overview of the program, leaving the listing to supply the details. In lines 00015 through 00040 you will find the equates for the various types of Commodore computers. Simply pick the set of equates which apply to your machine. (For the purpose of example, the code was assembled using the PET/CBM 4.0 ROM equates). Note that the actual program starts at address $0401 which is where a BASIC line would start in memory. What we do is create the BASIC line:

```
10 SYS1037
```

To effect this we need a pair of link bytes, a pair of bytes for the line number, the token byte for SYS, then the ASCII representation for "1037", followed by three zero bytes. When this BASIC line is run, a SYS1037 occurs, which transfers control to address 1037 ($040D in hexadecimal). $040D is the address of the machine language module entitled "INIT" (line numbers 00061 through 00074 in the assembler listing). By the way, the number 1037 will work for VIC–20's with the 3K expander added but will have to be changed to 4109 for the stock VIC–20 and 4621 for VIC's with 8K or more of extra memory. Use the number 2061 for the Commodore 64. These numbers differ, of course, since each of the three machines has a different address for the start of BASIC.

The module entitled "INIT" contains all of the code needed to transfer a program up to its proper place. Essentially you set a pointer to the start of the code to be moved, another to the end address plus one, and still another to the end address plus one of the code's final resting place. Next you call a subroutine from your computer's ROM which does the actual moving of the bytes. Finally you jump to the start of the program once it is in place.

With all of these details, let's not lose sight of the original problem. Notice how we have a BASIC line followed by the "INIT" module, which is itself followed by the program to be moved. That is, all of the code is now contiguous and at the start of BASIC. Thus it is possible to load the entire lot as a single BASIC program (no LOAD "filename",8,1 is needed). If this still seems unclear to you, refer to the assembler listing which is heavily annotated and provides many details.

Using "LOAD & RUN" in your own programs

Well, enough theory; let's see how to actually use this little marvel! Here's a checklist of what's needed to retrofit a program:

(1) Obviously we need a machine language program to change. This can be a game, utility, word processor or whatever program you wish to work over. It can be a commercial program or one that you have entered yourself.

(2) Next you will need some sort of extended machine language monitor. If you own a PET/CBM, any of the public domain monitors like Supermon or Micromon will do the trick. (Versions of Supermon for all machines can be found on any Transactor disk.) VIC-20 users also have access to these two monitors as well as commercial equivalents like VICMON or HESMON. Commodore 64 users can use the public domain Supermon or the commercial HESMON package. Whatever monitor you use, it makes no difference, as long as you have access to the "A", "M" and "T" commands (assemble, memory dump and transfer, respectively).

To modify your machine language program so that you can LOAD and RUN it like BASIC, simply follow these steps. For the sake of discussion, it is assumed that you will be doing this on a PET/CBM with 4.0 ROM's. The steps are similar for other Commodore computers and any variations in the procedure will be mentioned later on.

(1) First off, load in the extended machine language monitor of your choice. You may use a cartridge, tape or disk-based system, as long as it supports the "A", "M" and "T" commands.

(2) Now assemble the code of "LOAD & RUN" using the "A" and "M" commands of your monitor. Start at address $0401 and continue through to address $042A. At this point you still won't know the addresses of "MLE" or "NEWADD", (in the instructions at line numbers 00065, 00067, 00069 and 00071) so just put in dummy bytes for the moment. Likewise, the address of "RUN" in line number 00074 is still unknown, so again just put in two bytes to hold the place. We will come back and change these instructions in a later step.

(3) Load in the machine language program to be changed. (Make sure that it doesn't overwrite your monitor!) Make a note of its start address and end address. The end address plus one is "NEWADD", so go back to line numbers 00069 and 00071 and replace the dummy bytes with the true address. Likewise the start address of your program becomes the address of "RUN", so go back and replace the two dummy bytes with the true numbers. (See step (2), above).

(4) Now using the "T" command, transfer the entire program which you loaded in step (3) so that its first byte falls at address $042B. (This is the first free byte following "LOAD & RUN".) Note the address of the last byte of the transferred code and add one to it. Using this number, go back to "MLE" in line numbers 00065 and 00067 and change the dummy bytes accordingly. (See step (2)).

(5) We're almost done! Now that we know the true address of "MLE", stick that number into locations ($2A), ($2C) and ($2E). Remember, in 6502 machine language, the low order byte comes first, followed by the high order. So, for example, if "MLE" is $3120, you would put the $20 in $2A, $2C and $2E, and the $31 in locations $2B, $2D and $2F.

(6) Now leave the monitor with the "X" command, thus returning to BASIC. Save the program, using an ordinary SAVE to either tape or disk. Since we set the pointers in step (5) to point after all of the machine code, everything will automatically be saved, ready to just LOAD and RUN. Simple, isn't it!

And that's all there is to it. While the instructions may have seemed long-winded, in actuality the process is quite simple. I have modified all of my machine language programs to the "LOAD & RUN" format now, and it has never taken more than two minutes at most to complete the conversion.

A Few Words To VIC-20 And C-64 Users

VIC-20 and Commodore 64 users can apply this same process with just a few modifications. First, be sure to use the proper set of equates for your machine. Next, the locations mentioned in step are ($2D), ($2F) and ($31) for both the VIC-20 and Commodore 64. Finally, recall that C-64 users will start their assembly at $0801 and VIC-20 people at $0401, $1001 or $1201 depending on the amount of extra RAM added to the system. And don't forget to change the ASCII code in line number 00052, as mentioned above.

Since the VIC–20 has so many different memory configurations, be sure to use the correct addresses for the correct amount of memory. Clearly, a "LOAD & RUN" program designed for a stock machine won't work on an 8K machine, etc. If you need a more high powered system which does take into account any extra memory add-ons, check out Jim Butterfield's excellent utility, "Machine Language Auto-Location", (COMMODORE MAGAZINE, June/July 1982, pp. 82–84) and also see my letter to the editor on the same subject (COMMODORE MAGAZINE, March 1983, p. 23).

The LOAD & RUN Generator Program

This is for disk drive users. If you have a few programs you wish to convert and don't want to repeat the above steps over again for each one, or if you're having trouble with the conversion process, try the BASIC program in Listing 2. It will ask for the filename of the program you wish to convert to a "LOAD & RUN" program, the length of the program, and the SYS address. With that, it will generate a program on disk, using the same filename with the characters "LR." appended to the beginning.

If you're not sure of the length of the program you wish to convert, it usually won't hurt to err on the high side. To get an idea of the program's length, look at the number of blocks it occupies in the disk directory and multiply the block count by 254. That should be sufficient in most cases, except where the program lies immediately beneath a sensitive area of memory that must not be destroyed by the LOAD and RUN.

The version of the LOAD and RUN generator listed is for the C64. Make the indicated changes for PETs or VIC–20s.

Final Thoughts: Why Use "LOAD & RUN"?

As mentioned, it only takes a minute or two to convert any machine language program to a "LOAD & RUN" format, but some people may still consider this to be a needless hassle. So why do it? Well, if you're like me and have lots of machine language programs kicking around, you will definitely appreciate not having to remember a myriad different SYS addresses. And the ability to load a program on a VIC–20 or C–64 just like BASIC really simplifies things too.

But perhaps the best advantage is that beginners using your programs won't have to learn any new commands to get a program up and running. If you have children who use the computer they will appreciate this and in general, newcomers to the computer keyboard will teel far less threatened using machine language programs if they can LOAD and RUN them just like BASIC.

Listing 1: Assembler Code for the "LOAD & RUN" conversion (Add line numbers if using the PAL assembler.)

```
;***************************************
;*                                     *
;*             'load & run'            *
;*    a machine language boot-up aid   *
;*                                     *
;*    for the pet/cbm, vic-20 and c-64 *
;*                                     *
;*            thomas henry             *
;*          249 norton street          *
;*          mankato, mn 56001          *
;*                                     *
;***************************************
;
;*** equates for pet/cbm with 2.0 or 4.0 roms ***
;
newend   =   $55        ;new end address + 1.
oldend   =   $57        ;old end address + 1.
oldsta   =   $5c        ;old start address.
basic    =   $0400      ;start of basic.
move     =   $b357      ;block move routine –– 4.0,
;                       ;change to $c2df for 2.0 roms.
;
;
;*** equates for vic-20 ***
;
; newend =   $58        ;new end address + 1.
; oldend =   $5a        ;old end address + 1.
; oldsta =   $5f        ;old start address.
; basic  =   $1000      ;start of basic, stock vic.
; move   =   $c3bf      ;block move routine.
;
;
;*** equates for the commodore 64 ***
;
; newend =   $58        ;new end address + 1.
; oldend =   $5a        ;old end address + 1.
; oldsta =   $5f        ;old start address.
; basic  =   $0800      ;start of basic.
; move   =   $a3bf      ;block move routine.
;
;
;this next block of code creates
;the basic program- 10 sys1037
;
;
* = basic + 1
;
.word link              ;forward link byte.
.word 10                ;line number ten.
.byte $9e               ;token for 'sys'.
.byte '1037'            ;address of 'init' in ascii.
;(.asc '1037' for PAL assembler)
.byte $00 ;end of line byte.
```

```
link        .word $0000       ;end of program mark.
;
;
;now comes the routine to move the machine
;language program into its proper place.
;
;
init        lda  #<mls        ;set pointer to start
            sta  oldsta       ;of code to be moved.
            lda  #>mls
            sta  oldsta+1
            lda  #<mle        ;set pointer to end+1
            sta  oldend       ;of code to be moved.
            lda  #>mle
            sta  oldend+1
            lda  #<newadd     ;set pointer to end+1
            sta  newend       ;of new address for code.
            lda  #>newadd
            sta  newend+1
            jsr  move         ;go move code into place.
            jmp  run          ;go run code.
;
;
;here follows the block of code which is
;to be moved into position. this is the
;portion you supply for your own application.
;
;
mls         =    *            ;'mls' is the address of the
;start of code which is to be
;moved into place.
;
; (block of code goes in here)
;
mle         =    *            ;'mle' is the address of the
;next byte beyond the end of
;the code.
;
newadd      =    $8000
;
;'newadd' is the address of the next byte
;beyond the last byte of the code, after
;it has been moved into its final position.
;$8000 is used here merely as an example.
;
run         =    newadd-mle+mls
;
;'run' is the start address of the
;code after it has been moved into its
;proper place, (usually in high ram).
;
.end
```

Listing 2: BASIC program to create "LOAD & RUN" program on disk
C64 Version; see below for VIC/PET modifications

```
EE  100 rem load&run file creator
IB  110 dim s%(7)
NE  120 input" filename     ";f$
CM  130 input" program length ";pl
KA  140 input" sys address    ";sy
OH  150 open1,8,12,f$+",p,r"
GC  160 get#1,a$,b$: rem start address
FN  170 s1=asc(a$+chr$(0)):s2=asc(b$+chr$(0))
        :sa=s1+256*s2
EE  180 bs=2048: ptrs=43
CD  190 :
MB  200 mls=bs+43  :rem source start
BO  210 me=pl+mls+1 :rem source end
PO  220 na=sa+pl+1  :rem destination end
NM  230 s%(0)=mls and 255
EN  240 s%(1)=mls/256
PL  250 s%(2)=me and 255
HM  260 s%(3)=me/256
FC  270 nl%=na/256
PG  280 s%(4)=na-256*nl%
CE  290 s%(5)=nl%
GM  300 sy%=sy/256
OP  310 s%(6)=sy-256*sy%
DI  320 s%(7)=sy%
OL  330 :
HF  340 data 11, 8, 10, 0, 158, 50, 48, 54, 49, 0
KN  350 data   0,  0, 169,  -1, 133, 95, 169, -1
FI  355 data 133, 96, 169,  -1, 133, 90, 169, -1
FI  360 data 133, 91, 169,  -1, 133, 88, 169, -1
BE  370 data 133, 89,  32, 191, 163, 76,  -1, -1
AP  380 :
LC  390 open 2,8,11,"@0:lr. "+f$+",p,w"
FB  400 c=0:bs=bs+1
GE  410 print#2,chr$(bs and 255)chr$(bs/256);
LG  420 fori=1to42:read a:ifa=-1thena=s%(c):c=c+1
HN  430 print#2,chr$(a);:next
DE  440 fori=0to1:get#1,a$:s=st:print#2,left$(a$
        +chr$(0),1);:i=s:next
PC  450 close1: close2: end
```

Changes to make for PET/CBM BASIC 2.0/4.0 version:

```
180 bs=1024: ptrs=40
340 data  11,  4,  10,  0, 158, 49, 48, 52, 55, 0
350 data   0,  0, 169, -1, 133, 92, 169, -1
355 data 133, 93, 169, -1, 133, 87, 169, -1
360 data 133, 88, 169, -1, 133, 85, 169, -1
370 data 133, 86,  32, 87, 179, 76, -  1, -1
(replace 87, 179 with 223, 194 for BASIC 2.0)
```

Changes to make for VIC-20 (standard configuration)

```
180 bs=4096: ptrs=43
340 data  11,  8, 10, 0, 158, 52, 49, 48, 57, 0
370 data 133, 89, 32, 191, 195, 76, -1, -1
```

Extra EPROM Space for PETs

F. Arthur Cochrane
Jackson, SC

The average PET user probably has some type on ROM or EPROM plugged into the extra sockets inside his PET. This firmware can be commercially bought or programmed by the user. On PETs with 12 inch screens there are only two empty sockets at US11 and US12. When these are full the user can get a multiple socket ROM switcher such as a Socket–Me–Two or Spacemaker II.

On these machines if the user needs some ROM space in addition to the two sockets there is 1 3/4 K space for machine language programming. This extra memory is at addresses $E900 to $EFFF which is not currently used by the PET. The easiest way to use this ROM space is to remove the 2K BASIC ROM at UD7 (the only BASIC ROM on a socket) and replace it with a 4K EPROM (such as a 2532). The 2K BASIC ROM's contents will have to be transferred to the lower 2K of the 4K EPROM and then any machine code the user may wish can be put into the upper 1 3/4 K of the EPROM. The EPROM can then be plugged into the UD7 socket. When the PET addresses the I/O chips in locations $E800–$E8FF the EPROM will not be enabled because Commodore added logic on the newest PETs to not enable the UD7 socket for this address range. The PET thus has the following pattern for the address space $E000 to $EFFF:

$E000 to $E7FF	UD7 socket (BASIC ROM)
$E800 to $E8FF	I/O chips (6520s, 6522, and 6545)
$E900 to $EFFF	UD7 socket (user defined)

This modification will not work however if an expansion is made to the PET that uses the NO ROM pin on the expansion port (pin J4–20) to disable the BASIC ROMs. NO ROM is connected to pin 21 on all the ROM sockets and is normally pulled to plus five volts. Pin 21 on ROMs is a high enable chip select so when NO ROM is pulled to ground (zero volts) the ROM is disabled. However on EPROMs, pin 21 is for programming voltage and for normal operation it must be held at plus five volts. Also note that EPROMs plugged in the two empty sockets (UD11 and UD12) of the PET will be affected in the same way if the NO ROM pin is pulled to ground. The SuperPET probably uses NO ROM to disable BASIC and enable the 6809's operating system.

Small screen PET users will not be able to use this technique on their PETs because if an EPROM were to be plugged in at the $E000 address range it would be enabled in the $E800 to $E8FF address range at the same time as the I/0 chips. The enabling of the EPROM and I/0 chips at the same time would cause false data to be read or written to the I/0 chips. Also small screen PETs do not have the NO ROM pin on their expansion port so pin 21 on all the ROM sockets is always held at plus five volts.

I hope PET users can use this information to expand their PETs to fill their needs for firmware program storage

I have modified the 2K extended monitor Extramon to fit into the $E900 to $EFFF address space of the 8032 and Fat 40. Extramon had to be reduced slightly to fit into 1 3/4 K and the following compromises were made. The Fill command was removed, the Hunt with don't care and Hunt for an ASCII string was removed (Hunt for ASCII characters with the HEX equivalent), the 'J' key (finish execution of a JSR command) was removed from the Walk command, and the repeat key function was removed (this is now normal for cursor keys on the 8032 and Fat 40). All other commands remained the same. This E–ROM.MON source code for the MAE assembler is available.

A Really Cheap Multi–Channel Analogue Input for Micro Computers

Harold Anderson
Oakville, Ont.

Here is a really cheap way to equip your micro computer to read and monitor voltages. I have been using this circuit for several years to make my PET operate as a versatile six channel strip chart recorder (in conjunction with a printer of course).

The only major component of the circuit is a Motorola MC14447 integrated circuit which costs about $4.00 in single quantities. This device has six input channels, effectively infinite input impedance and a typical potential accuracy of 0.05%.

This IC can be interfaced easily to virtually any microcomputer which has a parallel port; even a parallel printer port would do. This article gives the software to run them from the Parallel Users Port of a Commodore PET.

The A to D converter shown in the accompanying circuit diagram is able to measure voltages in the range of 0 to 2.5 volts with an accuracy of 0.5 millivolts. Since the input impedance is effectively infinite, higher voltages can be measured easily by first putting them through a simple voltage divider made from two resistors.

How the MC14447 Should Be Used

The MC14447 is a somewhat unconventional A to D converter in that it depends on the intelligence of the computer to compensate for its own shortcomings. Without a computer capable of simple arithmetic it would be almost useless. The chip really has 8 input channels, but two of these are permanently dedicated to two reference levels. As shown in figure 1, the first reference level is zero volts and is internally connected to channel zero. The second reference level is 2.5 volts (or something close to that, provided that it is stable.) and is normally connected to channel 7. That leaves 6 channels for measuring unknown voltages. When the chip is used, the computer measures all eight channels in quick succession. The answer or count, which the computer gets for each channel, is linearly related to the voltage on each channel. This linear relationship can be expressed by the following formula:

$$CT(n) = Z + S \mathrm{X} V(n)$$

where:

n = the channel number
CT(n) = the answer or count that the computer got for channel n
V(n) = the voltage on channel n
Z = the zero offset (a constant)
S = the slope of the linear relationship (a constant)

Immediately after the the 8 channels have been measured, CT(0), V(0), CT(7) and V(7) are all known. (Remember V(0)=0 and V(7)= the reference voltage). This allows us to solve the two equations in two unknowns to find the value of the constants Z and S. Once Z and S have been calculated, the other six voltages can be found from the values of CT(n). This mathematical procedure sounds more complicated than it is, as the following solution to the problem shows.

$$\begin{aligned} Z &= CT(0) \\ S &= [CT(7)-CT(0)]/[V(7)-V(0)] \\ V(n) &= [CT(n)-Z]/S \end{aligned}$$

As you will see from lines 900, 920 and 940 of the BASIC listing accompanying this article, the value of all of the 6 unknown voltages can be found from a BASIC program containing less than 80 characters.

S and Z should be recalculated each time the device is used since they vary slowly as the temperature of the chip changes. S and Z also are dependent on the particular chip used and the values of the resistors and capacitors used. When the computer recalculates the value of S and Z each time, and then uses the new values to find the voltages, it is in effect compensating for the changes in S and Z. Since changes in S and Z are compensated for, it allows us to use very imprecise components in the circuit. The only things in the circuit which need be precise are the reference voltage and the linearity of the relationship between CT(n) and V(n). Motorola guarantees the non–linearity to be .2% of full scale maximum and .05% typical.

How The MC14447 Works

A functional block diagram of the MC14447 is shown in figure 1. The MC14447 chip is somewhat of a cross between a sample and hold network and a conventional dual slope A to D converter (Dual slope converters are used in most digital voltmeters). It is controlled by 4 digital input lines. Three of these lines are used to select the channel being measured (A0, A1 and A2). The other line, called the ramp start or ramp/sample line, sets the chip in either sampling or in timing mode.

When in use, the computer first selects the channel to be measured using the three channel address lines. The computer then sets the ramp/sample line low, connecting the selected input channel to the ramp capacitor. An offset built into the internal buffer amplifier causes the ramp capacitor to be charged to a voltage slightly higher than the actual input voltage. (The time required to charge the capacitor depends on the size of the ramping capacitor; I used 6.4 ms. The length of time is not critical provided that it is long enough.) Once the ramping capacitor is charged, the computer sets the ramp start line high, breaking the connection between the input voltage and the ramping capacitor. An internal current source causes the capacitor to discharge in a very linear fashion. The higher the initial voltage on the ramping capacitor, the longer it takes to discharge.

The voltage level on this capacitor is monitored by a comparitor. When the capacitor is discharged to about 0.27 volts the output of the comparitor swings low. The computer monitors the output of the comparitor and times the period between the start of the ramp (ramp/sample line set high) and the end of the ramp (comparitor output goes low). The length of this period is the computers initial measure of the voltage on the input channel and is, in fact, the CT(n) referred to in the previous section.

To abbreviate, the sequence to time one channel would be as follows:

1. Select input channel by setting the levels on the channel select lines A0, A1 and A2.
2. Transfer the voltage on the selected channel to the ramping capacitor by setting the ramp/sample line low.
3. Disconnect the capacitor from the input voltage by setting the ramp/sample line high.
4. Time the discharge of the capacitor by measuring the time until the comparitor output line goes low.

Supporting Software

The software which I wrote to operate this chip was written partly in BASIC, and partly in 6502 machine language. I have included the listings for both parts in this article.

The method of timing the ramp, as mentioned in the previous section is a matter of user preference. The method which I used on the PET was probably the simplest and best for most applications.

I wrote a short machine language program to time the ramp. The program runs in a tight loop during the timing operation. During this part of the program the X and Y registers are treated as a 16 bit counter. Each time the program goes through the loop it uses up 15 microseconds, increments the 16 bit binary counter, and checks the value of the comparitor output. When the comparitor output goes low, the 16 bit counter contains a binary number proportional to the length of time taken for the ramp capacitor to discharge.

Since I had to write a machine language program anyway, I also wrote sections to select the eight input channels in sequence and park the resulting counts in RAM for use by the BASIC part of the software. The machine language part also initialized the parallel port so that bit 7 was an input (connected to the comparitor output) and the other bits were outputs, (bits 0, 1 and 2 control channel selection and bit 4 controls the ramp start). This part of the program would have to be rewritten to match the configuration of the type of computer which you are using. The PET parallel user port happens to be a 6522 VIA mapped at addresses $E840 to $E84F.

The BASIC part of the software retrieves the counts from the count table starting at $0340 (put there by the machine language portion of the program) and converts these counts to voltages. The BASIC program then prints the voltages and time on the screen. Lines 480 to 510 calculate the time lost to the PET clock as discussed later. At line 570 the program waits for the operator to hit a key before repeating the program.

Speed

This program times the ramp in 15 microsecond increments, 15 microseconds being the time to get through the timing loop once. In order to measure voltages with a resolution of one part om 5000, you must have a count of 5000 for a the highest input voltage(about 2.5 volts. If each count consumes 15 microseconds then the time for 5000 counts is .075 seconds. Lower input voltages give correspondingly shorter counts. If the average voltage on the input channels is half of the maximum of 2.5 volts then the average count time will be about .0375 seconds and the total time to time all 8 channels will be 8 X .0375, or .3 seconds.

The counts obtained by the machine language program are most easily converted to voltage by using a higher level language, such as BASIC, Pascal, Fortran etc. In my case I used BASIC. It takes the BASIC interpreter in the Commodore PET about .4 seconds to convert the 8 counts to voltages.

Thus the total time to obtain the voltage on all of the channels is about .7 seconds. This is not exactly blistering speed, but it is plenty fast enough to keep up with most laboratory experiments and industrial data logging applications. It will take longer to print the the voltages than it does to measure them.

Much greater speed but poorer resolution can be obtained by reducing the size of the ramping capacitor. This allows the ramping capacitor to charge and discharge faster.

If some of the channels are not used, time can be saved by not performing the counts and converting the counts to voltages only on the channels that are used.

Notes and Suggestions

RE: "Soft" Real Time Clocks

You will notice from the machine language listing included in this article, that the interrupt is disabled during the machine language part of program. This is done to ensure that the processor will not be pulled off the timing function by an interrupt signal while it is timing the discharge ramp. On the PET, a time-keeping routine or "software clock" is driven by the interrupt signal. Disabling the interrupt signal effectively stalls the clock during the timing operation. In order to correct the clock for the time lost during the timing operation, I wrote the BASIC program to keep a running total of the time lost during the machine language part of the program. (Each single count takes 15 microseconds and each sample time is 6.4 milliseconds.) This totalized correction is added to the apparent time to get the real time.

RE: Negative Voltages

This chip is basically designed to measure positive voltage only. You will in practice get meaningfull answers for slightly negative voltages. Just how far negative you can go depends on the particular IC you have, but most will measure -.05 volts. If the input voltage gets more negative than that you will have trouble with the power supply current (as mentioned later) and the ramping capacitor voltage never getting high enough to switch the comparitor.

RE: Over Voltages

The MC14447 is a CMOS IC and like all CMOS ICs it objects strongly to having its inputs driven appreciably above the voltage of the positive power supply or below the voltage of the negative power supply. Over and under voltages activate parasitic transistors within the chip causing high power supply currents and some-times logic mistakes. If you expect to have over or under voltages we suggest that you put a

100k resistor in series with the input of each channel (as has been done in the accompanying circuit diagram).

RE: Power Supply "Hash"

If you decide to build one of these A to D converters using the + 5 volt power supply from your computer be sure that you filter all of the "hash" out before applying it to the positive power supply pin of the MC14447. Transient spikes will cause problems with the comparitor and buffer amplifier inside the chip, leading to serious inaccuracies.

RE: References

In the circuit diagram accompanying this article a 78L05 voltage regulator is used for both a power supply and the reference supply. These regulators are quite stable enough for most applications. If you are fanatical about stability, use an LM136 (a 2.5V precision reference) in place of R3.

RE: Input Variations

The voltage which this device actually measures is the voltage on the input at the instant the ramp start line goes high. If the voltage that you are measuring is noisy or has ripple on it and you wish to measure the DC component only, you should put a resistor and capacitor filter on the input to filter out the AC component. This is what I did in the circuit shown as figure 2. The resistor can be the same one that you may want to use to prevent overvoltage inputs.

RE: Common Ground

Notice that this circuit has a common ground with the computer. As a result one side of all the voltages which you measure will also be connected to the common of the computer. It is probable that sooner or later you will make a connection mistake and wire the common of the A to D converter to some voltage signal with enough current capacity to destroy the printed circuits in your computer. To prevent this I have fused the ground link. Now the fuse will melt instead of the ground tracks on your computer circuits. I would suggest a 1/2 amp fast blow fuse.

If this common ground is an intolerable situation, I would suggest that you put opto isolators in the 5 signal lines that run between the MC14447 and the computer. This will allow you to "float" the A to D converter. Since none of these are high speed signals, the sluggishness of the opto isolators will not be a problem.

Opto isolation opens up other improvement possibilities. First you can run the MC14447 with a 15 volt power supply. This allows an input range of 0 to 12 volts. A further extension allows the measurement of both positive and negative voltages. Use a 15 volt power supply and a 6 volt reference for the MC14447. If the reference input to the MC14447 is called common, the negative power supply pin will be at -6 volts. The MC14447 can now except inputs running from -6 volts to +6 volts. Use exactly the same software as I have shown except use the channel 7 input for the zero reference and channel 0 for the reference voltage input. In this case the reference is -6 volts.

RE: The Program Accompanying This Article

The program accompanying this article merely measures the voltage on all channels and then prints the results and time on the screen. In practical applications you would probably wish to have the computer do something useful with the voltages. In this case you would incorporate the measuring program as a subroutine in a larger program.

I have located the machine language portion at $6000 and poked the appropriate locations in the PET operating system to prevent the Basic interpreter from overwriting the machine code. Similar things would have to be done on other types of computers.

If you plan to use this converter regularly you might consider putting the machine language portion of the program into an EPROM.

RE: The Circuit Accompanying This Article

The circuit diagram (figure 2) accompanying this article is more sophisticated than really necessary. You could eliminate the pilot light, fuse, and input filters. You could also use a 9V radio battery to supply the voltage regulator instead of the transformer driven power supply. If you are planning to use the circuit as a real tool, however, I would suggest that you include all of these features. The circuit as drawn is easy to use and relatively fool-proof.

RE: Commodore 64

If you are going to use one of these on a Commodore 64 don't forget that the video chip is constantly interrupting the 6510 processor in some modes of operation. This will lead to timing inaccuracies unless corrective steps are taken. *(you can turn off the video chip by clearing bit 4 of location 53265. -T.Ed)*

I invite anyone to write me regarding this or other electronic projects

Harold Anderson Electronics Ltd.,
2261 Bethnal Green Road,
Oakville, Ont.,
L6J 5J8.
Phone (416) 844-0632

Analogue to Digital Circuit Programs

```
IB   100 goto 270
KD   110 program name = analogue 12
CO   120 analogue input demonstration for commodore
         pet computers
LL   130 use motorola mc14447 chip
BM   140 complete sample and correction time for all
         8 channels = .7 sec
FN   150 routine uses basic and machine code
IG   160 machine code does timing, etc.
FK   170 basic does conversion of counts to voltages
NB   180 on commodore pet ti = time since power-up
         multiplied by 60
BF   190 ti is used as a clock
FD   200 written by :
GE   210 :
NG   220 " Harold Anderson Electronics Ltd.
NB   230 " 2261 Bethnal Green Rd.,
OP   240 " Oakville, Ontario, Canada,
KE   250 " L6J-5J8,    (416) 844-0632
IH   260 :
NL   270 gosub 600: rem poke machine code into ram
PK   280 poke 53,96: rem top of ram pointer for basic set to
         $6000 to protect
BF   290 tc = -ti/60: rem correction for time at program start
AK   300 vr = 2.529: rem vr = reference voltage
NO   310 dim v(8),ct(8),c(8): rem v(x) = volts ct(x) = count ,
         c(x) = count-zero offset
```

```
EF  320 pl = 3*256 + 4*16: ph = pl + 1: rem pl = start
        address of count table
OL  330 :
KH  340 rem = = go to machine language for counts = =
LD  350 sys 6*4096: rem routine to time ramp for all
        eight voltage channels
MN  360 :
GM  370 rem = = transfer counts to basic variable ct(x) = =
IB  380 for x = 0 to 7: ct(x) = 256*peek(ph + 2*x)
         + peek(pl + 2*x): next x: rem get counts
KP  390 :
CM  400 rem = = convert counts to voltages = =
KF  410 z = ct(0): for x = 0 to 7: c(x) = ct(x)-z: next x
        : rem remove zero offset
CH  420 s = c(7)/vr: rem slope
NH  430 for x = 0 to 7: v(x) = c(x)/s: next x
MC  440 :
CP  450 rem = = print voltages = =
PH  460 for x = 0 to 7: print "#";x;"  ";v(x);"volts" : next x
KE  470 :
EE  480 rem == calculate time lost while interrupt
        mask set ==
GP  490 for x = 0 to 7: tl = tl + .0064 + ct(x)*15e-6: next x
        : rem running ttl of time lost
ND  500 rem .0064 = time to sample
JA  510 rem ct(x)*15e-6 = time to time ramp
MH  520 :
DG  530 rem = = print time = =
PN  540 t = ti/60 + tl-tc: print "time in seconds = ";t
KJ  550 :
OH  560 rem = = get signal for new sample = =
GO  570 get c$: if c$ = "" then 570: rem hit any key
DP  580 goto 350
CM  590 :
OJ  600 for j = 24576 to 24668: read x: poke j,x: next: return
GN  610 :
IH  620 data 120, 169,   0, 133, 188, 169, 127, 141
LI  630 data  67, 232, 173,  79, 232,  41, 240,   5
HF  640 data 188, 141,  79, 232, 173,  79, 232,  41
PJ  650 data 239, 141,  79, 232, 160,   0, 162,   5
CB  660 data 136, 208, 253, 202, 208, 250,   9,  16
HG  670 data 141,  79, 232, 169,   0, 170, 168, 169
JM  680 data 128, 234, 234,  45,  79, 232, 240,   7
IK  690 data 232, 208, 246, 200, 208, 245,   0, 134
MA  700 data 186, 132, 187, 165, 188,  24,  10, 170
IH  710 data 165, 186, 157,  64,   3, 165, 187, 232
FF  720 data 157,  64,   3, 230, 188, 165, 188, 201
GF  730 data   8, 208, 175,  88,  96
```

```
CM  100 ;program to operate mc14447 a/d converter
FD  110 ;interfaced to commodore pet
LE  120 ;(6502 machine language)
IP  130 ;
LH  140 ;— assign labels —
IF  150 * = $ba
MB  160 count *  =   *      ;temporary count storage, 2 bytes
AH  170 *        =   $bc
JP  180 chan *   =   *           ;channel counter, 1 byte
OK  190 *        =   $0340
OG  200 park *   =   *           ;count storage, 16 bytes
JD  210 *        =   $e84f
PA  220 op *     =   *           ;input – output port
BA  230 *        =   $e843
GN  240 ddr *    =   *           ;data direction reg for port
AH  250 ;
IO  260 *        =   $6000
```

```
GJ  270 ;— entry point from basic —
AA  280 enter     sei                 ;block interference in timing
FE  290           lda  #$00
OP  300           sta  chan           ;set channel counter to 0
IC  310 ;— program parallel port —
EL  320 ;bits 0 to 6 output, bit 7 input
KN  330           lda  #$7f
ML  340           sta  ddr
FK  350 ;— restart point for channel measurements —
EN  360 measurelda  op                ;set channel address bits
PG  370           and  #$f0
PF  380           ora  chan
NJ  390           sta  op             ;output port
KA  400 ;— sample output —
DI  410 load      lda  op
FA  420           and  #%11101111  ;set ramp start low
DE  430           sta  op
IO  440           ldy  #$00           ;6.4 ms delay
FF  450           ldx  #$05
BF  460 loop1     dey                 ;start of loops
BE  470           bne  loop1          ;end 5 microsec loop
HJ  480           dex
II  490           bne  loop1          ;end 1280 microsec loop
JI  500           ora  #%00010000
NF  510           sta  op             ;set ramp start high
IA  520 ;— time ramp —
KI  530           lda  #$00           ;set x & y to zero
MM  540           tax                ;x & y used as 16 bit counter
FP  550           tay
IM  560           lda  #%10000000
BP  570 ;15 microsecond counting loop
FK  580 lloop     nop               ;two nop compensate for dif in
AN  590           nop                 ;in carry and no carry times
EO  600 sloop     and  op             ;check comparitor output
BO  610           beq  storec
IE  620           inx
FF  630           bne  lloop
AG  640           iny
AH  650           bne  sloop
KN  660           brk                 ;trouble, count too big
JC  670 ;— store count —
LK  680 storec    stx  count
LA  690           sty  count + 1
IJ  700 store     lda  chan           ;set x = twice channel
MD  710           clc
LH  720           asl
FK  730           tax
GE  740           lda  count          ;get low byte
AF  750           sta  park,x         ;park low byte
DD  760           lda  count + 1      ;get high byte
CA  770           inx                 ;park high byte
CI  780           sta  park,x
DN  790           inc  chan           ;select a new input channel
GF  800           lda  chan
MM  810           cmp #$08            ;check if all channels done
NI  820           bne  measure        ;do next channel
KB  830           cli                 ;clear interrupt mask
JD  840           rts                 ;exit to basic
IM  850 ;
ID  860 .end
```

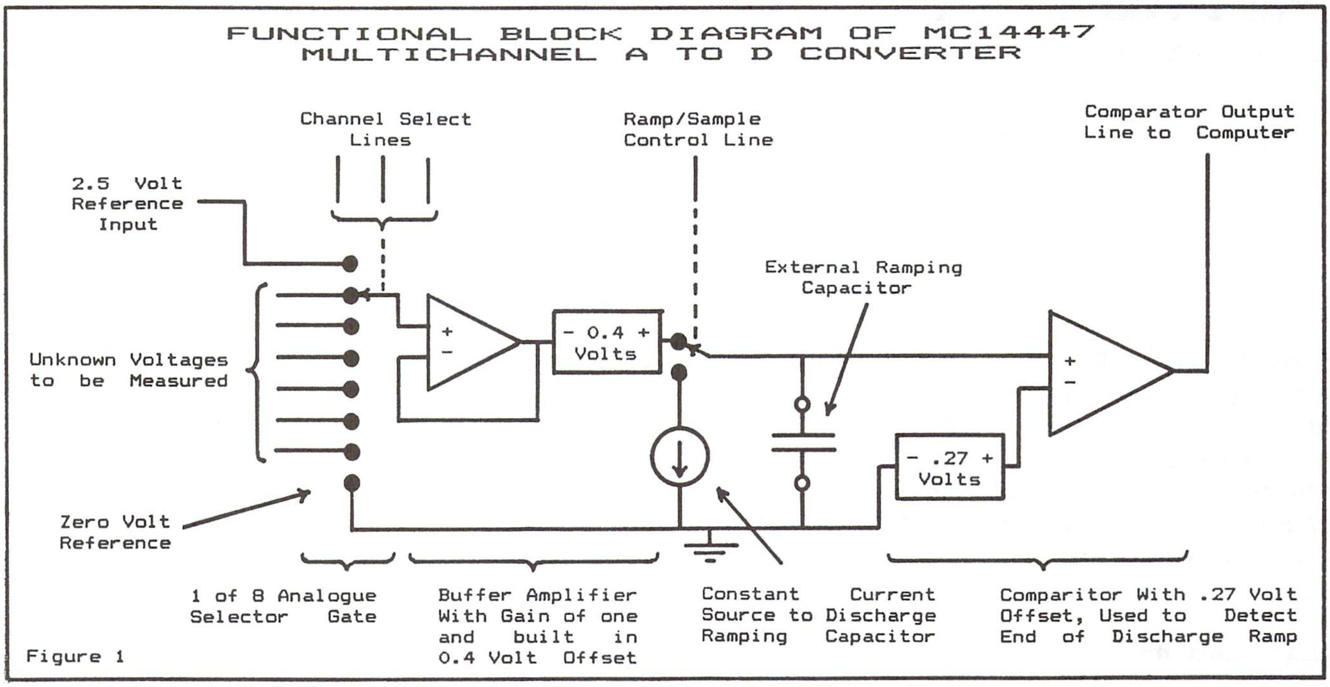

FUNCTIONAL BLOCK DIAGRAM OF MC14447
MULTICHANNEL A TO D CONVERTER
Channel Select Lines
Ramp/Sample Control Line
Comparator Output Line to Computer
2.5 Volt Reference Input
External Ramping Capacitor
Unknown Voltages to be Measured
- 0.4 + Volts
- .27 + Volts
+
-
Zero Volt Reference
1 of 8 Analogue Selector Gate
Buffer Amplifier With Gain of one and built in 0.4 Volt Offset
Constant Current Source to Discharge Ramping Capacitor
Comparitor With .27 Volt Offset, Used to Detect End of Discharge Ramp
Figure 1

Figure 1.

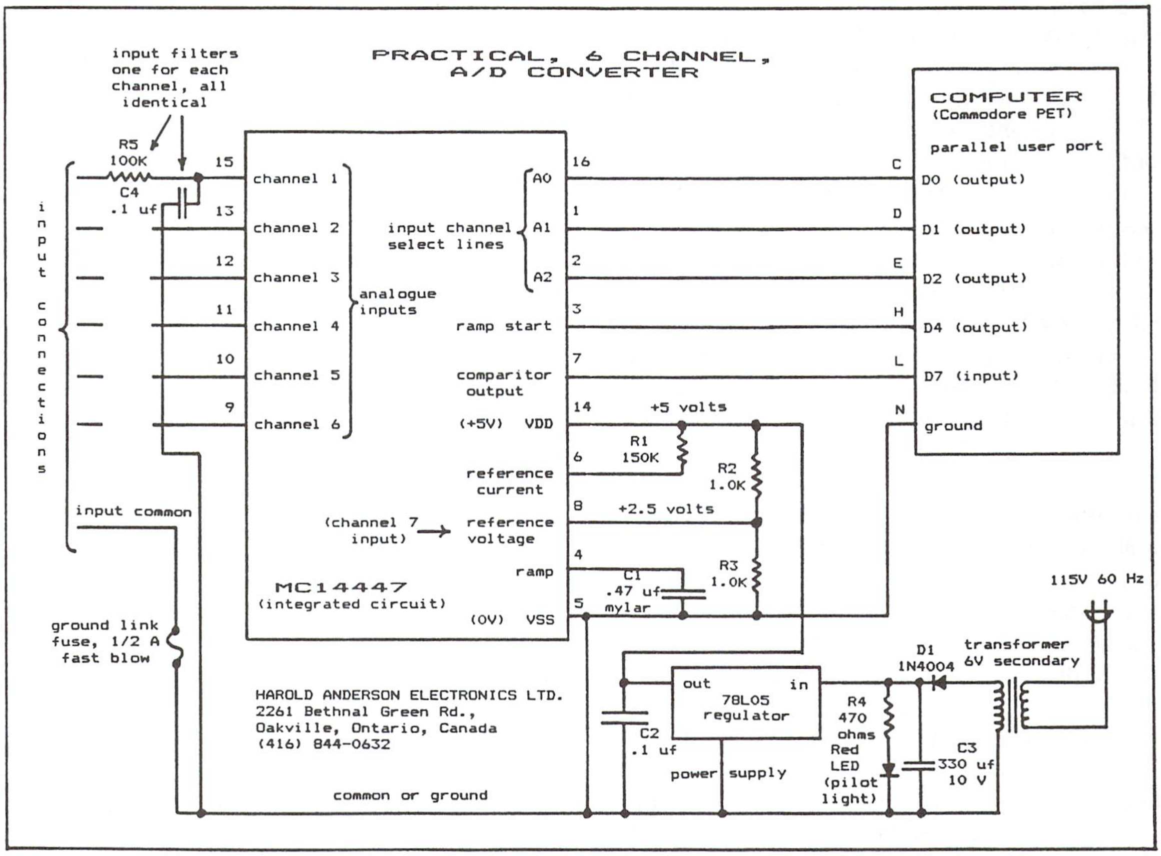


Figure 2.

The Plus 4 – A Quick Overview

Richard Evers, Editor

The Plus 4, a very peculiar creation. For many the very thought of it conjures up images of strange, slimy things oozing out from under rocks. For others, myself included, it's just right, a product of thought and creativity that does not deserve a rapid death as some have forecasted. By reading through many of the articles written about the new system, a few reasonable complaints do surface. But on the whole, capital punishment is not the answer.

Valid Complaints :

The Plus 4's built in user software isn't too terrific. The word processor is barely that, the data base defiles the name, and the spreadsheet has little spread. If you were to buy the computer for the sole purpose of relying on this software, then you may have bought a very attractive paperweight. But if you bought the computer because it is a computer, then the software, if ever used, would be a handy bonus. The wordprocessor is good for quick little notes and simple documents, the data base is good for fairly basic to intermediate file keeping activities, and the spread sheet would stand the test of home and school use. Each package is well written, taking into consideration the limitation of trying to make them all work within the confines of each other. Running two packages in tandem is possible with this system, a trick yet to be performed by many other packages for the Commodore machines. As a final note on this 'complaint', think of the software as an almost free bonus, and accept its limitations. The computer more than makes up for the shortcomings of the software.

The cassette port is not compatible with any other Commodore cassette unit, and the cassette unit that does work with it is SLOWER than the others. Although hard to believe, it's true. Commodore cassette drives have always been slow, but it's just possible the new drives have been made slower for greater accuracy. Regardless, the new system requires a whole new breed of cassette tapes. To offset the bad taste this leaves, a good note: the 1541 serial drive works fine with the unit.

The joystick ports are not compatible with any other joystick in the galaxy, which is a pretty rude turn of events. With luck, somebody will invent an adapter for the machine to allow the use of normal joysticks, thus not forcing you to buy the new ones. If the Commodore joysticks available for the Plus 4 are as good as the ones they have released for the Vic/C64, we had all better hope that an adapter is invented soon.

And Now, The Good Points :

In short, 64k RAM with over 60k available for BASIC programming. A super improved BASIC command set that makes graphics, sound, string manipulation, and disk access more pleasant. The sound is not as good as the 64's, but can be programmed by the novice from within BASIC. Graphics capabilities are as good as the 64's, and can also be easily controlled from within BASIC. The MID$ function has also been improved to allow replacement of specific sections of a string, without major string manipulation. The following is now totally acceptable:

```
mid$(a$,5,7) = "Amazing"
```

Terrific. And considering that the Plus 4 doesn't have slow garbage collection to worry about, string work is just fine.

The units have a built in machine language monitor, Tedmon to be exact, and it's pretty good. You can assemble code in RAM, disassemble RAM or ROM, display and modify memory at will, walk routines as they would normally execute, transfer code from one section of memory to another, fill memory with the value of your choice, jump to sections of code, and quite simply, do pretty much of everything that the public domain machine monitors do so well. There is a trick with this one though. The transfer command isn't bright enough not to step on itself during a transfer of code back into the initial transfer area. If a slight move of code is desired that would normally step all over itself, transfer the code to a safe area elsewhere in RAM, then transfer it into the correct area for you. A little bit more work, but worth it for the results. Forgiving this one oversight, the monitor is really good, and a welcome addition after using the Vic and 64 without.

The extra disk commands are actually just a sub–set of the original BASIC 4.0 command set. BACKUP, COLLECT, COPY, DIRECTORY, DLOAD, DSAVE, HEADER, RENAME, and SCRATCH is the sub–set in whole. BASIC 4.0 had quite a few more, of which I only miss one, RECORD#. Relative files are supported by the 1541 drive unit, and relative file handling is possible with BASIC 2.0 commands, albeit somewhat cumbersome. Of all the BASIC 4.0 commands they had to choose from, I think RECORD# should have been included. To get even for this horrible trick, let's look at the commands given :

BACKUP The stupid command of the century for single drive units. BACKUP will duplicate a diskette from one drive to the next in dual drive units, ie. 4040, 8050, and 8250. Without an IEEE card, how can this command ever be used by the Plus 4 user? Could have left this one out and kept my RECORD#.

COLLECT To collect a diskette is to validate its BAM with the directory, of which BASIC 2.0 can perform via OPEN 15,8,15, "V" admirably. It's a nice command to have around, but not overly important for normal programming activities. RECORD# would have fit nicely in its spot in ROM.

COPY Another dumb command for the single drive units. How often do you want to copy a file under a different file name on the same diskette. It's often a life saver with the dual IEEE units, but a close to useless bit of code for single drive users. Guess what, RECORD# would have fit perfectly in the ROM occupied by COPY.

DIRECTORY A useful command that should never have disappeared from the VIC and 64's command set. It allows you to perform a passive directory of your diskette to your screen, without harming RAM in the least. Nice to see it back again. DIRECTORY in the plus four is better than the old CATALOG, however; it lets you display disk files selectively, using the usual pattern-matching and wildcard characters. For example, to display all files on drive 0 starting with the letter s, you could enter:

```
DIRECTORY D1, "s*"
```

DLOAD A handy command to load programs from disk with a default of unit number 8, drive #0. It's a little bit less tedious to load from disk with this one, but life could exist without it.

DSAVE As with DLOAD, DSAVE will allow you to SAVE programs to disk with a default of unit number 8, drive #0. Saves a bit of time and is more pleasant to use.

HEADER A command that never should be used if you don't like to waste computer time. HEADER will NEW a diskette as the command OPEN 15,8,15, "N0:DISKNAME,ID" does, but with one surprise. Once under way, it waits for the error status to arrive before giving you back control of your computer. Though a 1541 HEADER may only take a few minutes, it's time wasted that could have been put to better use. As before, I'll continue using the error channel, thank you.

RENAME It allows you to change the name of a file on diskette, as does the BASIC 2.0 command OPEN 15,8,15, "R0:NEWNAME=OLDNAME". RENAME saves time in the key bashing department, and is easier to remember as far as syntax goes. It's a pleasure to see this command back again.

SCRATCH The BASIC 2.0 command OPEN 15,8,15, "S0:FILENAME" does the same thing, but gives you, the user, greater credibility. As with the HEADER command, SCRATCH asks you if "you are sure" before acting. Kind of like an overprotective parent looking out for little Yimmie. Though Yimmie knows what he's doing, he can't be trusted without a double check. As an extra insult, SCRATCH waits for the error status to return before giving you your computer back. For me, RECORD# would fit better in the Plus 4 than this one.

Final note about the special disk command set. Their token values are not the same as their BASIC 4.0 counterparts. A minor point, but still worth mentioning.

More Plus's :

As with the B Machine of Protecto fame, the PLUS 4 has a built in reset button on the side. With it you can break out of programs directly into the machine language monitor, even if your code is stuck in an endless loop. Nice to have when debugging code, or trying to look at the code of others. The best thing about the reset is that if you hold down the STOP key while resetting, you'll be able to get back to BASIC with the current program in memory intact - the important BASIC text pointers are not disturbed. An addition that will prove its worth to you many times over.

As briefly mentioned above, available memory is an almost pleasant surprise. Of the 64k system, there is 32k of ROM, and 60671 bytes available for BASIC programming. The top of memory pointer is set to $FD00, of which the system uses the balance for internal stuff. Unlike the 64, swapping out of the system ROMs cannot be done. The RAM under ROM is automatically accessed by BASIC, as is the ROM above the RAM when working in machine code. Some special tricks are performed by CHRGET and the rest of the operating system to only see the RAM under the ROM when working in BASIC. It's rather weird when peering through the code, but it works. My only major complaint is not being able to swap out the ROM for RAM. Sure is nice with the 64 to replace the existing ROM routines with custom ones. Other than this slight snarl, the extra available BASIC RAM is nice.

To return to the extra BASIC command set, let's quickly look at the extra commands available.

In the utilities area, we now have AUTO, DELETE, HELP, MONITOR, RENUMBER, TROFF, and TRON to help us out. AUTO automatically prints line numbers on the screen for you while programming. DELETE allows you to delete sections of BASIC code from memory. HELP is used whenever your BASIC program crashes. It will display the last line executed, and the possible condition on that line responsible for the crash. Once displayed, the line is flashed on and off on the screen, just to catch your attention. Incidentally, there's no magic ROM routine just to flash the line - characters can be printed in flashing mode as easily as reverse mode (there are FLASH ON and FLASH OFF keys). MONITOR will bring you into the machine language monitor, Tedmon. RENUMBER will allow you to change the numbering of the lines of your BASIC program as you desire. You can even RENUMBER from a specific line # up in the increment of your choosing. TROFF and TRON turn a BASIC trace facility OFF and ON as you choose. All taken together, almost as good as Basic Aid, without some of the advanced commands.

In the graphics department, we now have the following extra commands. BOX, CHAR, CIRCLE, COLOR, DRAW, GRAPHIC, GSHAPE, LOCATE, PAINT, RCLR, RDOT, RGR, RLU, SCALE, SCNCLR, and SSHAPE. The explanations would require more than a little room, so have faith that they're alright. Most of the commands have a vast amount of possible parameters, of which most can be left off with defaults of logical choices. In simple terms, graphics programming in BASIC is a pleasure.

In the sound section, there are only two commands. SOUND and VOL. SOUND allows you to set the frequency, duration, and voice to be accessed, and the VOL command sets the volume of the voice specified. Though not as powerful as the SID sound with the 64, it sure is a heck of a lot easier.

Finally, more extra commands do exist, but their sole purpose in life is to make programming more pleasurable. They are DEC, DO, ELSE, ERR$, EXIT, HEX$, INSTR, JOY, KEY, LOOP, MID$, RESUME, TRAP, UNTIL, USING, WHILE, and GET KEY. The following is a very brief rundown of their powers.

Error trapping is possible within program mode through the use of the TRAP command. The format is TRAP line#. If any error occurs within program mode, even the depression of the (stop) key, the line# specified by the TRAP statement will be executed, to allow you to interrogate the error condition encountered. For this two variables are used, ER and EL. ER returns the error condition #, and EL returns the line # that the error was encountered in. For a bonus, a description of the error generated can be found in the string variable ERR$. To continue execution of the program after the error, the RESUME statement is used.

A new method of looping has come about, or new in the world of Commodore at least. It's the DO/WHILE series of commands that will excite all who use them. There are 4 possible formats: DO routine LOOP UNTIL condition; DO routine LOOP WHILE condition; DO UNTIL condition LOOP; DO WHILE condition LOOP . For an extra twist, the EXIT statement can also be used. The 'condition' tested for can EXIT outside of the loop, thereby allowing you a method out without leaving the stack in a messy state. In general terms, the DO/WHILE loop makes the GOTO statement slightly obsolete.

DEC and HEX$ could also fit in the utilities department, because they allow instant conversion of Hexadecimal to Decimal notation, or vice–versa. They work much like ASC and CHR$; they require the same argument 'types' and can be placed anywhere in expression evaluation statements of similar gender.

The ELSE statement is an omen for all who have ever wasted tons of code testing for a certain condition, then jumping all over the place to avoid bumping into code. The format is: IF condition THEN action :ELSE action . A guaranteed life saver in the massive code department.

INSTR is an incredibly useful command. It allows you to search for the first occurence of a string within a string, with an optional start location. When complete, it returns the start location that the string was found at. For anyone doing file manipulation operations, it will prove invaluable.

The USING keyword is used via PRINT USING "string expressions";list of expressions; , where specially formatted output can be generated at will. Printing specific portions of the strings, or printing numeric variables in correct decimal notation, with $ preceding the value, with correct signing of the values, or with signing if negation present, or automatic right justification of the value padded with spaces, is all possible. Special programming assignments dealing in these aspects of life will surely benefit for the PRINT USING statement.

JOY is a command used to return the current x,y position of the joystick on the screen. A handy item to have around when BASIC games are being written.

GET KEY is actually two separate keywords, GET and KEY. The interpreter finds the GET token, jumps to the appropriate code, of which the GET code makes a further check to see if the token value for KEY follows it. If so, the GET KEY code is branched to. GET KEY is used to wait for the depression of a key on the keyboard. Once pressed, it returns with the correct string value in the assigned string. Identical in concept to:

```
10 get a$: if a$ = "" then 10.
```

The KEY command also has a more useful feature: it lets you define the function keys and display the current definitions. The default definitions are often-used BASIC statements like LIST, RUN, DIRECTORY, DLOAD, and HELP.

Before wrapping things up, one more fine feature should be mentioned. Super screen editing tricks can be performed by using the ESC key along with a few other choice keys. Setting windows on the screen, deleting and inserting lines, auto insertion of characters, deleting from the cursor to the start of a line or to the end of the line, and scrolling the screen up or down, are just a few of the features available. In many ways it's better than my 8032 for screen editing. Too bad it's doesn't have an 80 column display mode.

My final analysis of the machine is, if you're willing to live with the bad points, which time may cure with adapters, then a terrific programming machine is waiting for you. One major point may stand in its way of success, though. The price. The 64 is quite a bit cheaper now than it was a few years ago at conception. This factor makes the Plus 4 hard to market. People would rather buy the lower priced, high powered 64 than take a chance with the new, equally powerful but higher priced new kid on the block. If Commodore was to announce a price reduction for the Plus 4, salvation may be within reach. Otherwise it may be just another really good Commodore machine that died an early death.

The Programmable Kitchen

Where is there more computing power: The bridge of the Enterprise or your kitchen? What's smarter, Spock's scanner or your dishwasher? Dumb questions, both, but the point is, what was major–league computing power yesterday is standard household appliance–type power today.

Microwave ovens, Stereo components, automobiles, washing machines, and games and toys will never be the same, with microprocessors running rampant in our homes and driveways. The benefits to the average consumer are obvious: more features, lower cost (sometimes), and greater ease of operation (well, usually not, but that's the idea). So the average lay person gains from the CPU invasion, except maybe the fellow who wishes the nagging voice in his dashboard would leave him alone, but what about us programmers?

More accurately, what about us hackers, those of us who teach computers stunts for their own sake, to whom the act of programming is more important than the end result? We've been excited by each new generation of computers, each more powerful than the last. It's easy to understand lusting after new, more powerful machines; for some, it's the latest Porsche or Ferrarri. When it comes to computing, it might be the next Commodore. But a household appliance? What could be more boring than a washing machine or refrigerator? Do you show your new fridgidaire to your friends, saying things like, "Yeah, it's got a 48–cube capacity, but I'm only running with 24 now, and that's plenty. It'll chill a beer in 3.2 microyears, an' . . . "?

No, most of us who haven't toasted too many neurons staying up late tracking down and dissecting bugs, don't. But that could change. What manufacturers of these compu–pliances should do is put in a back door for hackers to use at their own risk. Just a little serial port behind a panel somewhere and a little change in the device's firmware could do wonders to change our outlook on even the most mundane items of necessity. A bit of free RAM (preferably non–volatile), a memory write and memory execute command, a few memory maps thrown into the manual, and the door is open to the enterprising programmer. Bored with your computer? Why not teach your Compact Disk unit to play backwards?

Think of the possibilities: The microwave beeps a few bars of "Hungry Like The Wolf" when the left-overs are reheated. Your VCR forward–scans for 3 minutes whenever you simultaneously press the forward and reverse keys - the "commercial" function. Your car gains 20 horsepower, perhaps at the expense of fuel economy (maybe a bit of pollution control, too). And that nagging voice under the dashboard could be made to change its tune: "Shut the door, stupid!".

While you're educating your electronic servants to the ways of the world, why not take the next step and set up a communications network? A bit of cable strung about between devices (LOTS of cable if you want to include your car) would allow an efficient household organization system. Picture this: The oven sends the FRFC (Food Ready For Consumption) signal to the coffee machine, which starts preparing the after-dinner coffee. The coffee machine sends a DRTBW (Dishes Ready To Be Washed) to the dishwasher when the coffee's finished, which starts sudsing up for its performance. Now the stereo receives a PDMM (Post–Dinner Mood Music) to complete the perfect evening. Inviting your date over for dinner will never be the same.

All your smart devices could be custom–programmed to reflect *your* needs and specifications, not those of some engineer in Japan. And as more and more devices incorporate microprocessors, your environment will become more sensitive to your needs. Someday even your toaster could concern itself with you state of mind that morning before deciding on a darkness level. Maybe someone could teach an electric razor about how to handle acne territory. There's really no limit to the computer's infiltration of electricland.

So take action now: write the manufacturers and urge swift action on this matter. Programming power to the people! Make the products conform to our needs, not us to theirs! I want the fabric softener added later in the wash cycle – what're you gonna do about it? Such inducements may bring results with time, and we'll all have more control over our environments, and ultimately, our lives.

Meanwhile, back on the Enterprise . . . "Captain, the bridge won't respond; due to a computer malfunction, we've been locked out. The enterpise will explode in a fiery storm of death in exactly 17.2 seconds." -CZ

Chopper and Labelgun for the Commodore 64

Chris Miller
Kitchener, ONT

A multiple statement line splitter and a label re-definition utility for PAL source programs

I very much doubt if there exists anywhere on Earth, assembly language source written in a style more horrible, more unreadable, more totally incomprehensible than some of my very own.

Sure, it has to be bad to be the worst. Real bad. But, imagine if you can, thousands upon thousands of lines of code, wherein a single helpful comment would die of sheer loneliness. Well, that is just the beginning. Now picture, if you will, in this barren place, a multitude of meaningless labels, a veritable menagerie of mindless symbols, scores of infinitesimal, idiotic identifiers, for whom the sole purpose of existence is to allow the assembler to assemble. What ever was I thinking!

I use the PAL assembler by Brad Templeton. One really keen feature of PAL (which I dearly love) is its willingness to let you link all sorts of statements together on a single line using colons. Neat–O! You can write a small subroutine on one line and still have room to start the next one, even attach another (teensy) label. Though seriously Brad, it might have been less cruel to leave this feature out. It's so easy to get carried away. . . or maybe I'm the only colon addict out here. Anyway, the only cure is total abstention. Now I wouldn't use a colon even if I had to pop the stack a billion times in a row. I've stopped cold turkey.

The first thing I noticed after kicking the colon habit was that my screen looked funny. The right hand side was bare, empty, utterly devoid of data. Lengthening the labels helped a little, but the right side still felt vacant.

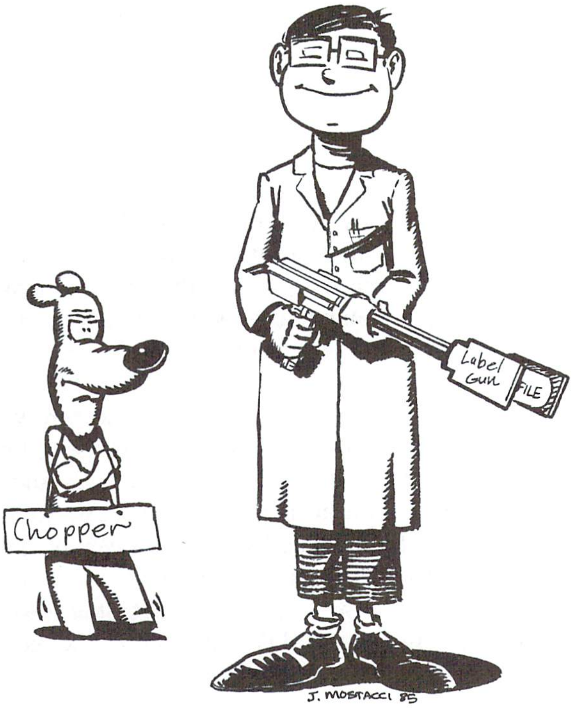
Then it came to me! Why not put useful comments and markers over there so that when I went, say, to look for the READKEY routine someday, it wouldn't take the usual half a fortnight.

In so doing I discovered a number of situations in which my comments, although accurately depicting the code, still didn't make any sense. The cure for these redundant, extraneous, often irrelevant annotations was, quite simply, to remove the offending line, code and all. Amazingly, my programs get along fine without them.

If you are presently asking yourself why you would ever want to type in a piece of code written by a lunatic such as myself. . . relax, I told you I'm almost cured. You may not understand my source perfectly, but it shouldn't drive you insane either. And it really works well.

So suppose you do decide to take an old colon–laden program and repair it; you know, lots of white space, code on the left, comments on the right, meaningful labels and so on: A lengthy piece could carry you clean into your next crisis.

Wouldn't it be much less work to copy my Chopper program? Then cure all your impacted code in an instant forevermore. Chopper is intelligent enough not to split a line where a colon occurs between quotation marks. Extra colons will be ignored.



Chopper is memory based. BASIC must be able to hold both the original and the de–colonized version of your source. This gives you 16K for the original, more for the new version. Pointers are set to the new one. After SYSteming through Chopper, simply list, sprinkle with wonderful, descriptive comments, and save your new program.

Labelgun

Easier said than done, I know. Bad style is hard to use and harder understood. Now all those lousy little labels catch up with you. Like, if I had only called it INPUTKEY instead of IK how much more sense its thirty some odd appearances would have lent my source. And take all those silly subroutines that have been modified and modified until only the name remains the same. What to do with those zillions of nonsensical, even downright misleading names?

I have fond memories of one particularly absorbing evening spent tracking down and changing all the occurences of an unusually popular little routine called "FM" whose job, it turned out, was to tell sprites whether to appear in Hi–resolution or Multi–colour mode. FM = FINDMODE. Well, it made sense at the time. Out of great misery springs great art, or so they say. Thus came Labelgun to be; to forgive the sin of sloppy symbol selection. If you take the trouble now to copy it you will forever be able to kill those lunkhead labels dead, instantly and automatically replacing them with descriptive, well thought–out ones.

Labelgun sets up as a Chrget wedge. That is, it adds a command to BASIC by creating a detour through itself in BASIC's search for commands routine.

The code is less than 700 bytes long and sits at $C000 hex or 49152. After loading you only need call it once via SYS 49152. This will set and save some essential pointers, but leads one down the path of despair (crash) if repeated with the wedge already in effect. Entering an X will turn off the wedge and restore the pointers.

With Labelgun active, the command: C,CM,COLORMEM will, instead of a syntax error, cause every occurence of the use of CM as a symbol to be changed to COLORMEM throughout any BASIC (PAL) source listing. The number of changes made is printed out upon completion.

Labelgun will not miss any occurences of a symbol, and only very rarely attack the wrong string (your comments). Any global changes such as Labelgun performs are frought with a small amount of risk. You will not, as a rule, want to generate an already existing label which would at best lead to redefinition errors, and at worst, program malfunction. To check the uniqueness of a label I try converting it to itself. "0 alterations made. . ." tells me it is safe to use.

Labelgun is not opposed to changing numbers into labels. For example, the command C,$FFD2,PRINT is valid and useful. It is considered good style to make abundant use of constants. Any value used more than once should probably be assigned to a constant at the start of your program. This will make your source much easier to modify and to understand. With a command like C,53281,BKGCOLOR the conversion is painless indeed.

Labelgun can also be used to rename variables in a BASIC program — it doesn't mess up your REMs, either! –T.Ed

Far more programming effort goes into modifying the old than writing the new. Some giant software packages have had meager beginnings. So if it's worth disk space, it's worth being made comprehensible and reworkable.

Labelgun and Chopper have easily saved me the time it took to write them. I use my own two–pass, label generating disassembler which dumps source into BASIC program space to be listed. With labelgun I then begin putting meaningful names to values and generated labels. I can't imagine an easier way of converting raw code to readable, usable source.

With Labelgun and Chopper in your utility arsenal you will be able to write PAL or similar source using lots of colons and single character labels. With everything fresh in your mind it may not matter, and be faster. When you are finished and have everything working the way you want you can easily render it understandable for future references.

Listing 1: BASIC loader for chopper

```
PI  10 rem* data loader for "chopper" *
LI  20 cs=0
EC  30 for i=51200 to 51426:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FG  60 if cs<>30292 then print "***** data error *****": end
JG  70 rem sys 51200
AF  80 end
IN  100 :
GF  1000 data 165,  43, 133, 251, 133, 253, 165,  44
KA  1010 data 133, 252,  24, 105,  64, 133, 254, 133
OA  1020 data  44, 169,   3, 141, 134,   3, 141, 135
DA  1030 data   3, 169,   0, 141, 136,   3, 141, 133
EE  1040 data   3, 160,   3, 145, 253, 198, 254, 160
KF  1050 data 255, 145, 253, 230, 254, 160,   1, 169
LK  1060 data  10, 145, 253, 200, 141, 132,   3, 145
BD  1070 data 253, 172, 134,   3, 200, 238, 135,   3
EI  1080 data 238, 134,   3, 177, 251, 240,  31, 201
JG  1090 data  34, 208,  10,  72, 173, 136,   3,  73
CC  1100 data   1, 141, 136,   3, 104, 201,  58, 208
LE  1110 data   5, 174, 136,   3, 240,  31, 172, 135
IE  1120 data   3, 145, 253,  76,  57, 200, 200, 200
CN  1130 data 177, 251, 208,   3,  76, 192, 200, 160
BE  1140 data   0, 177, 251, 170, 200, 177, 251, 133
BI  1150 data 252, 134, 251, 160,   3, 140, 134,   3
HK  1160 data 172, 135,   3, 192,   4, 240,  46, 169
JM  1170 data   0, 145, 253, 200, 152, 166, 254,  24
LH  1180 data 101, 253, 144,   1, 232, 133, 253, 134
NK  1190 data 254, 169,   1, 168, 145, 253, 200, 173
MK  1200 data 132,   3,  24, 105,  10, 144,   3, 238
OL  1210 data 133,   3, 141, 132,   3, 145, 253, 173
DL  1220 data 133,   3, 200, 145, 253, 169,   3, 141
CF  1230 data 135,   3,  78, 136,   3,  76,  57, 200
NN  1240 data 169,   1, 168, 145, 253, 172, 135,   3
CL  1250 data 169,   0, 145, 253, 200, 145, 253, 200
NN  1260 data 145, 253, 200, 152, 166, 254,  24, 101
MO  1270 data 253, 144,   1, 232, 133,  45, 134,  46
IJ  1280 data  76,  51, 165
```

Listing 2: Labelgun loader

```
BP  10 rem* data loader for "labelgun" *
LI  20 cs=0
KF  30 for i=49152 to 49831:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
JH  60 if cs<>83704 then print "***** data error *****": end
EI  70 rem sys 49152
AF  80 end
IN  100 :
NA  1000 data 173,   8,   3, 141, 232,   3, 173,   9
BH  1010 data   3, 141, 233,   3, 169,  23, 141,   8
PG  1020 data   3, 169, 192, 141,   9,   3,  96, 165
BI  1030 data  58, 201, 255, 208, 108, 165,  43, 133
CM  1040 data 251, 165,  44, 133, 252, 160,   1, 177
KH  1050 data 122, 201,  88, 208,   3,  76, 122, 192
EG  1060 data 201,  67, 208,  85, 169,   0, 141,  69
DB  1070 data   3,  32, 115,   0,  32, 115,   0, 240
PD  1080 data  81, 201,  44, 208,  71,  32, 115,   0
MN  1090 data 240,  72, 162,   0, 157,  72,   3,  32
LD  1100 data 115,   0, 240,  62, 201,  44, 240,   7
BA  1110 data 232, 224,  40, 208, 239, 240,  45,  32
JE  1120 data 115,   0, 240,  46, 232, 142,  65,   3
LN  1130 data 162,   0, 157, 112,   3,  32, 115,   0
BA  1140 data 240,  63, 232, 224,  40, 208, 243,  76
KG  1150 data 140, 192,  32, 115,   0, 173, 232,   3
CK  1160 data 141,   8,   3, 173, 233,   3, 141,   9
GM  1170 data   3, 108, 232,   3,  32, 152, 192,  76
MO  1180 data  29, 193,  32, 152, 192,  76,  34, 193
ML  1190 data 162,   0, 189,  72, 194,  32, 210, 255
JG  1200 data 232, 224,  46, 208, 245,  96, 162,  22
GD  1210 data 221, 145, 194, 240,   3, 202,  16, 248
BB  1220 data  96, 232, 142,  64,   3, 160,   4, 162
GL  1230 data   0, 177, 251, 240,  78, 200, 221,  72
JJ  1240 data   3, 208, 244, 232, 224,   1, 208,  18
AJ  1250 data 192,   5, 208,   5, 142,  68,   3, 240
```

```
CM  1260 data   9, 136, 136, 177, 251, 200, 200, 141
HE  1270 data  68,   3, 236,  65,   3, 208, 218, 177
GK  1280 data 251,  32, 166, 192, 208, 209, 173,  68
NA  1290 data   3,  32, 166, 192, 208, 201, 238,  69
CI  1300 data   3, 152,  56, 237,  65,   3, 141,  66
KF  1310 data   3, 166, 252, 152,  24, 101, 251, 144
DB  1320 data   1, 232, 142, 235,   3, 141, 234,   3
HM  1330 data  76, 101, 193, 160,   0, 177, 251, 170
DH  1340 data 200, 177, 251, 133, 252, 134, 251, 177
DB  1350 data 251, 208, 154, 240,   5,  32, 115,   0
BG  1360 data 208, 251, 165, 122, 208,   2, 198, 123
BH  1370 data 198, 122, 169,  13,  32, 210, 255, 162
BE  1380 data   0,  56, 173,  69,   3, 201, 100, 144
KI  1390 data   7, 169,  49,  32, 210, 255, 233, 100
NJ  1400 data 168,  56, 233,  10, 144,   3, 232, 208
HL  1410 data 247, 138,   9,  48,  32, 210, 255, 152
FM  1420 data   9,  48,  32, 210, 255, 162,   0, 189
JO  1430 data 118, 194,  32, 210, 255, 232, 224,  27
PG  1440 data 208, 245, 108, 232,   3, 173,  65,   3
CJ  1450 data  56, 237,  64,   3, 240,  19, 144,   6
OA  1460 data 141,  67,   3,  76, 252, 193,  73, 255
CE  1470 data  24, 105,   1, 141,  67,   3,  76, 153
FC  1480 data 193,  32, 135, 193,  76, 183, 192, 172
FI  1490 data  66,   3, 162,   0, 189, 112,   3, 145
HG  1500 data 251, 232, 200, 236,  64,   3, 208, 244
BK  1510 data  96, 166,  45, 142, 184, 193, 166,  46
LE  1520 data 142, 185, 193, 173,  67,   3,  24, 101
DA  1530 data  45, 144,   2, 230,  46, 133,  45, 141
CI  1540 data 187, 193, 165,  46, 141, 188, 193, 173
JG  1550 data 255, 255, 141, 255, 255, 206, 184, 193
CF  1560 data 173, 184, 193, 201, 255, 208,   3, 206
IL  1570 data 185, 193, 206, 187, 193, 173, 187, 193
ID  1580 data 201, 255, 208,   3, 206, 188, 193, 173
BG  1590 data 185, 193, 205, 235,   3, 208, 216, 173
DN  1600 data 184, 193, 205, 234,   3, 176, 208,  32
KI  1610 data 135, 193, 152,  72, 165, 251, 164, 252
MH  1620 data 133,  34, 132,  35,  32,  59, 165, 104
BF  1630 data 168,  76, 183, 192, 174, 235,   3, 142
CM  1640 data  23, 194, 173, 234,   3, 141,  22, 194
AL  1650 data  56, 237,  67,   3, 176,   1, 202, 141
GM  1660 data  25, 194, 142,  26, 194, 173, 255, 255
BF  1670 data 141, 255, 255, 238,  22, 194, 208,   3
HO  1680 data 238,  23, 194, 238,  25, 194, 208,   3
BH  1690 data 238,  26, 194, 173,  26, 194, 197,  46
HB  1700 data 208, 227, 173,  25, 194, 197,  45, 144
GN  1710 data 220, 165,  45,  56, 237,  67,   3, 176
AN  1720 data   2, 198,  46, 133,  45,  76, 231, 193
KD  1730 data  84,  79,  32,  18,  67,  72,  65,  78
DE  1740 data  71,  69,  32,  83,  84,  82,  73,  78
LI  1750 data  71, 146,  32,  58,  13,  67,  44,  60
PH  1760 data  79,  76,  68,  83,  84,  82,  73,  78
MG  1770 data  71,  62,  44,  60,  78,  69,  87,  83
AF  1780 data  84,  82,  73,  78,  71,  62,  32,  65
DJ  1790 data  76,  84,  69,  82,  65,  84,  73,  79
OI  1800 data  78,  83,  32,  77,  65,  68,  69,  32
OK  1810 data  84,  79,  32,  83,  79,  85,  82,  67
DI  1820 data  69,  35,  44,  59,  58,  41,  40,  47
PK  1830 data  45,  42,  43,  60,  62,  61,  32, 170
PI  1840 data 171, 172, 173, 177, 178, 179,   0,   1
```

Listing 3: "Chopper" PAL source

```
MI  10 rem  chopper   chris miller (1984)
EJ  20 sys700 ;pal 64 assembler
GI  30 .opt oo
JC  40 *              =    $c800
BF  50 ;*** zero page pointers ************
KI  60 sob            =    43
KO  70 sov            =    45
CK  80 oldprg         =    251
OB  90 newprg         =    253
MB  100 ;*******rom subroutine(s)***********
GD  110 rechain       =    $a533          ;*links lines*
FP  120 ; ****** variable addresses *******
BH  130 line          =    900
LK  140 oldindx       =    902
BH  150 newindx       =    903
PI  160 quotflag      =    904
BL  170 ; ***** constant values ***********
LM  180 spread        =    $40            ;*program seperation*
DH  190 step          =    10             ;*line# increment*
GI  200 ;***begin initialization************
LI  210 initialize    =    *
GA  220               lda  sob            ;*set old pointer to sob*
PF  230               sta  oldprg
KO  240               sta  newprg
CP  250               lda  sob + 1
PD  260               sta  oldprg + 1
EI  270               clc
HO  280               adc  #spread        ;*new to sob + spread*
IC  290               sta  newprg + 1
CG  300               sta  sob + 1
HC  310               lda  #3
AJ  320               sta  oldindx        ;*initialize indexes*
GH  330               sta  newindx
PD  340               lda  #0
JE  350               sta  quotflag       ;*and quote flag*
FJ  360               sta  line + 1       ;*hi byte of line#*
DM  370               ldy  #3
KH  380               sta  (newprg),y     ;*in place now*
JE  390               dec  newprg + 1
NK  400               ldy  #$ff
KJ  410               sta  (newprg),y     ;*basic start zero*
MI  420               inc  newprg + 1
LP  430               ldy  #1
FJ  440               lda  #step
EE  450               sta  (newprg),y     ;*phoney first link*
MK  460               iny
DO  470               sta  line           ;*lobyte of line#*
FK  480               sta  (newprg),y     ;*in place*
IG  490 ;*begin primary loop to find colons*
EC  500 fndcolon      =    *
LH  510               ldy  oldindx
IO  520               iny
AC  530               inc  newindx
JF  540               inc  oldindx
CO  550               lda  (oldprg),y     ;*look at next char*
CN  560               beq  eoln           ;*if zero, new line*
DB  570 quotc         cmp #34             ;*quotation mark*
JA  580               bne  colonchk       ;*if not check for colon*
KM  590               pha
FL  600               lda  quotflag
FO  610               eor  #1             ;*if on-off..if off-on*
HA  620               sta  quotflag
LP  630               pla                 ;*get back current char*
DJ  640 colonchk      cmp # " : "
OO  650               bne  next           ;*not colon, keep going*
JA  660               ldx  quotflag       ;*see if between quotes
FE  670               beq  maknulin       ;*if not, new line*
EE  680 next          ldy  newindx
NH  690               sta  (newprg),y
HB  700               jmp  fndcolon
MH  710 ;***** end of old program line ****
IK  720 eoln          =    *              ;    *y = oldindx*
KL  730               iny
EM  740               iny
KD  750               lda  (oldprg),y     ;*see next hilink*
```

```
NL  760           bne  ccc          ;*end of source test*
FG  770           jmp  finished
CA  780 ccc       ldy  #0
HC  790           lda  (oldprg),y   ;*set pntrs to link*
LO  800           tax
KA  810           iny
BK  820           lda  (oldprg),y
JH  830           sta  oldprg+1
LH  840           stx  oldprg       ;*pntrs now reset*
JO  850           ldy  #3           ;***initializes oldindx***
PA  860 ;***** end of new program line ****
FK  870 maknulin  =    *
LC  880           sty  oldindx
IM  890           ldy  newindx
EM  900           cpy  #4           ;*test for extra colons*
IO  910           beq  pass         ;*if so ignore*
DI  920           lda  #0
EG  930           sta  (newprg),y   ;*end of line zero*
HM  940           iny               ;
MG  950           tya               ;*distance to new line*
EO  960           ldx  newprg+1
AE  970           clc
IF  980           adc  newprg       ;*point to next line*
JO  990           bcc  eeee
EM  1000          inx
OJ  1010 eeee     sta  newprg
OF  1020          stx  newprg+1
DP  1030          lda  #1
PN  1040          tay
EF  1050          sta  (newprg),y   ;*phoney link*
EA  1060          iny
KL  1070          lda  line         ;*make new linenum*
OK  1080          clc
BB  1090          adc  #step
LC  1100          bcc  aaa
GM  1110          inc  line+1
LE  1120 aaa      sta  line
FD  1130          sta  (newprg),y
EM  1140          lda  line+1
OF  1150          iny
DF  1160          sta  (newprg),y
MP  1170 pass     =    *
AH  1180          lda  #3           ;*initialize new index*
CN  1190          sta  newindx
EI  1200          lsr  quotflag     ;*initialize quote flag*
FB  1210          jmp  fndcolon
NJ  1220 ;******* exit routine ***********
NM  1230 finished =    *            ;*get ready to exit*
FM  1240          lda  #1
BL  1250          tay
JD  1260          sta  (newprg),y   ;*phoney last link*
LG  1270          ldy  newindx      ;*index end of line*
AL  1280          lda  #0           ;*final three zeros*
FN  1290          sta  (newprg),y
EP  1300          iny
JO  1310          sta  (newprg),y
IA  1320          iny
NP  1330          sta  (newprg),y
PJ  1340          iny               ;*byte past last zero*
NP  1350          tya
BE  1360          ldx  newprg+1     ;*form new sov pntr*
AN  1370          clc
CB  1380          adc  newprg
PF  1390          bcc  ddd
EF  1400          inx
LO  1410 ddd      sta  sov
KF  1420          stx  sov+1
MJ  1430          jmp  rechain      ;*link program lines*
```

Listing 4: "Labelgun" PAL source

```
EI  10 rem label gun!!! by chris miller (1984)
DK  20 sys700 ;pal 64
GI  30 .opt oo
JB  40 *            =    $c000
EN  50 ;***** numerical constants *******
MH  60 maxlen       =    40
IM  70 ;****** subroutine+addresses ****
GH  80 chrget       =    $73
IJ  90 print        =    $ffd2
IA  100 rechain     =    $a53b
HJ  110 ;**** indirect pointers ********
GH  120 strtlink    =    $22
GB  130 sob         =    43              ;*start of source*
GN  140 sov         =    45              ;*end of source*
IE  150 textptr     =    $7a
FJ  160 link        =    251
NE  170 chrptr      =    776
OP  180 chrvec      =    1000            ;*holds chrget vector*
IC  190 labelend    =    1002            ;*end of label ptr*
NJ  200 ;**** variable addresses ********
AO  210 oldstrng    =    840
MC  220 newstrng    =    880
IL  230 newsize     =    832
AB  240 oldsize     =    833
PL  250 indexlbl    =    834
AK  260 howfar      =    835
FM  270 preceeds    =    836
EE  280 howmany     =    837             ;*changes made*
JO  290 ;***** flag registers ***********
LB  300 modeflag    =    58
OL  310 ;*** set charget pointer **********
JK  320             lda  chrptr
MN  330             sta  chrvec
PH  340             lda  chrptr+1
CL  350             sta  chrvec+1
AF  360             lda  #<getinput
JB  370             sta  chrptr
AG  380             lda  #>getinput
PO  390             sta  chrptr+1
MH  400             rts
PN  410 getinput    =    *
BE  420 ; **** see if in immediate mode ***
AA  430             lda  modeflag  ;*test mode (program/immediate)*
PF  440             cmp  #255
BJ  450             bne  stopchek
MN  460             lda  sob             ;*start of basic ptr*
AH  470             sta  link
IN  480             lda  sob+1
AJ  490             sta  link+1
BE  500             ldy  #1
EM  510             lda  (textptr),y     ;*see first char*
CA  520             cmp  #"x"            ;*is it an x*
DB  530             bne  checkc
PH  540             jmp  diswedge        ;*if so dismantle wedge*
LK  550 checkc      cmp  #"c"
PP  560             bne  stopchek
EL  570 ;***** check the target string ***
PC  580             lda  #0
LP  590             sta  howmany
EF  600             jsr  chrget          ;*collect the c*
MI  610             jsr  chrget          ;*get next char*
CB  620             beq  eolerror        ;*eoln sets z flag*
GJ  630             cmp  #","
LB  640             bne  error           ;*we want a comma*
MK  650             jsr  chrget
DA  660             beq  eolerror        ;*no eoln yet please*
FO  670             ldx  #0
PH  680 readold     =    *
GK  690             sta  oldstrng,x      ;*collect old label*
ON  700             jsr  chrget
KL  710             beq  eolerror        ;*no end of line please*
GI  720             cmp  #","            ;*seperates old/new*
MD  730             beq  readnew
AM  740             inx
HG  750             cpx  #maxlen
DJ  760             bne  readold
HA  770             beq  error           ;*string too long*
KE  780 readnew     =    *               ;*collect new label*
ID  790             jsr  chrget
OE  800             beq  eolerror        ;*no eoln*
GA  810             inx
OA  820             stx  oldsize         ;*save length*
```

```
FI  830              ldx  #0
EM  840 aaaa         sta  newstrng,x
EH  850              jsr  chrget
OG  860              beq  findstr         ;*valid eoln*
CE  870              inx
JO  880              cpx  #maxlen
GI  890              bne  aaaa
ON  900              jmp  error
ID  910 ;*****restore the wedge pointer*****
NL  920 diswedge     =    *
CP  930              jsr  chrget          ;*eat up the x*
AA  940              lda  chrvec
NF  950              sta  chrptr
GN  960              lda  chrvec+1
DD  970              sta  chrptr+1
OH  980 stopchek     =    *
EL  990              jmp  (chrvec)
DE 1000 ;****** input error ***********
CN 1010 error        =    *
JP 1020              jsr  howtodo         ;*message*
IG 1030              jmp  eatline
AG 1040 eolerror     =    *
ME 1050              jsr  howtodo
EG 1060              jmp  inputend
JI 1070 ;****** input error ***********
GG 1080 howtodo      =    *               ;*prints error message*
KD 1090              ldx  #0              ;
AB 1100 aaa          lda  errmsg,x
EM 1110              jsr  print
MD 1120              inx
OI 1130              cpx  #errmsgx-errmsg
MH 1140              bne  aaa
KG 1150              rts
FK 1160 ;*******check for operators********
FM 1170 checkops     =    *
MG 1180              ldx  #expressx-express-1
PD 1190 keepon       cmp express,x
NH 1200              beq  leave
BH 1210              dex
NN 1220              bpl  keepon
CL 1230 leave        rts
EM 1240 ;***** find the target string  ***
JL 1250 findstr      =    *
IM 1260              inx
DC 1270              stx  newsize         ;*length of new label*
KK 1280 nextline     =    *
NF 1290              ldy  #4
JC 1300 newtest      =    *
FG 1310              ldx  #0
NC 1320 getchar      lda  (link),y
FM 1330              beq  nextlink        ;*eol test*
MB 1340              iny
CM 1350              cmp oldstrng,x       ;*is it the target*
LI 1360              bne  newtest
GD 1370              inx
EL 1380              cpx  #1              ;*is it first char*
JJ 1390              bne  testnext
FL 1400              cpy  #5              ;*does label start line*
II 1410              bne  zzzz
LH 1420              stx  preceeds        ;*disable pre-check*
DE 1430              beq  testnext        ;*must brnanch*
PN 1440 zzzz         dey
FG 1450              dey
EH 1460              lda  (link),y
OJ 1470              iny
IK 1480              iny
NA 1490              sta  preceeds        ;*preceeding char*
MC 1500 testnext     cpx  oldsize;
NL 1510              bne  getchar
CA 1520 ; *****potential occurance*********
KG 1530              lda  (link),y        ;*char after label*
JJ 1540              jsr  checkops        ;*check for delimiters*
JE 1550              bne  newtest
KO 1560 prior        lda  preceeds        ;*char before*
KF 1570              jsr  checkops        ;*for valid delimiters*
HG 1580              bne  newtest
DP 1590 ;***** found an occurance *********
```

```
JK 1600 found        =    *
JN 1610              inc  howmany
LG 1620              tya                  ;*points end of label*
PN 1630              sec
LK 1640              sbc  oldsize
IL 1650              sta  indexlbl        ;*save label start*
AE 1660              ldx  link+1
ND 1670              tya
GA 1680              clc
IO 1690              adc  link
JL 1700              bcc  jjj
KI 1710              inx
AF 1720 jjj          stx  labelend+1
LF 1730              sta  labelend
BH 1740              jmp  changstr        ;*change the string*
DB 1750 ;**** ready to look at next line **
DP 1760 nextlink     =    *               ;*link pntr=next link*
FD 1770              ldy  #0
GD 1780              lda  (link),y        ;*lobyte of link*
JM 1790              tax
IO 1800              iny
BB 1810              lda  (link),y        ;*hibyte*
CM 1820              sta  link+1
LE 1830              stx  link            ;*new link in pointer*
EA 1840              lda  (link),y        ;*test for eof*
AN 1850              bne  nextline        ;*go do next line*
AG 1860              beq  inputend
JJ 1870 ;***** close up and quit ********
KC 1880 eatline      =    *               ; *get eoln*
MP 1890 qqq          jsr  chrget
EN 1900              bne  qqq
IP 1910 ;***end of line zero reached*******
JG 1920 inputend     =    *               ;*have eoln*
PM 1930              lda  textptr         ;*let basic find it*
MB 1940              bne  zzz             ;*by backing up textptr*
OP 1950              dec  textptr+1
CO 1960 zzz          dec  textptr
BK 1970 ;****** print number of changes ***
DJ 1980              lda  #13             ;*carriage return*
ED 1990              jsr  print
HB 2000              ldx  #0
LF 2010              sec
DA 2020              lda  howmany         ;*number of changes*
HI 2030              cmp #100             ;*works for n<200*
AD 2040              bcc  num
MO 2050              lda  #"1"
KH 2060              jsr  print
DG 2070              sbc  #100
AN 2080 num          tay
LK 2090              sec
BM 2100              sbc  #10
ME 2110              bcc  calc
EC 2120              inx
DL 2130              bne  num
MM 2140 calc         txa
HC 2150              ora  #$30            ;**** asci rep val *******
ON 2160              jsr  print
BD 2170              tya
NE 2180              ora  #$30
MP 2190              jsr  print
PN 2200              ldx  #0
DH 2210 changnum     lda  changes,x
KB 2220              jsr  print
CJ 2230              inx
CM 2240              cpx  #changesx-changes
AD 2250              bne  changnum
KK 2260              jmp  (chrvec)
GC 2270 ;***** change the string ********
MF 2280 changstr     =    *               ;*compare lengths*
FC 2290              lda  oldsize
NH 2300              sec
OE 2310              sbc  newsize
JD 2320              beq  replace
LD 2330              bcc  lll
JN 2340              sta  howfar          ;*to move memory down*
CM 2350              jmp  lessmem
CM 2360 lll          eor  #255            ;*get twos compliment*
```

```
IL  2370            clc
LC  2380            adc  #1
HP  2390            sta  howfar        ;*to move memory up*
KK  2400            jmp  moremem
FJ  2410 ;**** replace old string *********
MN  2420 replace    =    *
ND  2430            jsr  putlabel
MP  2440            jmp  newtest
EN  2450 ;********put in new label**********
HO  2460 putlabel   =    *
HL  2470            ldy  indexlbl      ;*index start of string*
HP  2480            ldx  #0
KM  2490 yyy        lda  newstrng,x
CM  2500            sta  (link),y
KK  2510            inx
IL  2520            iny
II  2530            cpx  newsize
EI  2540            bne  yyy
CO  2550            rts
DD  2560 ; ***** push memory up by howfar***
KP  2570 moremem    =    *
LC  2580            ldx  sov           ;*self made address*
CA  2590            stx  takefrom+1
IL  2600            ldx  sov+1
IB  2610            stx  takefrom+2
NB  2620            lda  howfar        ;*get howfar in a*
ML  2630            clc
NJ  2640            adc  sov
NL  2650            bcc  www
IL  2660            inc  sov+1
KO  2670 www        sta  sov           ;*sov now reset up*
DO  2680            sta  putitat+1
GL  2690            lda  sov+1
IP  2700            sta  putitat+2
IG  2710 ;**** start pushing memory up *****
OL  2720 takefrom   lda  $ffff         ;*self modifying*
HK  2730 putitat    sta  $ffff
GM  2740            dec  takefrom+1    ;*move takefrom ptr down*
IA  2750            lda  takefrom+1
KM  2760            cmp #$ff
GK  2770            bne  uuuu
DC  2780            dec  takefrom+2
AG  2790 uuuu       dec  putitat+1
NB  2800            lda  putitat+1
MP  2810            cmp #$ff
OM  2820            bne  tttt
HD  2830            dec  putitat+2
KM  2840 tttt       lda  takefrom+2    ;*see if done*
EI  2850            cmp labelend+1
EI  2860            bne  takefrom
AI  2870            lda  takefrom+1
AO  2880            cmp labelend
JL  2890            bcs  takefrom
EL  2900 ;****** readjust links ************
LD  2910 relink     =    *
HC  2920            jsr  putlabel
JC  2930            tya
IP  2940            pha
CO  2950            lda  link
IF  2960            ldy  link+1
FH  2970            sta  strtlink
LO  2980            sty  strtlink+1
FA  2990            jsr  rechain
AE  3000            pla
BJ  3010            tay
AE  3020            jmp  newtest
HJ  3030 ;*****  less memory needed ********
NI  3040 lessmem    =    *
JM  3050            ldx  labelend+1    ;*start at end of label*
AB  3060            stx  movefrom+2
KA  3070            lda  labelend      ;
AC  3080            sta  movefrom+1    ;*self made address*
DJ  3090            sec
MH  3100            sbc  howfar
PJ  3110            bcs  ppp
HO  3120            dex
CG  3130 ppp        sta  moveto+1
```

```
FN  3140            stx  moveto+2
FB  3150 ;**** start pulling memory down ***
DF  3160 movefrom   lda  $ffff         ;*self mod*
MF  3170 moveto     sta  $ffff
GA  3180            inc  movefrom+1    ;*point moveto next byte*
MM  3190            bne  nnn
CC  3200            inc  movefrom+2
AJ  3210 nnn        inc  moveto+1
AP  3220            bne  ooo
FL  3230            inc  moveto+2
CK  3240 ooo        lda  moveto+2
BA  3250            cmp sov+1          ;*see if sov reached*
KE  3260            bne  movefrom
KL  3270            lda  moveto+1
AH  3280            cmp sov
PD  3290            bcc  movefrom
EN  3300            lda  sov           ;*correct sov pointer*
PG  3310            sec
IF  3320            sbc  howfar
JG  3330            bcs  mmm
LD  3340            dec  sov+1
AA  3350 mmm        sta  sov           ;*sov now lowered*
PI  3360            jmp  relink        ;*adjust links*
HO  3370 ;******print ,messages*************
ID  3380 errmsg     =    *
HC  3390 .asc  "to —hange string : "
LG  3400 .byte 13 ;*carriage return*
AA  3410 .asc  " c,<oldstring>,<newstring>
GF  3420 errmsgx    =    *
JM  3430 changes    =    *
IO  3440 .asc  "  alterations made to source "
DJ  3450 changesx   =    *
EG  3460 ;********valid delimiters**********
BN  3470 express    =    *
AI  3480 .asc  "#" ;*asci values*
IE  3490 .asc  ","
BG  3500 .asc  ";"
KG  3510 .asc  ":"
DG  3520 .asc  ")"
MG  3530 .asc  "("
NH  3540 .asc  "/"
FI  3550 .asc  "-"
MI  3560 .asc  "*"
HJ  3570 .asc  " + "
CL  3580 .asc  "<"
OL  3590 .asc  ">"
HM  3600 .asc  " = "
CH  3610 .asc  "  "
DK  3620 .byte $aa ; " + "  *as basic tokens*
GP  3630 .byte $ab ; "-"
FP  3640 .byte $ac ; "*"
EB  3650 .byte $ad ; "/"
LE  3660 .byte $b1 ; ">"
CF  3670 .byte $b2 ; " = "
JF  3680 .byte $b3 ; "<"
OB  3690 .byte 0 ; end of line
OL  3700 .byte 1 ; begin of line
HH  3710 expressx   =    *
```

An R65C02 Assembler

Gary L. Anderson
Cedar Rapids, IA

. . .a pin for pin direct plug–in replacement for the NMOS 6502

Introduction

Many thanks to Eric Brandon for his fine article in the June 1981 issue of COMPUTE! entitled "Assembler in Basic for the PET". Some of the reasons he presented at that time for writing the program fit my situation quite well. I didn't want to spend $200 for an assembler and because I was new to machine language I wanted something simple I could dabble with but would still calculate branches and jumps.

Based on the knowledge gained from the article, I wrote the following assembler and as time went by I found one thing or another to add, change, or upgrade. The assembler was written in BASIC and compiled using PET SPEED (you can get the compiled version from the Transactor disk for this issue), and its main feature is the inclusion of the new op codes for the recently announced CMOS version of the 6502 microprocessor. Other improvements include adding cassette file and disk file storage of source code, linker disk file capability, full commenting capability of source files and linker files, find symbol command, "BYT" mnemonic, and finally "<" and ">" operand specifiers for labels and equates.

What is an R65C02?

It is a pin for pin direct plug–in replacement for the NMOS 6502 with its manufacturing process being CMOS (Complementary Metal Oxide Semiconductor). As a matter of fact I wrote this article on my 4032 PET with an R65C02 plugged in to replace the NMOS version. Its power consumption is advertised at being only four mA per MHZ with up to four MHZ versions specified on Rockwell's data sheet. This is obviously not much of a concern in the PET with an ample power supply and a clock of one MHZ but holds interesting possibilities for battery operated high–speed portable computers.

The most important improvement of this CMOS version is the expanded op code set with all of the existing op codes of the NMOS version to result in a powerful instruction set. The new op codes fall into two categories: new instructions, and added addressing modes of old instructions. Figure 1 shows all of the new op codes in alphabetical order of their mnemonic. There are 45 op codes covering 12 new instructions and 14 op codes covering new addressing modes of 11 old instructions for a total of 59 new op codes.

Another improvement is the determination of undefined op codes. They are now all no ops but beware, some are two byte and some three byte no ops of various machine cycle lengths. If one specifies only one byte of a two byte no op or two bytes of a three byte no op a crash will result. One byte no ops are at X3 and XB, two byte no ops at 02, 22, 42, 62, 82, C2, E2, 44, 54, D4, and F4, and three byte no ops at 5C, DC, and FC. The multiple byte no ops from my experimentation need not be all the same byte, it appears that the "filler bytes" after the first byte can be anything.

Other improvements are covered in my second reference and include incrementing the page address for the indirect JMP when the two operand bytes fall on either side of a page boundary, eliminating a read cycle of an invalid address and instead reading the last instruction byte during indexed addressing across a page boundary, changing Read/Modify/Write instructions from one read and two write cycles to two read and one write cycle and changing the decimal flag from indeterminate after reset to initialized to binary mode after reset and interrupts. Also in the NMOS version the N,V and Z flags were invalid after a decimal operation; they are valid in the CMOS version. Finally, if an interrupt occurs after the fetch of a BRK instruction the NMOS version will igrore the BRK vector and load the interrupt vector. The CMOS version will execute the BRK, then execute the interrupt.

The New Instructions

BBR(0–7) and BBS(0–7) will branch on bit reset and branch on bit set respectively. These are three byte instructions. The addressing mode is defined as zero page because the particular bit being analyzed resides in the zero page location defined by the second byte. The third byte is the branch byte. The fourth character in the mnemonic identifies the bit that will be analyzed.

BRA will branch always. No more does one have to use a three byte JMP to hop around a few bytes of code. If the destination is within 255 bytes then use BRA and save a byte because it only takes two bytes.

PHX, PHY, PLX, PLY will let a programmer push or pull X and Y registers to or from the stack directly. Transferring to or from the accumulator is no longer necessary when saving and restoring these two registers.

RMB(0–7) and SMB(0–7) will reset memory bit and set memory bit respectively. These are two byte instructions. The addressing mode is defined as zeropage because the particluar bit being changed resides in the zero page location defined in the second byte. The fourth character in the mnemonic identifies the bit that will be changed. These two instructions combined with BBR(0–7) and BBS(0–7) provide for very easy bit manipulation and conditional branching on bit logic states.

STZ will store zero to any location. No longer does one have to load the accumulator with #$00 then store the accumulator not to mention the possibility of having to save away the original contents of the accumulator and restoring it. Four addressing modes are provided, absolute, zero page, absolute,X and zero page,X.

TRB and TSB are called test and set bits and test and reset bits respectively. With these instructions one can reset or set any one or all bits of any location. Before executing these instructions the accumulator must be loaded with a byte representing the bits to be reset or set. Example: reset the high order nybble of location $1234.

```
LDA #$F0
TRB $1234
```

Both absolute and zero page addressing modes are provided.

New Addressing Modes Of Existing Instructions

As can be seen in Figure 1 of the 14 new op codes covering existing instructions eight are indirect addressing, those being ADC, AND, CMP, EOR, LDA, ORA, SBC, and STA. JMP has been given a new addressing mode being (Indirect,X). BIT has three new ones, Immediate, Zero Page,X and Absolute,X. Last but surely not least is accumulator addressing for DEC and INC. Many times have I wanted to decrement or increment the accumulator and now I can.

About The Program

The program listing shown takes 10,031 bytes* exactly. This is a two pass assembler, the first pass to generate a symbol table and the second pass to generate the object code. The two POKEs in line one set the top of memory and top of strings to reserve the upper 4,096 bytes of ram ($7000 to $7FFF) for object code assuming 32K of RAM total. The first line of an assembly would then be:

```
* = $7000 ; sys28672.
```

For a 16k PET change the two POKEs to whatever is convenient. For 1,024 bytes of reserved RAM, POKE in a value of 60 instead of 112 that is shown. The first line of an assembly would then become:

```
* = $3c00 ; sys15360
```

Also reducing the values of 'MEM' and 'M2' might be necessary. 'MEM' sets the total number of lines permitted per source module and 'M2' the total number of equates and labels that can be stored in the symbol table. The compiled version mentioned earlier will not fit into a 16K PET since it takes just over 21K bytes of memory.

For large assemblies containing lots of comments that would normally eat up tons of RAM, linker disk file capability has been added. Instead of writing one large source file one can break it up into smaller "modules" that reside on disk. The linker file then brings in each module one at a time, first to generate a symbol table, and again to generate the object code. Standard BASIC 2.0 disk commands are used, so the program will run on the C64 as well as BASIC 4.0 machines. No provision for a linked assembly was made for the cassette. For those wanting to modify the program a detailed breakdown on program structure is given in Figure 2.

Program Operation

MENU Upon running the program the menu appears and the mode of operation automatically defaults to "source" which appears in reverse video. Source mode, and the alternate Linker mode can be toggled by hitting return without making a selection from the menu. To make a selection type in the letter shown in reverse video of the operation wanted. If an illegal selection is made a new menu is displayed.

INPUT When entering this operation the program will ask for a starting line number; enter it and hit return. If starting a new machine language program or module always start with line number one. Although line number zero technically does exist do not use it because it is used by the program to keep track of the number of total lines used when writing or reading source or linker files to cassette or disk.

The line number will be displayed at the beginning of a new line and a non–blinking cursor will be displayed. If a line is to be comments only then hit a semicolon immediately after the line number. The semicolon tells the program to begin a comment field and can only be used at the beginning of a comment field. A full line of comments in source mode is 71 characters long and in linker mode is 73 characters long. If an error is made and it is caught before spacing to the next word or field then use the delete key to back up otherwise hit return to start a new line and type "FIX" and return. This will let the previous line be re-entered. To get back to the menu from INPUT just type "EXIT" on a new a line and hit return.

When inputting a linker file, if the first character in a line is not a semicolon then a 16 character file name field is enabled and should contain the exact file name of a file already on disk or one that will be on disk. If the file name is less than 16 characters and some commenting is necessary, just hit a semicolon and the program will automatically tab out to the comment field and display the semicolon and a non–blinking cursor. The comment field after the file name field is 56 characters long.

When inputting a source file, if the first character in a line is not a semicolon then a six character symbol field is enabled for labels or equates. The first character of a label must fall in the alphabet from A to Z. If the symbol is less than six characters then hit the space bar once to automatically tab to the four character mnemonic field. If no symbol is to be entered on a particular line then immediately hit the space bar once to automatically tab to the mnemonic field.

As just mentioned the mnemonic field is four characters long to hold the new mnemonics of the R65C02. There are two nonmnemonic operators that can be used in this field, an equals "=" sign for equates and "BYT" for entering data. Use immediate addressing mode when using BYT. If the entry in the mnemonic field is less than four characters then hit the space bar once to automatically tab to the operand field. If no operand or comments are to be entered on a particular line then instead of spacing to the operand field just hit return to terminate the line.

The operand field is a total of 13 characters in length. Again, if specifying a label or equate, the first character must be in the range from A to Z. If it is a number then the program will take that number as the value of the operand and not look it up in the symbol table. The operand specifiers include "#" for immediate addressing, "$" for hexadecimal, and "%" for binary. The specifiers "+", ">", and "<" can only be used in combination with labels and equates and are used as follows: "+" to increment the value of the operand, ">" to specify the hi byte of the address containing the operand, and "<" to specify the lo byte of the address containing the operand. These latter three specifiers can not be used in combination with one another on the same line. One can however use as many "+" specifiers on the same line after an operand within the field.

If comments are to be added after the operand field and the operand field has not been exited by using all 13 characters then hit the semicolon and the program will automatically tab to the comment field and display a semicolon followed by the non-blinking cursor. If the cursor is in the mnemonic field hit a space to enter the operand field then a semicolon to enter the comment field. To terminate the line just hit return. The comment field after the operand field is 31 characters long.

DELETE The program will ask for "FROM,TO" line numbers. When deleting just one line enter that line number for both from and to. Delete operates the same for both source and linker modes.

INSERT To add blank lines in the middle of existing source or linker files use the insert command. A prompt will ask for the first line and number of lines. The first line is the line that is to be "pushed down". Example: to insert three blank lines between lines 7 and 8 answer the prompt with 8,3 and return. Keep in mind that if the array, be it source or linker, is completely full then in the above example three lines will be "thrown off" the end of the array. To enter information in these three blank lines just go back to INPUT.

LIST To list lines to the screen make this selection and answer the prompt asking for "FROM,TO" line numbers. If you specify the end line as higher than the last line, the LIST routine will continue showing empty lines – hit any key to terminate the LISTing. No provision was made to list source code to the printer. Listing to the printer can be done during assembly.

WRITE FILE To save a source or linker file to tape or disk enter a "W" from the menu and answer the prompts. The program will ask first if the media is tape or disk. Secondly it will ask for a name for the file and in the case of disk will also ask for a drive number. When all the prompts are answered the program then begins from the end of the array and looks for a line with any information in it to determine the size of the array. When it finds an element in the array that is not a null string it places that number into element zero and then saves the elements from zero to the last line in a sequential format. When writing a file to disk the disk status and file name is displayed on the screen before returning to the menu.

READ FILE Being the sister command to WRITE FILE, READ FILE is similar with minor difference. A prompt will ask for the type of media and when reading in a file from tape no file name is asked for. When the length of the file is determined by reading element zero off the tape each line of code is displayed on the screen. Waiting for a large array to come in from tape is rather boring and watching it lumber in helps break the monotony.

When reading in a file from disk, a prompt will ask for the file name and drive number. The program then reads in element zero, determines the length, and reads in the rest of the array. When reading in a file from disk, the disk status and file name is displayed on the screen before returning to the menu.

After a file is read in all elements of the array after the last line number are set to a null string. A previous file could have been longer than the one just read in and if making minor edits and re–saving or assembling those extra lines would cause unwanted results.

ASSEMBLE The first prompt to be answered asks if the listing destination is the screen or printer. When assembling source code for the first time I always dump to the screen because of the possibility of errors. When in source file mode no more prompts are asked for. When assembling a single source file the code must already reside in RAM by either INPUT or READ FILE. When assembling in linker file mode a prompt will ask for the name of the linker file and drive number. No files are needed in RAM, they must all reside on the same disk including the linker file. The disk status and file name are displayed on the screen for each file fetch.

There are a number of error messages that will inform the operator of the reason of an aborted assembly. I was amazed to discover how easy it is to use an equate name or label twice in a large linked assembly so I included some checking and a response of "DUPLICATE SYMBOL ERROR" if it happens again. Other error messages include "UNDEFINED SYMBOL ERROR", "ILLEGAL MNEMONIC", "ILLEGAL ADDRESSING MODE", and "TOO LONG CONDITIONAL BRANCH". If an error occurs during an assembly the program will stop, display the error, and go back to the menu. The source file with the problem is left in ram so just go back to the source file input mode and make the fixes. Do not forget to save the fixed file to disk in the case of a linked assembly.

FIND SYMBOL If an assembly stops with a "DUPLICATE SYMBOL ERROR" or if the necessity arises to find out if a particular equate or label name has already been used, then use the find label command. A prompt will ask for the symbol name and if in linker mode, will also ask for the linker file and drive number. When finding a symbol in linker mode, the screen will display the source file name with the line number and line of code for every match found. When in source mode then just the line number and line of code will be displayed for each match found in RAM.

QUIT When done hit "Q" from the menu and the BASIC version will respond with "GET BACK IN WITH GOTO650". The compiled version omits this message because there is no line 650 to go to. When using "GOTO 650" to get back in the BASIC version existing variables and source code will be preserved from the previous RUN.

Examples

Eric Brandon included a machine language test program in his article that wrote every character to every location on the screen and is duplicated here in Figure 3A. Compare the original version, in this case assembled at $7000, with a slightly improved version that takes advantage of some new op codes with full comments in Figure 3B. This new version will only run with an R65C02 plugged into the PET. Memory consumption of the new routine's object code has been cut by approximately one third, mostly because of being able to branch on bit status instead of comparing bytes and being able to increment the accumulator instead of a memory location. I have shown in the comments the total length of the field widths. A full length comment line can accept 71 characters and comments after the operand field can hold 31 characters.

An example of a linker file is shown in Figure 4. Note that the same commenting capability is available when writing a linker file. A full length comment line can accept 73 characters and comments after the file name can hold 56 characters. Every source file specified in a linker file must reside on the same disk as the linker file. The total number of lines a linker file can hold is 64 which should be plenty, however if this quantity needs to be changed then adjust the variable 'LK' in line one.

Final Odds and Ends

I found that when compiling the assembler with PET SPEED, the NMOS 6502 must be reinstalled in the PET. Apparently the designers of PET SPEED used an undefined op-code that only the NMOS version can decode properly or used the problem in the indirect JMP to effectively encrypt their compiler, at any rate during the compilation PET SPEED goes off into never never land with the CMOS 6502 in place. However, an already compiled program works just fine with an R65C02 plugged in. (The compiled version is included on Transactor Disk #6)

GTE's G65SC02 does not have the bit manipulation instructions nor the branch on bit instructions so don't try to use those instructions on GTE's chip.

References:

1. Eric Brandon, "Assembler in BASIC for the PET", COMPUTE!, June 1981.
2. Rockwell International Corporation, Semiconductor Products Division, "R65C02", 1984 DATA BOOK.

Gary L. Anderson
1528 34th St. S.E.
Cedar Rapids, IOWA
52403

Editors Note (and *)

The program may no longer be 10,031 bytes long. Gary's original used BASIC 4.0 disk commands but they've since been changed to their equivalent BASIC 2.0 counterparts. Also, additional tests for ST and disk status were inserted following OPEN commands, etc., to counter the automatic test performed by DOPEN. The reasons? Although you can't plug an R65C02 into a Commodore 64, the assembler will still work, as long as you don't try to use any of the extra op-codes (unless you intend to transport them to an R65C02 machine).

The first two POKEs will naturally have to be changed. Add 3 to both addresses to assemble programs at $7000. However, the 64 has 8 more K of memory above $8000. You can move the pointer up some, or if you're planning to assemble programs in RAM at $C000, you can omit these two POKEs altogether.

Gary has structured his storage files with a somewhat familiar format. A quick look with Superscript shows that Gary's files are so close to the CBM Assembler format that it's hard to imagine them being incompatible. Source lines are written to the file without line numbers. Shifted spaces are used to separate fields and extra shifted spaces indicate blank fields. The only difference that will need attention is the very first line which indicates the total number of lines in the file. If this is deleted, the source files from Gary's assembler should be totally compatible with the Commodore Assembler. PAL users will find a CBM to PAL conversion program included with the PAL package. Both of these assemblers are written in machine language making them somewhat faster. Should you become more akin to machine code and decide to get either of these assemblers, your existing source files will only need a little work to be transportable.

Note: The listings shown have been generated by the Assemble option of Gary's assembler. The first lines output are the label equates that were defined within the source code. To enter these programs, start with line 1 and continue from there, omitting the assembled address and hex values. Remember, the SPACE bar moves the cursor to the next field and each field must contain the proper type of data (ie. labels in the 1st field, mnemonics in the 2nd field, operands in the 3rd field, comments in the 4th field, etc.)

Figure 2. Program Structure Breakdown.

Lines	Function
10–20	Initialization
30–120	Separate lables, op codes, and operands
130–150	Separate source file name from comments
160–360	Determine the value of the operand
370–380	Find the length of the source array
390	Clean out the remainder of the source array
400–410	Find the length of the linker array
420	Clean out the remainder of the linker array
430–570	Data statements for op codes
580–600	Convert hex to decimal
610	Convert binary to decimal
620–640	Convert decimal to hex
650–740	Print menu prompt, get command
750–780	Input lines executive
790–850	Delete lines
860–920	Insert lines
930–1070	List to screen
1080–1230	Write file to tape or disk
1240–1410	Read file from tape or disk
1420–1620	Assemble executive
1630–1820	Find symbol
1830–1990	First pass, create symbol table
2000–2580	Second pass, assemble source code
2590–2750	Input source file
2760–2940	Input linker file
2950–3080	Input from keyboard subroutine

Figure 1 The new op codes in order of their mnemonic. In comment field (1) denotes new instruction and (2) denotes new addressing mode of old instruction.

```
absolu = $1111    immedi = $0022    ind    = $0033    zeropg = $0044
offset = $7000
1    ; new op codes on the r65c02
2                    *       =    $7000
3                    absolu  =    $1111
4                    immedi  =    $22
5                    ind     =    $33
6                    zeropg  =    $44
7   7000  72 33      offset  adc  (ind)           ; (2)
8   7002  32 33              and  (ind)           ; (2)
9   7004  0f 44 f9           bbr0 zeropg,offset   ; (1) branch on bit 0 reset
10  7007  1f 44 f6           bbr1 zeropg,offset   ; (1) branch on bit 1 reset
11  700a  2f 44 f3           bbr2 zeropg,offset   ; (1) branch on bit 2 reset
12  700d  3f 44 f0           bbr3 zeropg,offset   ; (1) branch on bit 3 reset
13  7010  4f 44 ed           bbr4 zeropg,offset   ; (1) branch on bit 4 reset
14  7013  5f 44 ea           bbr5 zeropg,offset   ; (1) branch on bit 5 reset
15  7016  6f 44 e7           bbr6 zeropg,offset   ; (1) branch on bit 6 reset
16  7019  7f 44 e4           bbr7 zeropg,offset   ; (1) branch on bit 7 reset
17  701c  8f 44 e1           bbs0 zeropg,offset   ; (1) branch on bit 0 set
18  701f  9f 44 de           bbs1 zeropg,offset   ; (1) branch on bit 1 set
19  7022  af 44 db           bbs2 zeropg,offset   ; (1) branch on bit 2 set
20  7025  bf 44 d8           bbs3 zeropg,offset   ; (1) branch on bit 3 set
21  7028  cf 44 d5           bbs4 zeropg,offset   ; (1) branch on bit 4 set
22  702b  df 44 d2           bbs5 zeropg,offset   ; (1) branch on bit 5 set
23  702e  ef 44 cf           bbs6 zeropg,offset   ; (1) branch on bit 6 set
24  7031  ff 44 cc           bbs7 zeropg,offset   ; (1) branch on bit 7 set
25  7034  89 22              bit  #immedi         ; (2)
26  7036  34 44              bit  zeropg,x        ; (2)
27  7038  3c 11 11           bit  absolu,x        ; (2)
28  703b  80 c3              bra  offset          ; (1) branch always
29  703d  d2 33              cmp  (ind)           ; (2)
30  703f  3a                 dec  a               ; (2)
31  7040  52 33              eor  (ind)           ; (2)
32  7042  1a                 inc  a               ; (2)
33  7043  7c 33 00           jmp  (ind,x)         ; (2)
34  7046  b2 33              lda  (ind)           ; (2)
35  7048  12 33              ora  (ind)           ; (2)
36  704a  da                 phx                  ; (1) push x onto stack
37  704b  5a                 phy                  ; (1) push y onto stack
38  704c  fa                 plx                  ; (1) pull x from stack
39  704d  7a                 ply                  ; (1) pull y from stack
40  704e  07 44              rmb0 zeropg          ; (1) reset memory bit 0
41  7050  17 44              rmb1 zeropg          ; (1) reset memory bit 1
42  7052  27 44              rmb2 zeropg          ; (1) reset memory bit 2
43  7054  37 44              rmb3 zeropg          ; (1) reset memory bit 3
44  7056  47 44              rmb4 zeropg          ; (1) reset memory bit 4
45  7058  57 44              rmb5 zeropg          ; (1) reset memory bit 5
46  705a  67 44              rmb6 zeropg          ; (1) reset memory bit 6
47  705c  77 44              rmb7 zeropg          ; (1) reset memory bit 7
48  705e  f2 33              sbc  (ind)           ; (2)
49  7060  87 44              smb0 zeropg          ; (1) set memory bit 0
50  7062  97 44              smb1 zeropg          ; (1) set memory bit 1
51  7064  a7 44              smb2 zeropg          ; (1) set memory bit 2
52  7066  b7 44              smb3 zeropg          ; (1) set memory bit 3
53  7068  c7 44              smb4 zeropg          ; (1) set memory bit 4
54  706a  d7 44              smb5 zeropg          ; (1) set memory bit 5
55  706c  e7 44              smb6 zeropg          ; (1) set memory bit 6
56  706e  f7 44              smb7 zeropg          ; (1) set memory bit 7
57  7070  92 33              sta  (ind)           ; (2)
58  7072  9c 11 11           stz  absolu          ; (1) store zero
59  7075  9e 11 11           stz  absolu,x        ; (1) store zero
60  7078  64 44              stz  zeropg          ; (1) store zero
61  707a  74 44              stz  zeropg,x        ; (1) store zero
62  707c  1c 11 11           trb  absolu          ; (1) test and reset bits
63  707f  14 44              trb  zeropg          ; (1) test and reset bits
64  7081  0c 11 11           tsb  absolu          ; (1) test and set bits
65  7084  04 44              tsb  zeropg          ; (1) test and set bits
```

Figure 3A. The old NMOS version of writing every character to the screen.

```
char   = $7040       scrn   = $00b5      again  = $7008       loop   = $700f
                  *      =   $7000
                  char   =   $7040
                  scrn   =   $b5
7000  a9 00       lda    #$00
7002  a8                 tay
7003  8d 40 70           sta    char
7006  85 b5              sta    scrn
7008  a9 80     again    lda    #$80
700a  85 b6              sta    scrn+
700c  ad 40 70           lda    char
700f  91 b5     loop     sta    (scrn),y
7011  c8                 iny
7012  d0 fb              bne    loop
7014  e6 b6              inc    scrn+
7016  a6 b6              ldx    scrn+
7018  e0 84              cpx    #$84
701a  d0 f3              bne    loop
701c  ee 40 70           inc    char
701f  d0 e7              bne    again
7021  60                 rts
```

Figure 3B. The R65C02 version with full comments.

```
scrn   = $0020       again  = $7007       loop   = $7009
; 1234567890123456789012345678901234567890123456789012345678901
; ****************************************************
; *
; * new r65c02 version
; *
; * this is a demo of the speed of machine language graphics.
; * this subroutine fills the screen with every possible character.
; *
; * $7000 = 28672
; *
; ****************************************************
                  *      =   $7000          ; 1234567890123456789012345678901
                  scrn   =   $20            ; temporary loc of scrn data
7000  a9 00              lda    #0          ; clear accumulator
7002  a8                 tay                ; clear y
7003  64 b5              stz    scrn        ; clear lo byte of screen address
7005  a2 80              ldx    #$80        ; hi byte of screen address
7007  86 b6     again    stx    scrn+       ; reset hi byte of screen address
7009  91 b5     loop     sta    (scrn),y    ; write char to screen location
700b  c8                 iny                ; point to next screen location
700c  d0 fb              bne    loop        ; are 256 done? if not branch
700e  e6 b6              inc    scrn+       ; yes, point to next 256
7010  2f b6 f6           bbr2   scrn+,loop  ; have 1024 locatns been filled?
7013  1a                 inc    a           ; yes, next character
7014  d0 f1              bne    again       ; have all chars been written?
7016  60                 rts                ; yes back to basic
```

Figure 4. Linker file example. Lines 1 and 11 show the maximum length of the commenting capability. Line 11 also shows the maximum length of the file name.

```
1    ; 12345678901234567890123456789012345678901234567890123456789012 3
2    ; ****************************************************
3    ; *
4    ; * this is an example of a linker file. each file name must be on the
5    ; * same disk as the linker file.
6    ; *
7    ; ****************************************************
8    file one                ; first file to be pulled in
9    file two                ; second file
10   file three              ; third file
11   1234567890123456        ; 12345678901234567890123456789012345678901234567890123456
```

The R65C02 Assembler

```
LD   10 poke53,112:poke49,112:mem=500
        :m2=300:lk=64:m$="s":open15,8,15
FB   20 print"S":dima$(mem),s$(m2),v(m2),l(3),l$(lk)
        :h$="0123456789abcdef":goto650
EO   30 input#15,e,e$,e1,e2:e$=str$(e)+", "+e$+","
        +str$(e1)+","+str$(e2)
EE   40 return
CC   50 l(1)=0:l(2)=0:l(3)=0:l=0
PM   60 forq=1tolen(a$(t)):ifmid$(a$(t),q,1)=chr$(160)
        thenl=l+1:l(l)=q
HF   70 ifl<3thennextq
GM   80 lb$=left$(a$(t),l(1)-1)
DF   90 ifl(2)=0thenoc$=right$(a$(t),q-l(1)-1):op$="":return
MP  100 oc$=mid$(a$(t),l(1)+1,l(2)-l(1)-1)
HB  110 ifl(3)=0thenop$=right$(a$(t),q-l(2)-1):return
IJ  120 op$=mid$(a$(t),l(2)+1,l(3)-l(2)-1):return
DL  130 nm$="":cm$="":forq=1tolen(l$(j))
        :z$=mid$(l$(j),q,1)
ME  140 ifz$=";"thennm$=left$(l$(j),q-2)
        :cm$=right$(l$(j),len(l$(j))-q+1):return
MA  150 nextq:nm$=l$(j):return
HJ  160 ad=0:lo=0:hi=0
FP  170 q$=left$(o1$,1):q1=asc(q$)
DE  180 if(q1>47andq1<58)or(q1>64andq1<91)
        orq$="$"orq$="%"then200
KA  190 o1$=right$(o1$,len(o1$)-1):goto170
EB  200 q$=right$(o1$,1):q1=asc(q$)
PK  210 if(q1>47andq1<58)or(q1>64andq1<91)
        orq$="+"orq$="<"orq$=">"then230
NA  220 o1$=left$(o1$,len(o1$)-1):goto200
NL  230 ifright$(o1$,2)=chr$(108)+"x"
        theno1$=left$(o1$,len(o1$)-2):goto250
PF  240 ifright$(o1$,2)=chr$(108)+"y"
        theno1$=left$(o1$,len(o1$)-2)
IL  250 ifright$(o1$,1)=")"theno1$=left$(o1$,len(o1$)-1)
LK  260 ifleft$(o1$,1)="$"thenn$=right$(o1$,len(o1$)-1)
        :gosub580:nu=v:return
DL  270 ifleft$(o1$,1)="%"thenn$=right$(o1$,len(o1$)-1)
        :gosub610:nu=v:return
GN  280 ifasc(left$(o1$,1))<58thennu=val(o1$):return
BC  290 ifright$(o1$,1)="+"theno1$=left$(o1$,len(o1$)-1)
        :ad=ad+1:goto290
OE  300 ifright$(o1$,1)="<"theno1$=left$(o1$,len(o1$)-1)
        :lo=1:goto320
HB  310 ifright$(o1$,1)=">"theno1$=left$(o1$,len(o1$)-1)
        :hi=1
EA  320 forw=1tom2:ifs$(w)=o1$thennu=v(w):w=999
NK  330 nextw:ifw=m2+1thenprint"rundefined symbol
         error":goto650
LA  340 iflothenn=nu:gosub620:n$=right$(r$,2)
        :gosub580:nu=v:return
OO  350 ifhithenn=nu:gosub620:n$=left$(r$,2)
        :gosub580:nu=v:return
HC  360 nu=nu+ad:return
LP  370 forll=memto1step-1:ifa$(ll)<>""
        thena$(0)=str$(ll):ll=1
KO  380 nextll:return
IC  390 fori=val(a$(0))+1tomem:a$(i)="":nexti:return
CK  400 forll=lkto1step-1:ifl$(ll)<>""thenl$(0)=str$(ll):ll=1
IA  410 nextll:return
EF  420 fori=val(l$(0))+1tolk:l$(i)="":nexti:return
PC  430 dataadcn6ds65i69k7dl79p61o71q75m72,andn2
        ds25i29k3dl39p21o31q35m32
DO  440 dataaslh0an0es06k1eq16,bbr00f11f22f33f44f55f66
        f77f
EA  450 databbs08f19f2af3bf4cf5df6ef7ff,bccj90,bcsjb0,
        beqjf0,bitn2cs24i89k3cq34
LI  460 databmij30,bnejd0,bplj10,braj80,brkg00,bvcj50,
        bvsj70,clcg18,cldgd8,clig58
```

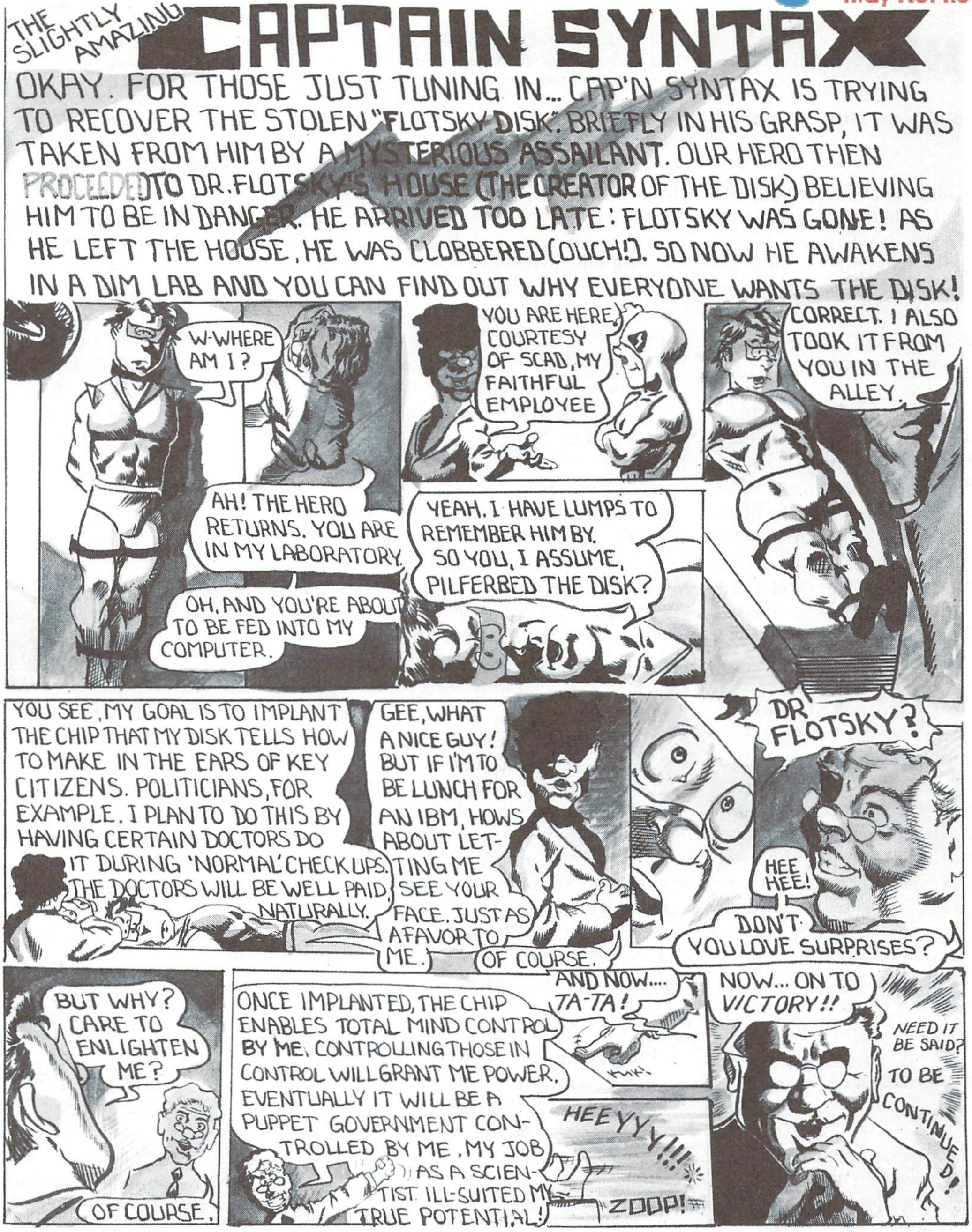
```
DF  470 dataclvgb8,cmpncdsc5ic9kddld9pc1od1qd5md2,
        cpxnecse4ie0,cpynccsc4ic0
OA  480 datadecncesc6kdeqd6h3a,dexgca,deyg88,
        eorn4ds45i49k5dl59p41o51q55m52
NF  490 dataincneese6kfeqf6h1a,inxge8,inygc8,
        jmpn4cm6cp7c,jsrn20
CN  500 dataldanadsa5ia9kbdlb9pa1ob1qb5mb2,
        ldxnaesa6ia2lberb6,ldynacsa4ia0kbcqb4
LG  510 datalsrh4an4es46k5eq56,nopgea,
        oran0ds05i09k1dl19p01o11q15m12,phag48,phpg08
EG  520 dataphxgda,phyg5a,plag68,plpg28,plxgfa,
        plyg7a,rmb00711722733744755766777
DD  530 datarolh2an2es26k3eq36,rorh6an6es66k7eq76,
        rtig40,rtsg60
FE  540 datasbcnedse5ie9kfdlf9pe1of1qf5mf2,secg38,
        sedgf8,seig78
OM  550 datasmb0871972a73b74c75d76e77f7,
        stan8ds85k9dl99p81o91q95m92
IO  560 datastxn8es86r96,styn8cs84q94,stzn9cs64q74k9e,
        taxgaa,tayga8,trbn1cs14
PA  570 datatsbn0cs04,tsxgba,txag8a,txsg9a,tyag98
OB  580 v=0:iflen(n$)<4thenn$=left$("0000",4-len(n$))+n$
JM  590 forr2=1to4:d$=mid$(n$,r2,1):tv=asc(d$)-48
        :iftv>9thentv=tv-7
KI  600 v=tv*16↑(4-r2)+v:nextr2:return
GM  610 v=0:forz=len(n$)to1step-1
        :v=v+val(mid$(n$,z,1))*2↑(len(n$)-z):nextz:return
NH  620 fd=int(n/4096):n=(n/4096-fd)*4096
        :sd=int(n/256):n=(n/256-sd)*256
FM  630 td=int(n/16):n=int((n/16-td)*16)
        :r$=mid$(h$,fd+1,1)+mid$(h$,sd+1,1)
GM  640 r$=r$+mid$(h$,td+1,1)+mid$(h$,n+1,1):return
PF  650 ifm$="s"thenprint"rsource fileR";
MC  660 ifm$="l"thenprint"rlinker fileR";
MI  670 print"[3 spaces]rrRead or rwRrite file"
FO  680 print"riRnput rdRelete irnRsert rlRist"
AI  690 print"raRssemble rqRuit rfRind symbol
         command? ";
PB  700 getc$:ifc$=""then700
FG  710 ifc$=chr$(13)andm$="s"thenm$="l"
        :print"QQQ":goto650
IG  720 ifc$=chr$(13)andm$="l"thenm$="s"
        :print"QQQ":goto650
FG  730 printc$
GB  740 ifc$="q"thenprint"qget back in with
        rgoto650R":end
FC  750 ifc$<>"i"then790
MD  760 input"qline";ln
FF  770 ifm$="s"andln>0andln<=memthen2590
JH  780 ifm$="l"andln>0andln<=lkthen2760
AD  790 ifc$<>"d"then860
IE  800 input"qlines - from,to";fl,ll
MI  810 ifm$="l"then840
KI  820 fort=fltoll:a$(t)="":nextt
GC  830 fort=lltomem-1:a$(t-ll+fl)=a$(t+1)
        :a$(t+1)="":nextt:goto650
PL  840 fort=fltoll:l$(t)="":nextt
CO  850 fort=lltolk-1:l$(t-ll+fl)=l$(t+1)
        :l$(t+1)="":nextt:goto650
FJ  860 ifc$<>"n"then930
ND  870 input"qlines - first,number";fl,ll
LM  880 ifm$="l"then910
GL  890 fort=mem-lltoflstep-1:a$(t+ll)=a$(t):nextt
DC  900 fort=fltofl+ll-1:a$(t)="":nextt:goto650
DK  910 fort=lk-lltoflstep-1:l$(t+ll)=l$(t):nextt
DG  920 fort=fltofl+ll-1:l$(t)="":nextt:goto650
HB  930 ifc$<>"l"then1080
MA  940 input"qlines - from,to";fl,ll:print"q"
KD  950 ifm$="l"then1030
KG  960 fort=fltoll:getc$:ifc$<>""thent=ll:goto1020
HE  970 iflen(a$(t))=0thenprintt:forel=1to250:next:goto1020
```

```
MP  980 ifleft$(a$(t),1) = " ; "thenprintttab(6)a$(t):goto1020
AE  990 gosub50:printttab(6)lb$tab(13)oc$tab(18)op$;
NH  1000 ifl(3)>0thenprinttab(32)right$(a$(t),len(a$(t))-l(3))
         :goto1020
EC  1010 print
ID  1020 nextt:goto650
DI  1030 forj = fltoll:getc$:ifc$<> "  "thenj = ll:goto1070
GG  1040 iflen(l$(j)) = 0thenprintj:forel = 1to250:next:goto1070
GC  1050 ifleft$(l$(j),1) = " ; "thenprintjtab(4)l$(j):goto1070
MN  1060 gosub130:printjtab(4)nm$tab(21)cm$
ME  1070 nextj:goto650
NN  1080 ifc$<> " w "then1240
EG  1090 print " q to r t R ape or r d R isk?  " ;
LG  1100 getc$:ifc$ = "  "then1100
CL  1110 printc$:ifc$<> " t "then1160
PG  1120 input " name of file " ;n$:open2,1,1,(n$)
JP  1130 ifm$ = " l "then1150
JH  1140 gosub370:fori = 0toval(a$(0)):print#2,a$(i):nexti
         :close2:goto650
MK  1150 gosub400:fori = 0toval(l$(0)):print#2,l$(i):nexti
         :close2:goto650
IJ  1160 ifc$<> " d "then650
EK  1170 input " q name of file,drive# " ;n$,d
         :open2,8,8,mid$(str$(d),2) + " : " + n$ + " ,s,w "
JP  1180 gosub30:ifethenprinte$ "    "n$:close2:goto650
MC  1190 ifm$ = " l "then1220
IF  1200 gosub370:fori = 0toval(a$(0)):print#2,a$(i):gosub30
         :ife<20ore = 50thennexti
EC  1210 gosub30:printe$ "    "n$:close2:goto650
FJ  1220 gosub400:fori = 0toval(l$(0)):print#2,l$(i):gosub30
         :ife<20ore = 50thennexti
ID  1230 gosub30:printe$ "    "n$:close2:goto650
PG  1240 ifc$<> " r "then1420
JC  1250 print " q from r t R ape or r d R isk?  " ;
PB  1260 getc$:ifc$ = "  "then1260
IF  1270 printc$:ifc$<> " t "then1340
NJ  1280 open2,1,0
DJ  1290 ifm$ = " l "then1320
KP  1300 input#2,a$(0):fori = 1toval(a$(0)):input#2,a$(i)
         :printi;a$(i)
MD  1310 nexti:close2:gosub390:goto650
GG  1320 input#2,l$(0):fori = 1toval(l$(0)):input#2,l$(i):printi;l$(i)
DE  1330 nexti:close2:gosub420:goto650
ME  1340 ifc$<> " d "then650
CP  1350 input " q name of file,drive# " ;n$,d
         :open2,8,8,mid$(str$(d),2) + " : " + n$
NK  1360 gosub30:ifethenprinte$ "    "n$:close2:goto650
ON  1370 ifm$ = " l "then1400
JD  1380 input#2,a$(0):fori = 1toval(a$(0)):input#2,a$(i)
         :gosub30:ife<20thennexti
EN  1390 gosub30:printe$ "    "n$:close2:gosub390:goto650
KJ  1400 input#2,l$(0):fori = 1toval(l$(0)):input#2,l$(i):gosub30
         :ife<20thennexti
FN  1410 gosub30:printe$ "    "n$:close2:gosub420:goto650
IO  1420 ifc$<> " a "then1630
FO  1430 print " q to r s R creen or r p R rinter?  " ;
CP  1440 getdv$:ifdv$ = "  "then1440
DP  1450 printdv$:ifdv$ = " s "thendv = 3:goto1480
NJ  1460 ifdv$ = " p "thendv = 4:goto1480
GH  1470 goto650
CH  1480 close1:open1,dv:sb = 1:ifm$ = " l "then1500
NJ  1490 gosub370:gosub1830:ln = 1:pc = og:gosub2000
         :close1:goto650
BP  1500 input " q name of linker file,drive# " ;l$,d
         :open2,8,8,mid$(str$(d),2) + " : " + l$
PD  1510 gosub30:ifethenprinte$ "    "l$:close2:goto650
CB  1520 input#2,l$(0):fori = 1toval(l$(0)):input#2,l$(i):gosub30
         :ife<20thennexti
PD  1530 gosub30:print " q " e$ "    "l$:close2:gosub420
DF  1540 forpa = 1to2:forj = 1toval(l$(0))
NO  1550 ifleft$(l$(j),1) = " ; "thengoto1620
```

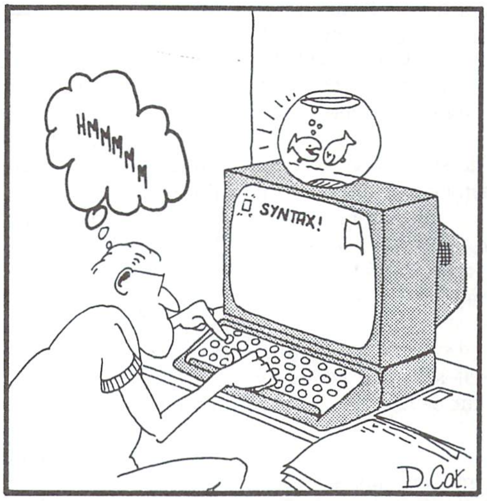
```
AF  1560 gosub130:open2,8,8,mid$(str$(d),2) + " : " + nm$
IO  1570 gosub30:ifethenprinte$ "    "nm$:close2:goto650
BA  1580 input#2,a$(0):fori = 1toval(a$(0)):input#2,a$(i)
         :gosub30:ife<20thennexti
JD  1590 gosub30:printe$ "    "nm$:close2:gosub390
BI  1600 ifpa = 1thengosub1830
FH  1610 ifpa = 2thengosub2000
LI  1620 nextj:ln = 1:pc = og:nextpa:close1:goto650
GH  1630 ifc$<> " f "then650
DI  1640 input " q symbol " ;la$
IB  1650 ifm$ = " l "then1670
LI  1660 gosub370:gosub1770:goto650
LJ  1670 input " q name of linker file,drive# " ;l$,d
         :open2,8,8,mid$(str$(d),2) + " : " + l$
JO  1680 gosub30:ifethenprinte$ "    "l$:close2:goto650
ML  1690 input#2,l$(0):fori = 1toval(l$(0)):input#2,l$(i):gosub30
         :ife<20thennexti
HG  1700 gosub30:print " q " e$ "    "l$:close2:gosub420
         :forj = 1toval(l$(0))
FJ  1710 ifleft$(l$(j),1) = " ; "thengoto1760
AP  1720 gosub130:open2,8,8,mid$(str$(d),2) + " : " + nm$
II  1730 gosub30:ifethenprinte$ "    "nm$:close2:goto650
BK  1740 input#2,a$(0):fori = 1toval(a$(0)):input#2,a$(i)
         :gosub30:ife<20thennexti
NO  1750 gosub30:printe$ "    "nm$:close2:gosub390
         :gosub1770
OP  1760 nextj:goto650
JC  1770 fort = 1toval(a$(0)):ifleft$(a$(t),1) = "    "
         orleft$(a$(t),1) = " ; "then1790
OI  1780 gosub50:ifla$ = lb$then1800
EH  1790 nextt:return
HL  1800 printttab(6)lb$tab(13)oc$tab(18)op$;
BL  1810 ifl(3)>0thenprinttab(32)right$(a$(t),len(a$(t))-l(3));
KC  1820 print:goto1790
MF  1830 fort = 1toval(a$(0)):ifleft$(a$(t),1) = " ; "then1950
AC  1840 gosub50:iflb$ = "  "then1890
AO  1850 ifoc$<> " = "then1880
KP  1860 o1$ = op$:gosub160:iflb$ = " * "thenpc = nu
         :og = nu:goto1950
AP  1870 s$(sb) = lb$:v(sb) = nu:n = nu:gosub620
         :gosub1970:goto1950
GF  1880 s$(sb) = lb$:v(sb) = pc:n = pc:gosub620:gosub1970
BE  1890 ifop$ = "  "orop$ = " a "oroc$ = " byt "
         thenpc = pc + 1:goto1950
PO  1900 ifright$(op$,1) = " < "orright$(op$,1) = " > "
         thenpc = pc + 2:goto1950
KA  1910 ifleft$(oc$,1) = " j "orleft$(oc$,2) = " bb "
         thenpc = pc + 3:goto1950
BG  1920 ifleft$(oc$,1) = " b "andoc$<> " bit "
         andoc$<> " brk "thenpc = pc + 2:goto1950
HP  1930 o1$ = op$:gosub160:ifnu<256thenpc = pc + 2
         :goto1950
FG  1940 pc = pc + 3
GG  1950 nextt:print#1:return
FP  1960 print#1,lb$spc(6-len(lb$)) "   =   $ "r$spc(6);
         :gosub1970:sb = sb + 1
FA  1970 print#1,lb$spc(6-len(lb$)) "   =   $ "r$spc(6);
BG  1980 forq = 0tosb-1:ifs$(q) = s$(sb)thenprint " r duplicate
          symbol error " :goto650
NH  1990 nextq:sb = sb + 1:return
GF  2000 er = 0:fort = 1toval(a$(0))
CE  2010 ifleft$(a$(t),1) = " ; "thenprint#1,
         lnspc(5-len(str$(ln)))a$(t);:goto2580
EC  2020 gosub50:ifoc$ = " = "thenmv$ = " [2 spaces] "
         :pc$ = " [4 spaces] " :il = 0:goto2530
KE  2030 ifop$ = "  "thenam$ = " g " :il = 1:goto2460
FC  2040 ifop$ = " a "thenam$ = " h " :il = 1:goto2460
LG  2050 ifoc$ = " byt "theno1$ = op$:gosub160:il = 1
         :mv$ = right$(n$,2):goto2510
DO  2060 ifleft$(oc$,2) = " bb "then2320
KC  2070 if mid$(oc$,2,2) = " mb "then2400
```

```
IH   2080 x = 0:y = 0:i = 0:m = 0:z = 0
FK   2090 forq = 1tolen(op$):q$ = mid$(op$,q,1):ifq$ = " ) "
          theni = 1:goto2110
PM   2100 ifq$ = " # " thenm = 1
EE   2110 nextq:forq = 1tolen(op$)-1:q$ = mid$(op$,q,2)
OK   2120 ifq$ = chr$(108) + " y " theny = 1:goto2140
NE   2130 ifq$ = chr$(108) + " x " thenx = 1
EL   2140 nextq:o1$ = op$:gosub160
CA   2150 ifleft$(oc$,1) = " j " then2180
IC   2160 ifleft$(oc$,1) = " b " andoc$<> " brk "
          andoc$<> " bit " then2430
NP   2170 ifnu<256then2240
NB   2180 ifiandxthenam$ = " p " :goto2230
MN   2190 ifxthenam$ = " k " :goto2230
LO   2200 ifythenam$ = " l " :goto2230
HM   2210 ifithenam$ = " m " :goto2230
NA   2220 am$ = " n "
KG   2230 so = int(nu/256):fo = (nu/256-so)*256:il = 3:goto2460
EO   2240 ifmthenam$ = " i " :goto2310
NF   2250 ifiandythenam$ = " o " :goto2310
KG   2260 ifiandxthenam$ = " p " :goto2310
DD   2270 ifxthenam$ = " q " :goto2310
CE   2280 ifythenam$ = " r " :goto2310
CB   2290 ifithenam$ = " m " :goto2310
MG   2300 am$ = " s "
AN   2310 fo = nu:il = 2:goto2460
NJ   2320 am$ = right$(oc$,1):o1$ = " " :o2$ = " " :il = 3
ID   2330 forq = 1tolen(op$)
KM   2340 ifmid$(op$,q,1) = chr$(108)theno1$ = left$(op$,q-1)
          :o2$ = right$(op$,len(op$)-q)
FC   2350 nextq
DA   2360 gosub160:ifnu>255thenprint " rillegal addressing
           mode " :goto650
HP   2370 fo = nu:o1$ = o2$:gosub160:ifnu>pc + 2
          thenso = nu-pc-3:ifso>127thener = 1
NP   2380 ifnu<pc + 2thenso = 253 + nu-pc:ifso<128thener = 1
DG   2390 goto2450
OD   2400 am$ = right$(oc$,1):il = 2:o1$ = op$:gosub160
CK   2410 ifnu>255thenprint " rillegal addressing mode "
          :goto650
LH   2420 fo = nu:goto2460
CG   2430 am$ = " j " :il = 2:ifnu>pc + 1thenfo = nu-pc-2
          :iffo>127thener = 1
IA   2440 ifnu<pc + 1thenfo = 254 + nu-pc:iffo<128thener = 1
CN   2450 ifer = 1thenprint " rtoo long conditional branch "
          :goto650
IL   2460 restore:forw = 1to68:readi$:ifleft$(i$,3) = left$(oc$,3)
          thenw = 70
CH   2470 nextw:ifw = 69thenprint " rillegal mnemonic "
          :goto650
GK   2480 forw = 4tolen(i$)step3:ifmid$(i$,w,1) = am$
          thenlw = w:w = 32
OG   2490 nextw:ifw<32thenprint " rillegal addressing mode "
          :goto650
ON   2500 mv$ = mid$(i$,lw + 1,2):n$ = mv$:gosub580
KK   2510 pokepc,v:ifil>1thenpokepc + 1,fo:ifil>2
          thenpokepc + 2,so
CG   2520 n = pc:gosub620:pc$ = r$:pc = pc + il:n = fo
          :gosub620:fo$ = r$:n = so:gosub620:so$ = r$
PE   2530 ifil<3thenso$ = " [2 spaces] " :ifil<2
          thenfo$ = " [2 spaces] "
NA   2540 print#1,lnspc(5-len(str$(ln)))pc$ "   " mv$ "   "
          right$(fo$,2) "   " ;
IO   2550 print#1,right$(so$,2) "   " lb$spc(7-len(lb$))oc$;
IK   2560 print#1,spc(5-len(oc$))op$;:ifl(3) = 0then2580
JA   2570 print#1,spc(14-len(op$))right$(a$(t),len(a$(t))-l(3));
KB   2580 print#1:ln = ln + 1:nextt:return
ID   2590 println;:tb = 6:x = 0:lt = 6:gosub2950
          :ifi$ = " exit " then650
IC   2600 ifi$ = " fix " thenln = ln-1
          :printchr$(-13*(asc(g$)<>13));:goto2590
```

```
BF   2610 ifi$ = " " andg$ = " ; " thena$(ln) = " ; "
          :print " [1 crsr left ] " g$;:tb = 8:x = 0:lt = 71:goto2710
IJ   2620 a$(ln) = i$:ifg$ = chr$(13)then2740
PO   2630 tb = 13:x = 0:lt = 4:gosub2950:a$(ln) = a$(ln)
           + chr$(160) + i$:ifg$ = chr$(13)then2740
KH   2640 tb = 18:x = 0:lt = 13:gosub2950:ifi$ = " "
          andg$ = chr$(13)then2740
EH   2650 a$(ln) = a$(ln) + chr$(160) + i$:ifg$ = chr$(13)
          then2740
LA   2660 ifg$ = " ; " thenprint " [1 crsr left ]   " tab(32) " ;   " ;
          :a$(ln) = a$(ln) + chr$(160) + g$:goto2700
PB   2670 tb = 32:x = 0:lt = 1:gosub2950:ifi$ = " "
          andg$ = chr$(13)then2740
ME   2680 ifi$ = " " andg$ = " ; " thenprint " [1 crsr left ];   " ;
          :a$(ln) = a$(ln) + chr$(160) + g$:lt = 32:goto2710
JJ   2690 goto2730
GH   2700 tb = 34:x = 1:lt = 32
KJ   2710 gosub2950:ifi$ = " " andg$ = chr$(13)then2740
LE   2720 a$(ln) = a$(ln) + chr$(160) + i$:ifg$<>chr$(13)
          andx<>ltthen2710
GH   2730 ifg$<>chr$(13)thenprint
EN   2740 ln = ln + 1:ifln>memthen650
DN   2750 goto2590
GK   2760 println;:tb = 4:x = 0:lt = 16:gosub2950
          :ifi$ = " exit " then650
MM   2770 ifi$ = " fix " thenln = ln-1
          :printchr$(-13*(asc(g$)<>13));:goto2760
CD   2780 ifi$ = " " andg$ = " ; " thenl$(ln) = " ; "
          :print " [1 crsr left ] " g$;:tb = 6:x = 0:lt = 73:goto2900
KF   2790 l$(ln) = i$:ifg$ = chr$(13)then2930
PK   2800 ifg$ = " ; " thenprinttab(21)g$;:tb = 23:x = 0
          :lt = 56:goto2890
FM   2810 ifx = ltthen2860
IA   2820 gosub2950:ifi$ = " " andg$ = chr$(13)then2930
CM   2830 l$(ln) = l$(ln) + "    " + i$:ifg$ = chr$(13)then2930
HN   2840 ifg$ = " ; " thenprinttab(21)g$;:tb = 23:x = 0
          :lt = 56:goto2890
FO   2850 ifx<>ltthen2820
BO   2860 tb = 21:x = 0:lt = 1:gosub2950:ifi$ = " "
          andg$ = chr$(13)then2930
EI   2870 ifi$ = " " andg$ = " ; " thenprint " [1 crsr left] " g$ "    " ;
          :lt = 57:goto2890
OF   2880 goto2920
KP   2890 l$(ln) = l$(ln) + "   ; "
IF   2900 gosub2950:ifi$ = " " andg$ = chr$(13)then2930
KK   2910 l$(ln) = l$(ln) + "    " + i$:ifg$<>chr$(13)
          andx<>ltthen2900
ED   2920 ifg$<>chr$(13)thenprint
IM   2930 ln = ln + 1:ifln>lkthen650
GJ   2940 goto2760
EL   2950 i$ = " " :printtab(tb);
OI   2960 print " r  R[1 crsr left ] " ;
HP   2970 getg$:ifg$ = " " then2970
JJ   2980 ifg$> " z " org$< "   "
          andg$<>chr$(13)andg$<>chr$(20)then2970
FI   2990 x = x + 1:ifg$ = chr$(13)org$ = chr$(20)then3040
BO   3000 ifg$ = "    " org$ = " ; " thenprint "    " ;:return
MJ   3010 ifg$ = " , " theng$ = chr$(108)
BP   3020 printg$;:i$ = i$ + g$:ifx = ltthenreturn
IP   3030 goto2960
HM   3040 ifg$<>chr$(20)thenprint "    " :return
BD   3050 iflen(i$)<2then3070
BF   3060 print "   [2 crsr left ] " ;:x = x-2
          :i$ = left$(i$,len(i$)-1):goto2960
IO   3070 iflen(i$) = 0thenx = x-1:goto2960
DP   3080 print "   [2 crsr left ] " ;:x = x-2:i$ = " " :goto2960
```


THE SLIGHTLY AMAZING
CAPTAIN SYNTAX
OKAY. FOR THOSE JUST TUNING IN... CAP'N SYNTAX IS TRYING TO RECOVER THE STOLEN "FLOTSKY DISK". BRIEFLY IN HIS GRASP, IT WAS TAKEN FROM HIM BY A MYSTERIOUS ASSAILANT. OUR HERO THEN PROCEEDED TO DR. FLOTSKY'S HOUSE (THE CREATOR OF THE DISK) BELIEVING HIM TO BE IN DANGER. HE ARRIVED TOO LATE: FLOTSKY WAS GONE! AS HE LEFT THE HOUSE, HE WAS CLOBBERED (OUCH!). SO NOW HE AWAKENS IN A DIM LAB AND YOU CAN FIND OUT WHY EVERYONE WANTS THE DISK!
W-WHERE AM I?
AH! THE HERO RETURNS. YOU ARE IN MY LABORATORY.
OH, AND YOU'RE ABOUT TO BE FED INTO MY COMPUTER.
YOU ARE HERE, COURTESY OF SCAD, MY FAITHFUL EMPLOYEE
YEAH. I HAVE LUMPS TO REMEMBER HIM BY. SO YOU, I ASSUME, PILFERRED THE DISK?
CORRECT. I ALSO TOOK IT FROM YOU IN THE ALLEY.
YOU SEE, MY GOAL IS TO IMPLANT THE CHIP THAT MY DISK TELLS HOW TO MAKE IN THE EARS OF KEY CITIZENS. POLITICIANS, FOR EXAMPLE. I PLAN TO DO THIS BY HAVING CERTAIN DOCTORS DO IT DURING 'NORMAL' CHECK UPS. THE DOCTORS WILL BE WELL PAID, NATURALLY.
GEE, WHAT A NICE GUY! BUT IF I'M TO BE LUNCH FOR AN IBM, HOWS ABOUT LETTING ME SEE YOUR FACE. JUST AS A FAVOR TO ME.
OF COURSE.
DR FLOTSKY?
HEE HEE!
DON'T YOU LOVE SURPRISES?
BUT WHY? CARE TO ENLIGHTEN ME?
OF COURSE.
ONCE IMPLANTED, THE CHIP ENABLES TOTAL MIND CONTROL BY ME. CONTROLLING THOSE IN CONTROL WILL GRANT ME POWER. EVENTUALLY IT WILL BE A PUPPET GOVERNMENT CONTROLLED BY ME. MY JOB AS A SCIENTIST ILL-SUITED MY TRUE POTENTIAL!
AND NOW.... TA-TA!
HEEYYYY!!!
ZOOP!
NOW... ON TO VICTORY!!
NEED IT BE SAID?
TO BE CONTINUED!

At this point in time John really appreciates the fact he bought the "Big Boss" joystick with the heavy duty extension cord



"The fool. . . Obviously his problem is on line 1040. . . He's misplaced a Y coordinate with a control function."

News BRK

Late Note

Don't forget the Computer Fair (May 9th through the 12th) at Toronto's brand new Metro Convention Centre at the foot of the CN Tower. This is the first major computer show for the downtown area and it should prove very entertaining. The Transactor will be at booth 1720.

$4 Million "Computers and Children" Program Launch

"Computers and Children", a new $4 million Ontario government program ensuring that the children of this province will have equal and adequate access to computers and related information technologies, was launched today by the Honourable Susan Fish, Minister of Citizenship and Culture and the Honourable Larry Grossman, Treasurer.

The first 15 of an eventual 230 community computer centres were opened by Ms. Fish and Mr. Grossman at Toronto's Central Neighbourhood House. Connected with Toronto by an audio teleconference, the centres participating in today's ceremonies were: London, West Bay (Manitoulin Island), Cornwall Island, Sudbury, Chatham, Peterborough, and two in Chapleau. Six other centres in Metropolitan Toronto were opened immediately following today's launch.

Approximately 4,000 microcomputers will be placed in communities across the province, under the new program funded by BILD (Board of Industrial Leadership and Development) and administered through the Ministry. Community organizations such as libraries, museums, art galleries, and native, multicultural and service groups will house the computer centres.

"The primary objective is to provide computers and computer learning to children, regardless of their ethnic or socio-economic background, to ensure that no child is deprived of future employment prospects because of a lack of computer knowledge," Mr. Grossman said.

Up to 15 microcomputers, colour monitors and software packages will be placed in each centre using hardware from Apple, IBM, Commodore, Acorn and Atari. At least two hours of free time to students from kindergarten to grade eight, outside their regular school hours, will be provided with remaining time being available to the entire community at a minimal user fee.

"The key to success of Computers and Children is partnership," Ms. Fish said. Ministry-trained co-ordinators will work with volunteer assistants to ensure that the program meets the unique needs of each community, and private sector fund-raising will occur to finance each centre's ongoing operations. The result of these partnerships will be entirely self-sufficient centres integrally involved in the development of their communities.

"Computers and Children" will certainly help young people prepare for future employment, but it will also provide incentives for Canadian software and hardware development improved language instruction for newcomers; the opportunity for parents to learn this new technology with their children; and for seniors to volunteer to help familiarize young people, as well as themselves, with the world of microcomputer," added Ms. Fish.

This program will complement initiatives existing in our schools, The new facilities will be located primarily to reach less advantaged young people who would not otherwise have access to this equipment outside the academic setting. In addition, adults and small businesses may have access to the machines.

In December of this year, another eight centres are planned to open: Red Rock, Campbelville, Etobicoke, Burlington, Deseronto, Windsor, and two in Ottawa, Another 11 are scheduled to open early in the new year: Trenton, North Bay, Hamilton, Foleyet, Sudbury, Brantford, Winchester, St. Catharines, Fort Frances, Schumacher (Timmins), and Toronto.

Stanford University (Children's Learning Lab) in California, Edinburgh University (Children's Education Unit) in Scotland, and the Massachusetts Institute of Technology) Learning Lab for Children) (MIT) have all expressed interest and enthusiasm for Ontario's plan to open up computers to the community on such a large scale. Initiated by the Treasurer in his 1984 Budget, it is the first time a project like this has been undertaken.

Ministry of Citizenship and Culture
Program contact: Dorothy Netherwood
Computers and Children
454 University Avenue
Toronto, Ont. M5G 1R6   416 963-3304

Events

The Fourth Annual TPUG Conference.

TPUG, The Toronto PET User's Group (and also the world's largest Commodore users group) is holding another "year end" conference that will no doubt be every bit as entertaining as their last three.

Dates: Sat. May 25 & Sun. May 26, 1985
9:30 a.m. to 5:00 p.m. (both days)

Location: 252 Bloor St. W.
(at St. George subway),
Second Floor, Toronto, Ontario

Activities:
Full two-day program of speakers covering topics for beginners and experts.
Club library (5000+ programs) at special conference price of $4.00 per disk.
Exhibitors of hardware, software, accessories, "Answer Room" – free 10 minute consultation.
Trader's corner for used equipment.

Cost: Pre-registration (before April 15 and including one year's Associate membership) $45.00. For more information, contact:

Doris Bradley
TPUG Business Office
1212A Avenue Road
Toronto, Ont. M5M 4A1
416 782 9252

The Second Annual Commodore User Computer Fair

MARCA, The Mid-Atlantic Commodore Association, is planning their second exhibition to be held at the new Sheraton Convention Centre in Valley Forge, PA. The new centre, which is still under construction, will open this spring as the "largest privately owned convention centre in the U.S.A." Over 30,000 square feet of exhibit space has been reserved for the show.

Dates: July 26, 27, and 28, 1985.

A complete line-up of speakers and seminars is already organized with the schedule so far standing at about 12 different lectures every hour. For more Information:

Joel A Casar
Exhibits Chairman
2015 Garrick Drive
Pittsburgh, PA 15235
412 371 2882

Survey Shows Commodore in Top Ranks of Electronics Industry

Commodore International Ltd. continues to maintain its stronghold in the microcomputing and consumer electronics market, according to a report published recently by Electronic Business Magazine, Boston.

Of the top 200 companies in the field of electronics over the past five years, Commodore International was ranked in first or second position in all four major financial catagories researched.

Commodore ranks at the top in 5-year net income growth rate (91.8% per year average) and in return on investment (30.7%) and second in 5-year revenue growth rate (68.5% per year average) and in return in equity (46.1%).

For further information, contact Allan Reynolds at 416 922-5556

5-YEAR REVENUE GROWTH RATE

	Percent per year	Rank
Top 10		
Tandem	76.7%	84
Commodore International	68.5%	73
ROLM	58.5%	78
Cray	58.1%	149
CPT	54.7%	135
Paradyne	51.8%	129
Wang Laboratories	50.7%	29
Computervision	47.6%	89
SCI Systems	47.4%	102
Docutel/Olivetti	45.6%	132
Bottom 5		
ITT	−1.5%	4
Cincinnati Milacron	−2.5%	171
LTV	−2.7%	189
AMF	−5.9%	144
Gould	−6.7%	39

5-YEAR NET INCOME GROWTH RATE

	Percent per year	Rank
Top 10		
Commodore International	91.8%	73
Tandem	70.3%	84
Computervision	63.9%	89
Cray Research	58.6%	149
ROLM	58.0%	78
Wang Laboratories	57.7%	29
CPT	54.7%	135
Computer Consoles	53.1%	199
Watkins Johnson	52.2%	139
SCI Systems	47.8%	102
Bottom 5		
Bally	−30.4%	120
AT&T Technologies	−32.4%	2
Scientific-Atlanta	−40.7%	94
AMF	−40.9%	144
Management Assistance	−69.3%	92

RETURN ON INVESTMENT

	Percent	Rank
Top 10		
Commodore	30.7%	73
Dysan	23.4%	130
Aydin	23.3%	159
Micom Systems	22.7%	190
SmithKline Beckman	22.1%	72
E-Systems	21.9%	65
Tandy	21.8%	22
IBM	20.6%	1
EG&G	20.2%	118
Diebold	19.6%	115
Bottom 5		
Centronics	−8.9%	152
Docutel/Olivetti	−10.8%	132
Texas Instruments	−10.2%	10
Warner Communications	−26.5%	45
Kratos	−32.3%	182

RETURN ON EQUITY

	Percent	Rank
Top 10		
Transitron Electronics	64.1%	186
Commodore International	46.1%	73
Lockheed	31.0%	53
Dysan	27.2%	130
Aydin	25.4%	159
Ford Motor	25.1%	30
Tandy	24.8%	22
SmithKline Beckman	23.6%	72
IBM	23.6%	1
Technicom International	23.5%	169
Bottom 5		
Centronics	−12.7%	152
Docutel/Olivetti	−20.1%	132
Savin	−18.2%	117
LTV	−20.5%	189
Warner Communications	−43.5%	45

Transactor News

Transactor Disk Offer Update

As of this issue there are 6 Transactor Disks:

Disk 1: All programs from Volume 4
Disk 2: Volume 5, Issues 01–03
Disk 3: Volume 5, Issue 04 (Business & Ed.)
Disk 4: Vol. 5, Issue 05 (Hardware/Periphs.)
Disk 5: Vol. 5, Issue 06 (Aids & Utilities)
Disk 6: Vol. 6, Issue 01 (More Aids & Utilities)

Transactor disks are now available on a subscription basis through the order form in the centrespread of the magazine. Disks can be purchased individually for $7.95 (Cdn.) each; the extra two dollars that was to be charged for the first disk has been dropped.

Perhaps a word of explanation is in order here. The original idea was to charge $9.95 for the first disk you purchased, and we'd make up a mailer for that disk (and each subsequent one) which contained your name, address, paid postage, and a two dollar off coupon for future disks. As it turned out, post office regulations nixed the mailer idea, so we decided to just send disks out on an individual basis for $7.95, and offer disk subscriptions. The subscription is mainly as a convenience, since our pricing philosophy dictates a rock–bottom price for single disks, and the discount for a subscription rather than individual purchase isn't that great.

To anyone who already sent in $9.95 for their first disk we'll credit the two dollars toward future disks or subscriptions.

The Complete Commodore Inner Space Anthology

In the last issue we announced that orders were being accepted for The Complete Commodore Inner Space Anthology. Then we referred to the back cover ad which proceeded to say, "Watch For It, January 1985". Due to a slight mix-up, this ad was not the one we intended for the back cover spot. In fact, as most of you probably noticed, it was the exact same ad as the previous issue published in December 1984. The back cover ad of this issue is the one that should have appeared and we apologize for any confusion or inconvenience this may have caused.

Also, the price announced last issue has been changed to $14.95. An oversight in costing is the reason for the slight increase but we fully intend to honour any orders we've already received at the former price.

Once again, we are now taking orders for the long awaited second edition of the Special Reference Issue. As you can see, the title is somewhat different than its predecessor, but then so is the inside. Of course most of the material from the first edition is included, with twice as much again added.

The price? Just $14.95! Originally the price was projected at around $25 dollars. Two reasons account for the difference. First, the disk we intended as part of the package will now be made available separately (details next issue). Secondly, we have decided to publish the book on our own. Previously we had considered releasing the book to an outside publisher but by doing it ourselves the price can brought down substantially.

The Complete Commodore Inner Space Anthology is currently available by mail order only through The Transactor. (Eventually they will be appearing in the major book chains, but at a slightly higher price) The easiest way to order is with the postage paid reply card at the center of the magazine. Mark the card appropriately and don't forget your postal/zip code. If you're paying by charge card, please include the expiry date. If you're sending a cheque, you can tape the postage paid card to the outside of an envelope. Please allow 4 weeks for delivery.

Notice to "Transactors"

Planning to write an article to submit for publication? Great! However, we get a lot of articles from authors around the world and although 90% of them are superb, we only regret that we can't publish them all.

One of our main criteria for choosing an article is the amount of time it will take us to prepare the material. If a diskette is included with your article in some kind of text file, the preparation time is reduced by easily 95%. Most authors are using a wordprocessor to write their material and most articles are submitted along with a disk, but those that aren't get pushed to the bottom of the pile. With the price of a disk hovering at about $4 each, it shouldn't be too hard to justify the expense, considering we pay $40 per printed page. And the investment just might secure your article a spot in the magazine.

Summed up, if you are sending us an article you have written on a wordprocessor, please include a disk with the text files. Any file types are acceptable since even if we don't have the particular software you have, it usually isn't too hard to convert them for our use. Drive type isn't a problem for us either - we have them all. Naturally you should also include any programs that accompany your text.

Software

New Software From Commodore

Commodore Business Machines Limited is please to announce 14 new software titles for the Commodore 64. These educational programs are designed for children from preschool to junior high.

CHILDSPLAY is a series of educational games specifically designed for the preschooler. The first package, Letters, Shapes & Numbers introduces a child to the alphabet, different shapes and the number system. Interaction between the child, the screen and the keyboard reinforces the fundamental concepts of learning. The second package in the Childsplay series is Match'em, Copy Cat. Match'em is a game where the child is challenged to recognize the difference between two objects on the screen. The computer will keep track of the score and display it after the child has completed a round. Copy Cat is a computerized version of 'follow the leader'. This program will display a series of shapes on the screen which the child must duplicate - a fun way to develop memory recall skills. Childsplay would also benefit any child with learning disabilities. Childsplay is designed to give preschoolers the head start they deserve.

EDUKAT Junior Math Series is comprised of 8 different math topics designed for children 8 to 1. The topics covered are:

- Addition
- Subtraction
- Division
- Multiplication
- Geometry
- Graphing
- Decimals part 1
- Decimals part 2

The EDUKAT Series was created by professional educators to allow students to be tutored at the touch of a button. Each program gives the student step by step instructions on how to solve problems in each particular topic. The program will then give the student a lesson on what they have just learned. If the student runs into any difficulty help is available in the form of hints. After the student has completed a lesson, a report card can be generated which will outline the students strong areas and the areas where they encountered problems.

COMPUQUIZ is a trivia game for kids of all ages. The series consists of 4 different packages. These are:

- Sports
- Science
- History
- General Interest

Each category comes on its own separate disk with 1,000 different questions. In COMPUQUIZ you match wits against your opponent. A question is flashed on the screen with 4 possible answers. Use either a joystick or the keyboard to be the first to answer correctly and the prize is yours. Answer 9 questions and you win the game. For more information, contact:

Customer Service
Commodore Business Machines, Ltd.
7261 Victoria Park Avenue
Markham, Ont. L3R 2M7

Magnetic Templates

Database and spreadsheet programs have a great potential for becoming the workhorses of home and small business computing. With proper application templates, there is no end to the type of projects they can tackle. They are also written by people who would be unwilling or unable to write the same program "from scratch" in Basic or any other language. In addition, as anyone who has a shelf–load of specific programs will tell you, not having to learn a new set of keystrokes and set-up procedures for each application greatly simplifies its use. Once you learn the ins and outs of the database or spreadsheet program, you already know how to run, modify and print the new application. Running several different types of applications will go much smoother if you can focus on the information you are working with, and not how to run The Program.

We believe there is a lot of potential for both general and specific application templates for these master programs. Also, the tie–in to the "shareware" concept and the automatic royalties to authors makes the project just too tantalizing NOT to try!

This is initially an experiment, the success of which hinges on a rather optimistic perception of human nature. First, there must be enough spreadsheet and database users out there who can develop high quality applications, who can explain their use and modification clearly to others, and who are willing to put the product of their labour out in the marketplace. Second, there must be a sufficiently large group of users who will be willing to make a non–mandatory payment to the authors and to the distribution organization that made the whole thing possible.

If this project is successful, it will be one of those business situations in which everyone WINS. We will be rewarded for our concept and organization, authors will be rewarded with royalty checks, and users will be rewarded by only paying for the software they actually USE.

Interested? If so, do two things:

1. Send a U.S. stamped, self–addressed #10 envelope. In a few months you will either receive a catalog of available templates, or a full progress report on the evolution of the project.

2. Make a list consisting of two categories: Templates you would like to see, and Templates that you could develop (note the program they would run on).

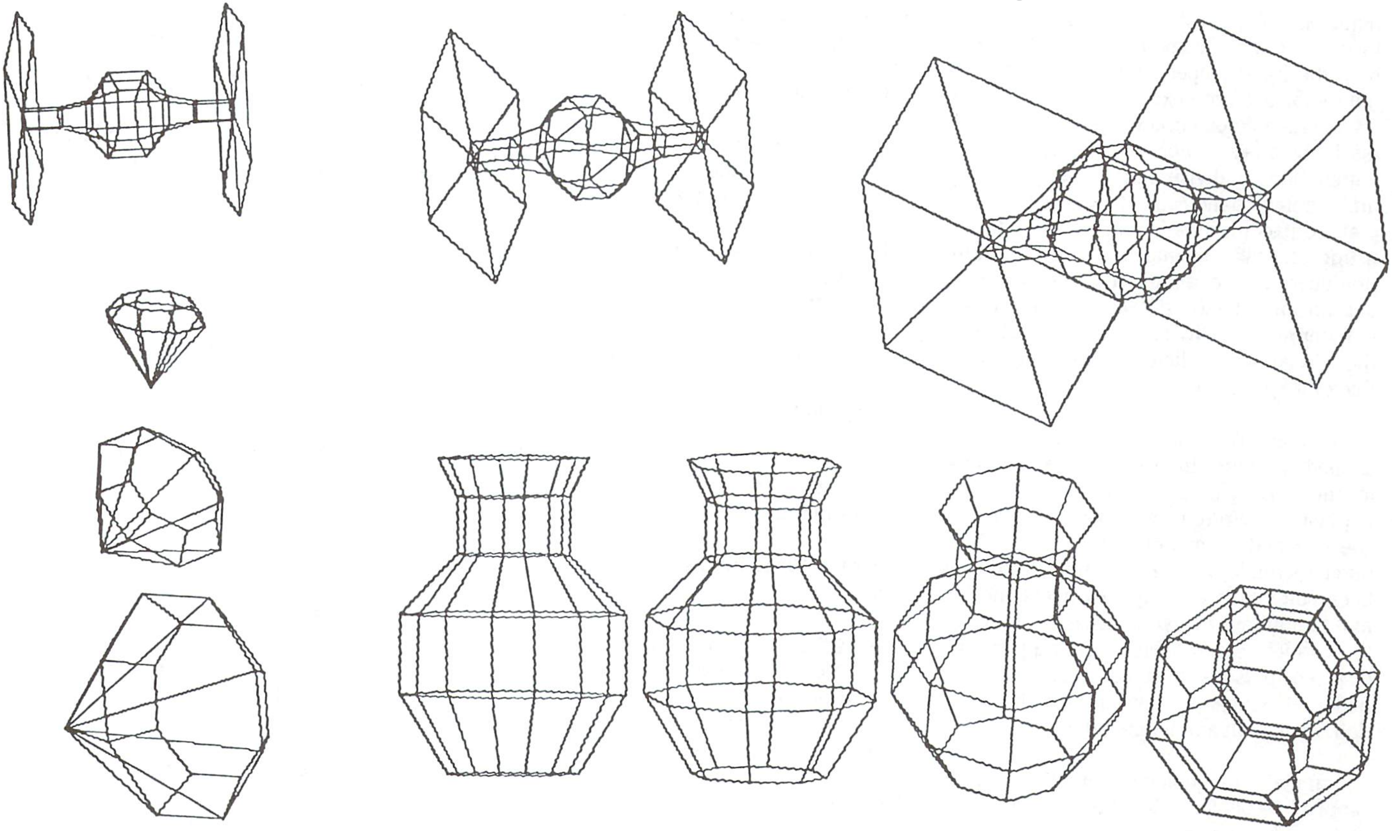
If you express interest in developing templates that we feel are appropriate for this first catalog, we will send you a copy of "guidelines for submission" with information on format and documentation.

For now the project is limited to the Commodore 64 computer with disk drive. This is the machine that Steward Brand calls the "BIC lighter" of computers. I call them both TOOLS. They are both igniters of potential energy. What we have after is the warmth of the fire not the prestige of the tool. Both the BIC and the C–64 are capable of lighting some pretty big fires. For further details, contact:

Cooperative Design
P.O. Box 138 / 500 Aurora Ave. N. #201
Langley, WA 98260 / Seattle, WA 98109
206 221 2373 / 206 682 8663

Graph–Tech Software Co. Releases 3–D World 64

Graph–Tech Software Co. of new York has just released 3–D WORLD 64, a new wireframe graphics package for the C–64, which



allows the creation of complex 3-dimensional objects, which can then be viewed on screen and/or plotted as line-art to the Commodore 1520 Plotter/printer. Objects are defined by entering points in a 3-dimensional space utilizing the cartesian coordinate system, and then line segments, also entered by the user, are drawn connecting those points.

Objects created using 3-D WORLD can be manipulated on screen as well as on paper. This program allows the ROTATION of any image in 3-dimensions on all three axes, a well as SCALING and TRANSFORMATION. 3-D WORLD 64 is totally menu-driven, and requires no additional hardware or software to generate the high-resolution screens or to handle the plotter routines. The program is 100% compiled code, and is set up in integrated program modules; in addition, "object definition files" can be created outside of 3-D WORLD 64 (ie. in BASIC), and as long as they are formatted to disk in the same structure as explained in the accompanying 50 page manual, they can be used and manipulated within 3-D WORLD 64.

Sample objects such as the ones pictured here are included on the program diskette. Suggested retail price is $39.95 U.S. Address all inquiries to:

Graph-Tech Software Co.
1315 Third Avenue, No. 4C
New York, NY 10021

Enhanced Version of Flexidraw Available From Inkwell Systems

San Diego-based Inkwell Systems releases Flexidraw 4.0, a major update of its successful light-pen graphics package for the C-64. Enhancements to the program include increased drawing capabilities, an expandable work area, custom fonts, more than 500 pattern fills and an oops feature for "fill-spills." The update allows picture editing in the colour program and has improved telecommunications capability. A wider variety of printers and interfaces are now compatible with the update. This Flexidraw program was exhibited at World of Commodore II.

The new program includes arcs and ellipses as well as three line width choices increasing drawing versatility. Moreover, the addition of two-dimensional 90° rotation and flip capability expand figure manipulation potential.

Using a split-screen feature, larger drawings are now possible by creating an expandable work area. These drawings can then be printed two screens at a time by linking them horizontally or vertically.

Graphic design capability includes 6 custom fonts, each available in regular or expanded size, and one large letter font in an ornate, Old English style. The 7 choices compliment the standard Commodore character set.

With over 500 pattern fills, Flexidraw provides the user with a larger choice of patterns to add texture and dimension to projects.

The colour program includes a zoom function for one-pixel editing, an improvement which allows original designs to be altered and then saved to disk. These coloured pictures can now be printed on several of the new color printers currently available.

Improved telecommunications capability facilitates quicker transmission of design work for adjustments and approval.

Inkwell Systems
P.O. Box 85152, MB290
San Diego, CA 92138

Books

Closing The Gap announces publication of "Computer Technology For The Handicapped"

Henderson, Minnesota — Closing The Gap (CTG), internationally recognized resource authority on microcomputer applications for special needs populations, announces the publication of "COMPUTER TECHNOLOGY FOR THE HANDICAPPED," the selected proceedings from the 1984 CTG Conference held September 13-16 in Minneapolis, MN.

"Computer Technology For The Handicapped" is a treasury of state-of-the-art microcomputer applications written for special education and rehabilitation professionals as well as handicapped individuals, their families and associates. This 260 page book details 45 of the more than 80 presentations made at the CTG Annual Conference and focuses on how computer technology can help the handicapped or disabled person – not tomorrow, not next year, but today. Topics include microcomputer applications in all disability areas – hearing and speech impaired, blind and vision impaired, physically and mentally handicapped – as well as complete contact information on all presenters, a listing of nearly 60 hardware and software producers who exhibited at the conference, and an introduction by Dolores Hagen, co-founder of CLOSING THE GAP and author of the acclaimed "Microcomputer Resource Book for Special Education."

"Computer Technology For The Handicapped" provides information that is crucial for the delivery of technology to the disabled population. Written by experts in their respective fields from around the world, "Computer Technology For The Handicapped" has been edited into an easy-to-understand format that allows everyone access to information which can enable handicapped and disabled persons to meet their everyday needs of education, communication, vocation, recreation and independent living.

"Computer Technology For The Handicapped" retails for $17.95. For further information, please contact:

Budd Hagen
Closing The Gap
P.O. Box 68
Henderson, MN 56044
612 248 3294

Directory of Online Industry Professionals – The Who, What, When, Where, and How

A unique resource for online industry professionals and subsidiary markets is available in the just-released Marquis Who's Who Directory Of Online Professionals, containing more than 6,000 professional profiles representing every part of the field, from users to vendors of online products and services.

"We believe the industry has reached a level of growth," states editor-in-chief Nancy Gorham "that makes such a reference book vital". Advancing, by conservative estimates, at an annual growth rate of more than 23%, the online industry is eyeing a gross value by year-end of $2 billion. The Directory is designed to be the primary communication tool for the entire industry.

"We have culled essential facts from the professionals included, for example, online expertise, current monthly online usage, equipment used, database subject expertise, databases and systems used, professional affiliation and function, articles and books, and of course, other career and training information, plus address and phone number(s) when available. I believe there is no better resource for the exchange of ideas and communication in the industry. We have the who, what, when, where and how of the industry wrapped up in this volume."

Multiple indexes enable users of the Directory to locate individuals by subject expertise, online function, or geographic area.

"The information presented," notes Gorham, "has given us a rare picture of the nature of the industry and the people in it."

The Directory pulls together for the first time the growing online field. While the industry is a developing one, it is already internationally important. Forty-seven countries are represented in the Directory; over 20% of the professionals listed are non-U.S., including 11% from European countries."

Regarding functional skills represented in the Directory:

- 54% are intermediary searchers
- 23% are information managers
- 10% producers of database products & services
- 8% are end users.

The database subject expertise of these online professionals reveals that:

- 30% cover business areas
- 25% have expertise in medicine
- 12% have expertise in current affairs
- 9% have expertise in energy.

The Directory of Online Professionals is a hardbound volume containing 829 pages. The cost is $85.00.

A computerized version of the Directory, known as the Marquis PRO-Files Database, will be available on DIALOG in 1985 through the Marquis Data Products Department. For further information, please contact:

Morris Wattenberg
Marquis Who's Who, Inc.
200 East Ohio Street
Chicago, Illinois 60611
312 787 2008

Hardware

"True Piano Feel" Keyboard Released by Commodore

Commodore Business Machines, Ltd. has introduced the Music Mate keyboard for the C64 to provide realistic instrument reproduction.

Covering 2 1/2 octaves, the keyboard features the ability to play three notes simultaneously; record and playback capabilities; adaptability to joystick port; and eight instruments preset – plus the option to define your own.

The Music Mate Keyboard is featured at an attractive suggested retail price of $149.95 and is packaged with the initial disk-based software which normally retails for $39.95.

Additional program disks are available for the keyboard, including: Song Builder, featuring additional speed control, recording and overdubbing qualities; Song Editor, which will display and edit songs written with the Song Builder; Song Printer; and Sound Maker – a full colour graphic display panel allowing the user to personally program the shape, volume and tone of desired sounds.

Richard G. McIntyre
Vice President, Sales
Commodore Business Machines, Ltd.
3370 Pharmacy Avenue
Agincourt, Ont. M1W 2K4
416 499 4292

Kobetek Announces Valiant Turtle

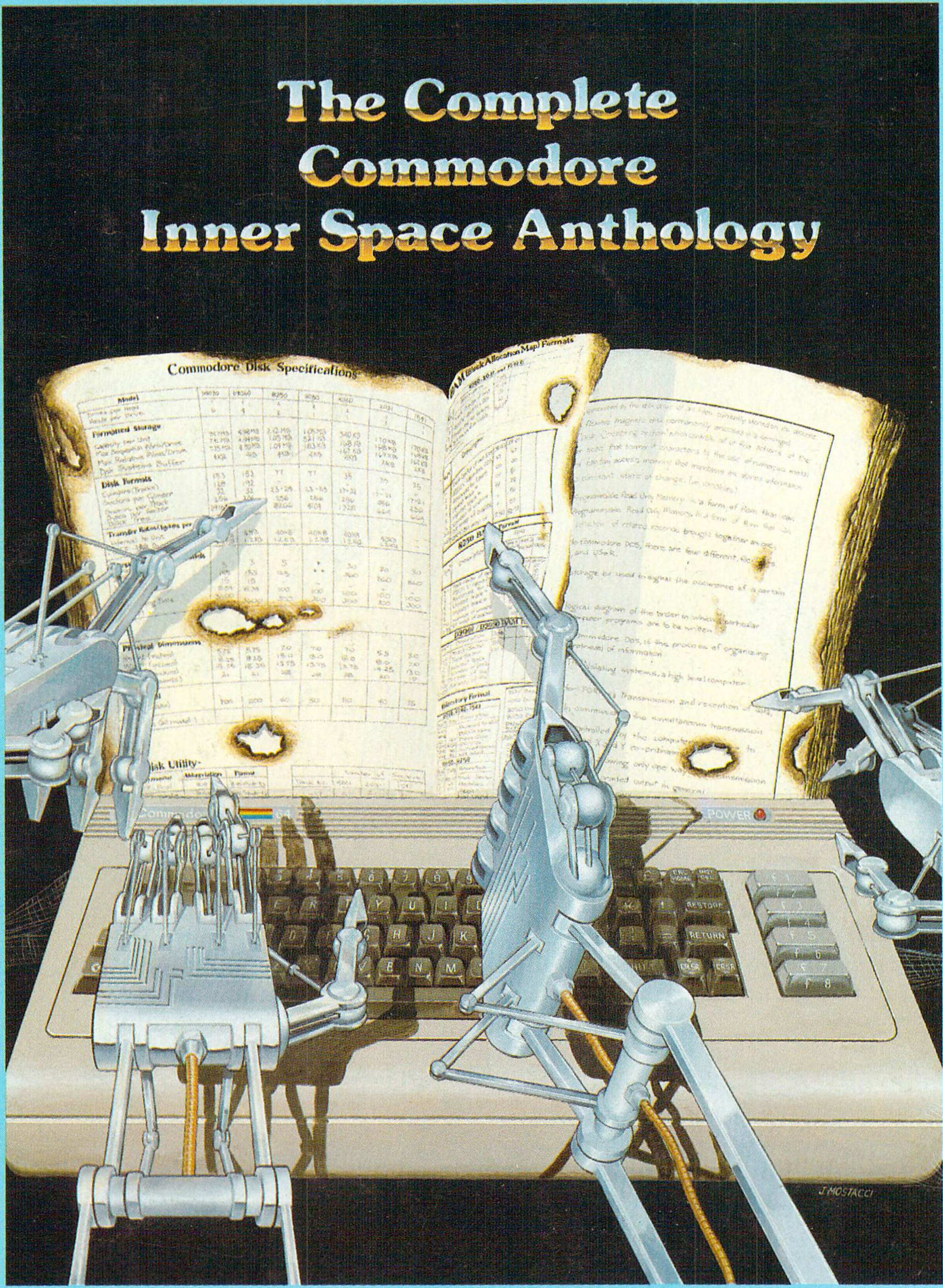
Kobetek is pleased to announce the availability of the VALIANT TURTLE. Unlike earlier turtles, this incredibly versatile robot is remote controlled by an infra-red transmitter and the software allows it to execute all LOGO commands. Children love its looks and teachers appreciate its impressive array of capabilities: the turtle's two independent stepper motors make it the most accurate on the market; it is powered by ten nickel-cadmium rechargeable batteries – simply plug the power adaptor (included) into a socket on the turtle; two illuminated eyes serve as power indicators – they go out before any other functions fail; it carries a pen which can be raised or lowered to trace its movements. The turtle moves in units of 1 cm but can be programmed to move in units of 1 mm, 1 inch or 1 meter and it can draw smooth circles and arcs. The Valiant Turtle comes as a complete package: turtle, batteries infra-red transmitter, power adaptor, pen, manuals and software.

The Valiant Turtle interfaces with most popular microcomputers including: C-64, Apple, BBC, DEC Rainbow, IBM and Spectrum. At present, only the C-64 and BBC versions are available, but the others will follow in the first quarter of 1985. A French language version of the software is available. LOGO workbooks can be obtained at an additional cost. The Valiant Turtle is manufactured in England by Valiant Designs Limited. For further information, please contact:

Kobetek Systems Limited
1113 Commercial Street
New Minas, N.S. B4N 3E6
902 678 9800

J. Mostacci
T. JOHN M.S
The Transactor
actor
actor
actor
tor
actor

The Transactor presents, The Complete Commodore Inner Space Anthology



Only $14.95

Postage Paid Order Form at Center Page
See News BRK (page 77) For Details