Simulations and Modelling

- Random Numbers in Machine Language
- N-Body Simulator for the 64
- RLE Picture Files Revealed
- Beyond Bulletin Board Systems
- Commodore in Europe
- 1350 Mouse Modifications

- Structure Browser for the Amiga
- Guru Meditation Numbers Explained
- More on Amiga File Structure
- Recursion Techniques Described
- Textscan for CP/M
- 1571 RAM Disk Copier
Life Doesn't Stand Still... Why Should Your Pictures?

Moving Pictures

Chris Zamara
Nick Sullivan

For the Commodore 64

Full Screen Animation On Your Commodore 64!

It used to be even the experts couldn't do it. Now, anyone can.
Moving Pictures by AHA! is more than just another animation package. It's a whole new breakthrough in software technology.
Moving Pictures is fast, smooth, full-screen animation that is totally under your control.
You use your favourite graphics tool* to draw the frames of your movie, then show it at full animation speed with a single command.
Write movie "scripts" in BASIC, using the powerful Moving Pictures command set for complete control of your creations!
Whether you're a programmer or a novice, you'll be able to put together and display intricate scenes of your own invention. You can even edit your scripts or execute a BASIC program while a movie is being displayed - Moving Pictures is a multitasking system!

Besides being fun in itself, Moving Pictures lets you easily add animation sequences to your own BASIC programs.
Just a few of the many Moving Pictures features:
• allows split screen operation - part graphics, part text - even while a movie is running
• repeat, stop at any frame, change position, and colors, vary display speed and more
• hold several movies in memory and switch instantly from one movie to the other
• instant, on-line help available at the touch of a key
• no copy protection used on the disk
• and here's the best part: the price is just $29.95!

* Graphics program not included. Moving Pictures uses a standard hi-res bitmap, so many graphics programs are fully compatible, including: Flexidraw™, Doodle™, Gold Disk Art Package™, Print Shop Screen Magic™, Perspectives™.

Mail Orders: Transactor Publishing Inc., 500 Steeles Avenue, Milton, ON, Canada, L9T 3P7 (416) 878-8438
(or use order card at center).

Canadian and International Dealer Inquiries To: Norland Software Products, 251 Nipissing Road, Unit 3, Milton, ON, Canada, L9T 4Z5. (416) 876-4774.

USA Dealer Inquiries To: American Software Distributors Inc., Box 290, Urbana IL, USA 61801 1-800-225-7941.
Simulations and Modelling

Start Address Editorial .................................................. 3

Bits and Pieces ..................................................... 6
Using the WAIT Function For Screen Changes
* Last File used * update
Easy C-128 Un-New
Printer output from an ML monitor
Quick Directory Hider
Editing in the C-Power Shell
* High-Res 128 * Video Fix
Little-Known Features of DOS
The Amazing 1660 Modern
Bit Correction: DOS Wedge
Correction: Format Track 36
Assembler Start-up Code
C128 RS-232 Bugs
Using The C-128 System Vector
Moving The Cassette Buffer
Reset and Run
Border Animation!
Verifier and Fast-Load Cartridges
Amiga Bits
The Vanishing Workbench
Beating The Low Memory Blues
Modifying The Epsom Printer Driver
To Work With Other Printers

Letters ......................................................... 12
Angling for weird parts
Helarious plots
Yet another request for Urdu software
C-64 Numerics
Remote line feeds
Kernal revision revision
Eavesdropping on modems
IEEE for C-128
Amiga marketing
No niche for Amiga

News BRK .............................................. 77
Submitting NEWS BRK Press Releases
Transactor Currency Standard Decreed
Subscription Intersection Set
Disk Subscription Notes
Toronto ComputServe Node
Free Transactor T’s
Subscriber Mail Orders
Customs/Duty on Hardware Products
Sold Out!
Transactor Mail Order
Transactor Disks, Back Issues, and Microfiche
New Books from Abacus
Eye-Scan for C-64/128
Spartan now with Apple II disk
Peek A Byte 128

TransBloopers .................................................. 15
EPROM Burns
Hi-res Trace Utility SYS Address
Uncontrolled Sprite Bits
Tape Verifizzle
Slashing and Pounding

Smile! You’re on RLE a short TeleColumn followed by RLE files revealed .................................................. 16
Beyond Bulletin Board Systems a comparison of online services .................................................. 19
Commodore in Europe with a comparison of equipment prices .................................................. 21
Provoking Thought recursion described, and applied in two games .................................................. 24
Machine Language Random Number Generation .................................................. 27
C64 N-Body Simulator solar systems in action! .................................................. 31
A Two Button Mouse modifications and software for the 1350 mouse .................................................. 36
EPROM Programmer Update corrections, modifications, and more .................................................. 48
Help! Help! a transparent instant help utility .................................................. 43
1571 RAM Disk Copier copy double sided disks in one gulp .................................................. 52
Textscan a CP/M source file browser .................................................. 55
That Guru Does Have A Message decipher those meditation numbers .................................................. 59
Structure Browser a handy tool for Amiga explorers .................................................. 71
Amiga File Structure another look at disk data format .................................................. 72
Amiga Dispatches with a special World of Commodore 86 report .................................................. 74
Compu-toons .................................................. 76

Note: Before entering programs, see “Verifizer” on page 4
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 'O' and the letter 'o' will of course be in lower case. Secondly, the lower case L ('l') is a straight line 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 with 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 " [10 spaces] flush right "
```

```
Programs

Contributing Writers

Ian Adam          James E. LaPorte
Jim Barbarello    James A. Lisowski
Anthony Bertram   Richard Evers
Tim Bobbach       Scott McLean
Ranjan Bose       David Martin
Anthony Bryant    Steve McCrystal
Jim Battleday     Stacy McNamara
Betty Clay        Steve Michel
Joseph Caffrey     Terry Montgomery
Gary Cobb          Ralph Morrill
Tom K. Collins     Rick Morris
Robert V. Davis    Michael Mossman
Elizabeth Deal     Gerald Neufeld
Frank D. DeGioia   Noel Nyman
Chris Dunn         Kevin O’Connor
Michael J. Erskine  Richard Pritchard
Jack Farrah        Donald Piven
William Fossett    Terry Pritchard
Jim Frost          Raymond Quirin
Miklos Garamszegy  Gary Royal
Martin Goebel      John W. Ross
R. James de Graff  David Shiloh
Tim Grantham      Fred Simon
Adam Herst         P. A. Slaymaker
John Holtum        Edward Smeda
David Hook         Darren J. Spruyt
Tomas Hruby        Aubrey Stanley
Robert Huenion     David Stidolph
David Jankowski    Richard Stidolph
Bob Jonkman        Anton Treuenfels
Brian Junker       Karel Vander Lugt
Clifton Karnes     Audryl Villas
Lorne Klassen      Jack Weaver
Jesse Knight       Evan Williams
Gregory Knox       Jack Wong

Production

Attic Typesetting Ltd.

Printing

Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7, Canada. Second Class mail registration number 9042. USPS 735-9556, 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. ESS 0857-2330.

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 $5.00. U.S.A. $5.00. All other $2.00. 75.
Air Mail (Overseas only) $4.00. US. (15 post-age issue)

Send all subscriptions to: The Transactor, Subscriptions Dept., 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7. 416-878-8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address above will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

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 8.5 x 11, 31%, 80%, or 8x10 diaries with WordPerfect, WordCraft, Scriptedit, or SGX 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 or illustrations will be included with articles depending on quality. Authors submitting diaries will receive the Transactor Disk for the issue containing their contribution.

Please Note: The Transactor’s phone number is: (416) 878-8438

Production

Attic Typesetting Ltd.

Printing

Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7, Canada. Second Class mail registration number 9042. USPS 735-9556, 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. ESS 0857-2330.

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 $5.00. U.S.A. $5.00. All other $2.00. 75.
Air Mail (Overseas only) $4.00. US. (15 post-age issue)

Send all subscriptions to: The Transactor, Subscriptions Dept., 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7. 416-878-8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address above will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

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 8.5 x 11, 31%, 80%, or 8x10 diaries with WordPerfect, WordCraft, Scriptedit, or SGX 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 or illustrations will be included with articles depending on quality. Authors submitting diaries will receive the Transactor Disk for the issue containing their contribution.

Please Note: The Transactor’s phone number is: (416) 878-8438

Production

Attic Typesetting Ltd.

Printing

Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7, Canada. Second Class mail registration number 9042. USPS 735-9556, 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. ESS 0857-2330.

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 $5.00. U.S.A. $5.00. All other $2.00. 75.
Air Mail (Overseas only) $4.00. US. (15 post-age issue)

Send all subscriptions to: The Transactor, Subscriptions Dept., 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7. 416-878-8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address above will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

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 8.5 x 11, 31%, 80%, or 8x10 diaries with WordPerfect, WordCraft, Scriptedit, or SGX 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 or illustrations will be included with articles depending on quality. Authors submitting diaries will receive the Transactor Disk for the issue containing their contribution.

Please Note: The Transactor’s phone number is: (416) 878-8438

Production

Attic Typesetting Ltd.

Printing

Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7, Canada. Second Class mail registration number 9042. USPS 735-9556, 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. ESS 0857-2330.

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 $5.00. U.S.A. $5.00. All other $2.00. 75.
Air Mail (Overseas only) $4.00. US. (15 post-age issue)

Send all subscriptions to: The Transactor, Subscriptions Dept., 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7. 416-878-8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address above will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

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 8.5 x 11, 31%, 80%, or 8x10 diaries with WordPerfect, WordCraft, Scriptedit, or SGX 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 or illustrations will be included with articles depending on quality. Authors submitting diaries will receive the Transactor Disk for the issue containing their contribution.
Commodore's most recent worst kept secret is the Amiga 500 and Amiga 2500. So far I've heard about the new models from no less than 8 different people from as many different cities since the January CES in Vegas, and I've read about them in a couple of club newsletters published before the show even opened. Word is that several dealers who attended CES were offered a demonstration, but the machines were not "officially announced".

The 500 sounds curiously similar to the Atari 1040 ST. A self-contained unit, except for the external power supply, with a 3 1/2 inch drive (probably at one end), 1 Meg of RAM, and Kickstart in ROM. Apparently the ROM Kickstart will have enough "hooks" so that updates can be osmosed into the system. Price is supposed to open around $600 U.S., the same tag the Commodore 64 carried when it arrived.

The 2500 is being aimed at the business market, especially in the computer aided design field. The word "IBM compatibility" is getting tossed around again, with unconfirmed reports of IBM card slots and an onboard 8088. The case can be fitted with 3 disk drives in any combination of 3 1/2, 5 1/4, and hard disk. Price is supposed to be around the $2000 U.S. mark, but it's rumoured that Europe and Canada will get test market shipments before it's released in the States.

Back when the 1000 hit the market, Commodore said the Amiga would be a new line of machines, not just another sibling to be shoved out the door. Commodore should be proud to be keeping this promise. The Amiga is a fine product, and exploiting a good thing is good business. For this I tip my hat toward West Chester, because business is what Commodore needs to stay alive, and I, for one, want them to. However, it's the execution of this exploit that has me scratching my head.

Commodore wanted to keep news of the 500 and 2500 as quiet as possible, particularly the 500. They're afraid that too much early publicity would hurt sales of the 1000. But I don't think that should be their largest concern. (Even if I did, the publicity they'll get here probably won't affect a single sale - most computer hobbyists don't read The T. until well after they buy) Commodore intends to build the 2500 first, followed by the 500, and I think that's the wrong order.

If a whole neighborhood of $200,000 dollar homes were going up, but 2 months later you know another neighborhood not far away will have acres of similar homes for only $60,000, wouldn't you wait and see? I know I would. Now granted, the less expensive models might not appeal to me. But if they came along first, and had everything I wanted, I'd probably bite even if I could afford the luxury model.

Regardless, most won't have the extra 1400 or so dollars to sink into the Amiga 2500, and most won't need the added power. Perhaps Commodore wants this to make the 1000 more attractive at point of purchase. But the 500 is not much of a secret, and it would seem to me that less people would wait for the more expensive machine to arrive, as opposed to those who will wait for the less expensive Amiga. Introduce the 2500 first, and I believe sales will slow for both of the available models. Bring out the 500 first, and there will be popularity for it that includes those who couldn't afford the 1000. The small market ratio of potential 2500 users will have to wait, but not for long if Commodore gets moving on it.

Commodore shouldn't fear for the 1000 for other reasons. The price difference between the 1000 and 500 will not be that big, and unlike the 500, the 1000 will have the separate keyboard and the expandability to maintain its posture. Once all three models are out I believe none of them will be ignored at the counter.

Commodore has probably considered this situation very carefully, and their decision is one that undoubtedly spans much more than just one page of details. One report states that both machines should be ready by early 1987. If that happens, Commodore should fare pretty well whichever way they do it.

One thing to remember, all of the above is rumour. If Commodore shelves any or all of it, disappointment will be unjustifiable. On the other hand, when the new Amigas do arrive, Commodore should be looking forward to subsequent profitable quarters, and those shares I didn't buy back in '78 just might be worth looking into right now. Tomorrow, in fact.

Remember, there's nothing as constant as change, I remain...

Karl J.H. Hildon, Editor in Chief
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 five versions of VERIFIZER here; one for PET/CBM, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum 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, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 4099 to enable the Plus 4 version (off: SYS 4099)
SYS 3072, 1 to enable the C128 version (off: SYS 3072,0)
BANK 15: SYS 1024 for B128 (off: BANK 15: SYS 1027)

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).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

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 like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifier could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER 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: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a dataseat 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.

PET/CBM VERIFIER (BASIC 2.0 or 4.0)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI</td>
<td>10 rem+ data loader for &quot;verifier 4.0&quot; *</td>
</tr>
<tr>
<td>CF</td>
<td>15 rem pet version</td>
</tr>
<tr>
<td>LI</td>
<td>20 cs = 0</td>
</tr>
<tr>
<td>HC</td>
<td>30 for i = 634 to 754:read a:poke i,a</td>
</tr>
<tr>
<td>DH</td>
<td>40 cs = cs + a:next i</td>
</tr>
<tr>
<td>GK</td>
<td>50 :</td>
</tr>
<tr>
<td>OG</td>
<td>60 if cs&lt;&gt;15580 then print &quot;***** data error *****&quot;: end</td>
</tr>
<tr>
<td>JO</td>
<td>70 rem sys 634</td>
</tr>
<tr>
<td>AF</td>
<td>80 end</td>
</tr>
<tr>
<td>IN</td>
<td>100 :</td>
</tr>
<tr>
<td>ON</td>
<td>1000 data 76, 120, 173, 163, 2, 133, 144</td>
</tr>
<tr>
<td>IB</td>
<td>1010 data 173, 164, 2, 133, 145, 88, 96, 120, 165</td>
</tr>
<tr>
<td>CK</td>
<td>1020 data 154, 91, 2, 240, 16, 141, 164, 2, 165</td>
</tr>
<tr>
<td>EB</td>
<td>1030 data 144, 161, 163, 2, 169, 165, 133, 144, 169</td>
</tr>
<tr>
<td>HE</td>
<td>1040 data 2, 133, 145, 88, 96, 85, 228, 225, 217</td>
</tr>
<tr>
<td>CI</td>
<td>1050 data 201, 13, 208, 62, 165, 167, 208, 58, 173</td>
</tr>
<tr>
<td>PB</td>
<td>1060 data 254, 1, 133, 251, 162, 0, 134, 253, 189</td>
</tr>
<tr>
<td>PA</td>
<td>1070 data 0, 2, 168, 201, 32, 240, 15, 230, 253</td>
</tr>
<tr>
<td>HE</td>
<td>1080 data 165, 253, 41, 3, 133, 254, 32, 236, 2</td>
</tr>
<tr>
<td>EL</td>
<td>1090 data 198, 254, 16, 249, 232, 152, 208, 229, 165</td>
</tr>
<tr>
<td>LA</td>
<td>1100 data 251, 41, 15, 24, 105, 193, 141, 0, 128</td>
</tr>
<tr>
<td>KB</td>
<td>1110 data 165, 251, 74, 74, 74, 74, 24, 105, 193</td>
</tr>
<tr>
<td>EB</td>
<td>1120 data 141, 1, 128, 108, 163, 2, 152, 24, 101</td>
</tr>
<tr>
<td>DM</td>
<td>1130 data 251, 133, 251, 96</td>
</tr>
</tbody>
</table>

VIC/C64 VERIFIER

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

VIC/64 Double Verifier Steven Walley, Sunnymead, CA

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifier-displayed codes. DOUBLE VERIFIZER solves that problem by showing the two-letter verifier code on both the first and second row of the TV screen. Just run the below program once the regular Verifier is activated.

May 1987: Volume 7, Issue 06
VERIFIER For Tape Users

Tom Potts, Rowley, MA

The following modifications to the Verifier loader will allow VIC and 64 owners with Datasets to use the Verifier directly (without the loader).

After running the new loader, you will have a special copy of the Verifier program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/64 VERIFIER loader:

```
NB 30 for i=850 to 980: read a: poke i,a
AL 60 if cs<>14821 then print "*****data error*****": end
IB 70 rem sys850 on, sys853 off
  80 delete line
  100 delete line
OC 1000 data 76, 96, 3, 165, 251, 141, 2, 3, 165
MO 1030 data 251, 169, 121, 141, 2, 3, 169, 3, 141
EG 1070 data 133, 90, 32, 205, 3, 199, 80, 16, 249
BD 2000 a$=" verifier.sys850[space]"
KH 2010 for i=850 to 980
GL 2020 a$=a$+chr$(peek()): next
DC 2030 open 1,1,1,a$: close 1
IP 2040 end
```

Now RUN, PRESSING PLAY AND RECORD when prompted to do so (use a rewound tape for easy future access). To use the special Verifier that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode:

```
OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifier loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.
```

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIER first will usually result in a crash. If you wish to use VERIFIER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Plus 4 VERIFIER

```
NI 1000 rem + data loader for "verifier + 4"
PM 1010 rem + commodore plus4 version
EE 1020 graphic 1: sncir: graphic 0: rem make room for code
NH 1030 cs=0
JI 1040 for i=4096 to 4216: read x: poke j,x: ch=ch+x: next
AP 1050 if ch<>1314 then print "checksum error": stop
NS 1060 print "sys 4096: rem to enable"
JC 1070 print "sys 4099: rem to disable"
ID 1080 end
PL 1090 data 76, 14, 16, 165, 211, 141, 2, 3
CA 1100 data 165, 212, 141, 3, 3, 96, 173, 3
OD 1110 data 3, 201, 16, 240, 17, 133, 212, 173
LP 1120 data 2, 3, 133, 211, 169, 39, 141, 2
EK 1130 data 3, 169, 16, 141, 3, 3, 96, 165
```

C128 VERIFIER

```
PK 1000 rem + data loader for "verifier c128"
AK 1010 rem + commodore c128 version
JH 1020 rem + use in 40 column mode only!
NH 1030 cs=0
OK 1040 for j=3072 to 3214: read x: poke j,x: ch=ch+x: next
IP 1050 if cs<>17860 then print "checksum error": stop
MP 1060 print "sys 3072, 1: rem to enable"
AG 1070 print "sys 3072, 0: rem to disable"
GD 1080 end
KD 1090 data 208, 24, 105, 193, 141, 1, 12, 108, 211
SE 1100 data 0, 165, 210, 24, 101, 208, 133, 208
BG 1110 data 96
```

B128 VERIFIER

```
1 rem save @:verifier128.b",8
10 rem + data loader for "verifier b128"
20 cs=0
30 bank 15:for i=1024 to 1163:read a: poke i,a
40 cs=a+:next i
50 if cs<>16828 then print "** data error **": end
60 rem bank 15: sys 1024
70 end
1000 data 76, 14, 4, 165, 251, 141, 130, 2, 165, 252
1010 data 141, 131, 2, 96, 173, 130, 2, 201, 39, 240
1020 data 17, 133, 251, 173, 131, 2, 133, 252, 169, 39
1030 data 141, 130, 2, 169, 4, 141, 131, 2, 96, 165
1040 data 1, 72, 162, 1, 134, 1, 202, 165, 27, 133
1050 data 233, 32, 118, 4, 234, 177, 136, 240, 22, 201
1070 data 234, 32, 110, 4, 198, 234, 16, 249, 200, 208
1080 data 230, 165, 233, 41, 15, 24, 105, 193, 141, 0
1090 data 208, 165, 233, 74, 74, 74, 74, 24, 105, 193
1100 data 141, 1, 208, 24, 104, 133, 1, 108, 251, 0
1120 data 164, 137, 133, 132, 134, 32, 38, 186, 24
1130 data 32, 78, 141, 165, 133, 56, 229, 136, 168, 96
1140 data 170, 170, 170, 170
```

The Transactor

May 1987: Volume 7, Issue 06
Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in — if we use it in the Bits column, we'll credit you in the column and send you a free one-year's subscription to The Transactor.

Using the WAIT Function
For Screen Changes
Francis O. Saffell
Eugene, OR

If you've worked with graphics on the C-64 you know that changes in sprite values, screen colour and bank selection can often look jittery. This is because you caught the raster beam off guard with your change. For nice clean transitions you can ask the computer to wait till the raster is out of sight before making the change:

```
WAIT 53265, 128
```

...will stop your program until the beam is out of sight.

"Last File used" Update
Dan Schein
West Lawn, PA

In the Bits section of Volume 7, Issue 04 (Jan 1987), Dave Newberry submitted a SYS statement to print the last filename used on a 64. Here are equivalent SYS commands for other Commodore models.

```
VIC-20 SYS 63065
C-64 SYS 62913
Plus/4 SYS 61810
C-128 SYS 62753
```

Easy C-128 Un-New
Michael D. Paul
Mansfield, Ohio

Everyone can always use a program to restore and accidentally NEW'd BASIC program. Here's one for the C-128 that uses only 12 bytes and can be located almost anywhere. I store it on disk and use it by typing BOOT "UNNEW".

```
LDA $08 ; device number
TAX
LDA $00 ; no secondary command
JSR $FFBA ; set file parameters
LDA $00 ; no file name
JSR $FFBD ; set name
JSR $FFC0 ; open channel
LDA $04 ; logical file number
JSR $FFC9 ; assign default output device
BRK
; return to monitor
SCREEN JSR $FFCC ; clear channel
BRK
; return to monitor
```

Printer output from an ML monitor
David O. Rowell
Marietta, NY

It seems that the folks who wrote the excellent ML monitor programs for the C-64 forgot that a printer would be handy for many uses within the monitor. This omission bugged me for quite a while before I gave in and worked out a solution.

The following bit of machine code will allow you to toggle printer output on and off:

```
PRINTER LDA #$04 ; logical file number
TXA ; device number
LDY #$FF ; no secondary command
JSR $FFBA ; set file parameters
LDA #$00 ; no file name
JSR $FFBD ; set name
JSR $FFC0 ; open channel
LDA #$04 ; logical file number
JSR $FFC9 ; assign default output device
BRK ; return to monitor
```

This little routine has two entry points: PRINTER to turn the printer on (as in OPEN 4,4: CMD 4) so that everything that would have printed on the screen now goes to the printer, and
SCREEN to return output to the CRT. Decide where you want it in memory and use the assembler function of the monitor to put it there - it's completely relocatable so it can go anywhere. Remember the address of the two entry points so that you can use the .G option of the monitor to jump to the function you need. You may want to save it on tape or disk for the next time, maybe along with the monitor itself.

Quick Directory Hider

Doug Resenbeck
Centrailia, IL

A friend of mine, Bill Kir, came across yet another trick that can be done to the BAM on a disk. This routine of mine makes good use of his find. When run, you are given the option to 'hide' a directory from loading and listing to the screen or 'recover' a directory that has been previously hidden. One thing to keep in mind is that once a directory has been hidden or recovered, you won't be able to see the changed directory unless you reset the drive or remove and re-insert the diskette. This is because the drive keeps a copy of the directory in its buffer. Anyway, the program will hide any of your directory from being LOADED and LISTed, but will not prevent any of the programs or files on disk from being accessed. Is there any way to view a directory without recovering it first? Yes there is. Just load in the hidden directory, POKE 2076,50 and SYS 42291. Now list the directory. Have fun!

P.S. Please print my complete address.
O.K. Here it is . . .

Doug Resenbeck
Box 7711 N-00592
Centrailia, IL 62801

0 rem directory hider - doug resenbeck
1 open 15,8,15:open 2,8,2,"#";print#15,"u1";,2;0;18;0
:prient#15,`;p-`;2;165
2 print "1) hide 2) recover ",wait 198,1:geta:POKE 198,0
:a=a-1:ifa<0ora>1then2
3 if a=1 then a=50
4 print#2,chr$(a):print#15,"u2";,2;0;18;0:close 2:close 15

"High-Res 128" Video Fix

Erik J. Palm
Rockford, IL

Paul T. Durrant's program, "Commodore 128 High-Res Graphics", is a wonderful utility for 128 users who wish to gain an extra 16K of storage or for those who want to exploit the 128's extra screen resolution (Volume 7, Issue 2, page 72). Readers with newer model 128's may have noticed a flashing vertical line on the far right of the 80 column display when using this program. This occurs because of a minor chip modification at Commodore. There is a simple fix however. Just change:

00BC9 A9 80 LDA #$80

To:

00BC9 A9 87 LDA #$87

Little-Known Features of DOS

Gavin Bell
Princeton, NJ

Commodore DOS has some nifty features that Commodore doesn't tell anybody about. For example, both the directory command ($) and the scratch command can take multiple arguments, separated by commas. For example:

open 15,8,15,"s0:green,blue,frog,???,t*":close 15

...will scratch the files "green", "blue", "frog", any files with three-character names, and any files with names starting with the letter "t". Similarly,

load ",a*,b*",.8

...will load a directory containing all filenames beginning with either "a" or "b". The number of arguments is limited only by the 40-character command length limit, and any kind of pattern-matching and drive specification will work as expected. For example,

"s0:test,1:tester"

as a disk command will scratch "test" on drive zero and " tester" on drive one of a dual drive.

The Amazing 1660 Modem

Kevin Kleca
Connellsville, PA 15425

One piece of Commodore hardware is the 1660 modem. One of the unique features of the modem is that you can make it do tone dialing by generating the tones with the SID chip the way you would normally produce sound through your TV or monitor. Many people don't realize that any sounds produced by the 64 while the modem is 'off-hook' will be sent over the phone line. This opens up some interesting possibilities. With a 1660 modem and some sort of speech synthesizer like S.A.M., you
can very easily turn your Commodore 64 into an answering machine. Just use the short program below with your speech package and your 64 will answer away! It can easily be changed to take the time of the call, or do whatever else you wish. Maybe if you're good with electronics, you could connect a voice recognition system to the modem and have it answer or digitize the caller response! When using the program, make sure your modem is set on ANSWER, not ORIGINATE.

90 oh = 56577: hi = 32: lo = 255–32: c = 0
100 print "waiting for call."..."
110 if (peek(oh) and 8) then 110
120 poke(oh + 2), (peek(oh + 2) or hi)
   : poke oh, (peek(oh) and lo)
130 print "phone ringing, now answering."..."
140 rem insert commands for your speech program here
150 rem with the speech that you want the caller to hear
160 poke oh, (peek(oh) or hi): c = c + 1
170 print "calls answered." : c:
180 goto 100

Bit Correction: DOS Wedge

Robert C. Kodakek
Aston, PA

In the Bits column, Volume 7, Issue 4, reader Joel Pickett wrote in to say that the DOS wedge program supports only one 1541 drive. Actually, the DOS wedge (version 5.1) has a command for changing the device number — the "#". The proper syntax for the little-known command is:

@#(device number)

For example, to change from device 8 to device 9, the command is: @#9. The use of this command is certainly easier than going to the trouble to change the loader program and having to remember to POKE address 52343 with the current device number.

Correction: Format Track 36

Thomas E. Calise
Brick, NJ

I have found an error in Mr. D.A. Hook's program (format track 36) that was published in the September '86 Transactor. After running the program and checking the extra formatted track for errors using the 1541 SuperKit Super Scan, I continuously came up with a *20 read error for sector *16. Upon inspection of the code presented in his article, I realized that the wrong number was being stored in location $43. Instead of a $10 it should be a $11. To correct the situation, change line 1020 in the program to:

1020 data 141, 34, 6, 169, 17, 133, 67, 32

After the change was made, all sectors checked out properly.

Assembler Start–up Code

Andrew Walduck
Orillia, Ont.

Here is some assembler code, in PAL-compatible format, that I use as the start of many of my programs. It makes a program that can be LOADED and RUN like any BASIC program, and the user doesn't have to remember a SYS command to get it going. When the program is loaded, it will appear as the BASIC line "0 SYS 2062", and when run, control will be passed to the machine code that follows.

100 open 15.8.15, "s0:program name": close 15
110 open 3.8.3, "0:program name,p,w"
120 sys 700 : start up pal
130 .opt o3 : output object to disk
140 r = $0801 : put program at start of basic
150 .wor $080c : link address
160 .wor $0000 : line number zero
170 .byt $9e : token for ‘sys’
180 .asc " 2062": ascii literal for sys address
190 .byt $00 : end of basic line token
200 .wor $0000 : end of basic program
210 :
280 :your source code should start here

C128 RS–232 Bugs

Albert J. McCann, Jr.
Glenolden, PA 19036

There are two bugs in the Commodore 128 in the RS–232 routines that interfere with RS–232 operation. Both bugs have to do with the carry flag in the 8502 being set to the wrong state upon exiting the routines.

The first bug is in the "BASIN" code for RS–232 input. The code is as follows:

EF65 CLC
EF66 RTS
EF67 JSR $EEFD : get rs–232 byte
EF6A BCS $EF65 : exit if carry set, branch to wrong loc.
EF6C CMP #00: null
EF6E BNE $EF66 : if valid char, exit. carry set as a result
               : of compare, branch goes to wrong loc.
EF70 REST OF CODE

The changes needed to make this work are:

EF6A B0 FA BCS $EF66
EF6E D0 F5 BNE $EF65

The second bug is even nastier. It is in the RS–232 OPEN routines and prevents opening an RS–232 "X-LINE" handshake channel. The following open statement will work on a 64 but not on a 128 because the 64 ignores the carry flag.
OPEN 2,2,0,CHR$(6)+CHR$(1):REM 300 BAUD, FULL DUPLEX, X-LINE HANDSHAKE

When this executes, you will get random errors and it will stop a BASIC program. The code is as follows:

F094 LDA $0A11 ;rs-232 command reg
F097 LSR A ;bit 0 to carry
F098 BCC $F0A3 ;branch if 3-line rs-232
F09A LDA $DD01 ;check if dsr missing
F09D ASL A ;bit 7 to carry
F09E BCS $F0A3 ;if input pin is not connected, or else is "high" then carry = 1, dsr is "true high" at this point
F0A0 JSR $E755 ;set status variable to indicate dsr is missing, input pin is "low". this routine clears carry so if you ground the dsr input pin then the above open will work

The first thing the program does is restore I/O by calling J0INIT at $FF84. Then it jumps to the monitor. This vector has many uses, for example making a 'reset-proof' program, or with an un--new routine to restore any basic program in memory.

Note to David Mora: we can't find your complete address! Please give us a call so that we can send you your free subscription. -CZ

FO03 REST OF CODE:
FO0F RTS

There is not enough room at this point to include a CLC instruction so I jump to a patch. The code to fix this is:

F00C 4CF3 FE JMP $FEF3 ;jump to patch
FEF3 8D1A 0A STA $0A1A ;original code
FEF6 18 CLC ;this is added
FEF7 60 RTS ;exit

These changes were put into a 27128 eprom to replace the KERNAL rom in the 128. I use Promenade which works nicely in 64 mode. While you're poking around the KERNAL making fixes, here is one that is unrelated to RS-232 but bugs many people:

FC22 D1

This fixes Caps-Lock Q. Also, to replace the KERNAL rom in the 128, you will need two computers: a 128 and a 64 or second 128. The reason being that there is 4K of invisible rom (Z-80 code). This is not accessible from 128 mode. The KERNAL rom is chip U35. You must remove this chip and read it into the second computer through the eprom programmer. Then you can modify the code and send it to an eprom.

Moving The Cassette Buffer

John Tellefon
Salina, KS

The following tip applies to VIC 20 and 64 users who use tape for storage. It may apply to other Commodore computers.

Many small utilities such as "Verifizer" are written to run in the tape buffer, and any tape operation clobbers the utility. This can be avoided by moving the tape buffer -- in the VIC and 64, the buffer is located by addresses 178-179, and the system doesn't mind if you move it. It is 192 bytes long, and you can put it anywhere in free RAM as long as it doesn't overwrite your program, variables or ROM. If you only SAVE and LOAD from direct mode, it doesn't matter if you overwrite variables, since they will be cleared the next time you RUN anyway. Here is a sample code fragment that moves the tape buffer to 192 bytes below the bottom of string storage space:

```
a = peek(51) + 256*peek(52) - 192: poke 179,a/256 : poke 178,a-256*peek(179)
```

(if memory is tight, you may want to check 'if fre(0)>192' before doing the above allocation).

An extra advantage: With the tape buffer moved, you can put code in the tape buffer's normal place and save it to tape with a monitor. You can then load the program in and use it at its intended location.

Reset and Run

Noel Nyman
Seattle WA

Here's a way to have your favorite BASIC program available at the push of a button without building the hardware described in "C64 RAM Cartridge" (Volume 7, Issue 4). Make the following changes to listing #1 on page 52:

The Transactor

May 1987: Volume 7, Issue 06
Change the file name in line 1090 to "0.nocart"
Change the end of loop in line 1110 to 33008 (from 32999).
Change the checksum in line 1120 to 29267 (from 28345).
Change line 1290 to read:

1290 DATA 198, 76, 232, 128, 56, 165, 46, 229
Add line 1291:
1291 DATA 0, 0, 169, 128, 133, 56, 108, 2, 3

Run the modified loader to create "nocart" on disk. Then use "nocart" and follow the instructions in the article to store your favorite BASIC program in RAM above $8000. Then enter the command:

POKE 56,128: CLR

This protects the RAM area above $8000 from BASIC string variables. The code you added to the loader will perform this function again whenever your stored BASIC program is RUN.

Using SYS 64738 or hitting a reset switch will RUN the stored BASIC program, just as if it were in a RAM cartridge. The only difference is that the program is wiped out when the computer is turned off.

**Border Animation!**

Terry Montgomery
Walls, MS

Ever try to display anything in the screen border? Other than changing the colour, it's pretty hard to get the border to do anything exciting. This machine language program from Terry Montgomery, however, will put a colourful display in your screen borders that you never thought possible! Not only that, but the program is interrupt-driven, meaning that you can edit and even run most BASIC programs while the display continues. A great addition to your "Gee-Whiz" collection.

```
10 rem+ data loader for "colour bars" +

LI 20 cs = 0
KG 30 for i = 49152 to 49394: read a: poke i,a
DH 40 cs = cs + a: next i
GK 50 :
EC 60 if cs<>29666 then print "!data error!": end
EI 70 rem sys 49152
AF 80 end
IN 100 :
KI 1000 data 120, 169, 127, 141, 13, 220, 169, 1
MJ 1010 data 141, 26, 208, 173, 17, 208, 41, 127
DB 1020 data 141, 17, 208, 169, 36, 141, 20, 3
DA 1030 data 169, 192, 141, 21, 3, 169, 0, 141
GC 1040 data 18, 208, 88, 96, 173, 25, 208, 41
KN 1050 data 1, 208, 3, 76, 101, 192, 169, 1
IK 1060 data 141, 25, 208, 172, 176, 192, 185, 179
KJ 1070 data 192, 141, 32, 208, 238, 176, 192, 174
AA 1080 data 176, 192, 236, 177, 192, 208, 8, 32
OE 1090 data 104, 192, 162, 0, 142, 176, 192, 173
DN 1100 data 17, 208, 41, 127, 141, 17, 208, 185
HM 1110 data 191, 192, 141, 18, 208, 173, 176, 192
NJ 1120 data 208, 3, 76, 49, 234, 76, 188, 254
CK 1130 data 238, 178, 192, 173, 178, 192, 201, 15
MO 1140 data 176, 14, 160, 0, 185, 204, 192, 153

PP 1150 data 191, 192, 200, 192, 12, 208, 245, 96
CN 1160 data 173, 176, 192, 201, 30, 176, 14, 160
PG 1170 data 0, 185, 217, 192, 153, 191, 192, 200
GA 1180 data 192, 12, 208, 245, 96, 173, 178, 192
HN 1190 data 201, 45, 176, 14, 160, 0, 185, 230
EO 1200 data 192, 153, 191, 192, 200, 192, 12, 208
GC 1210 data 245, 96, 169, 0, 141, 178, 192, 96
GJ 1220 data 2, 12, 33, 6, 9, 5, 1, 12
IO 1230 data 8, 4, 2, 7, 0, 14, 6, 50
AE 1240 data 55, 60, 65, 70, 75, 80, 85, 90
EN 1250 data 95, 100, 105, 40, 50, 70, 90, 110
PN 1260 data 130, 150, 170, 190, 210, 230, 250, 14
GO 1270 data 40, 50, 60, 70, 80, 90, 100, 110
IC 1280 data 120, 130, 140, 150, 160, 170, 50, 55
GG 1290 data 60, 65, 70, 75, 80, 85, 90, 95
MF 1300 data 100, 105, 110

Verifier and Mike Iafrate
Fast-Load Cartridges
Parkersburg, WV 26101

I was having trouble using the Verifier for the 128: when I ran the program, sometimes it would work, and sometimes the monitor would break in. After some troubleshooting, I found that the Mach128 fast load cartridge was causing the problem. I have come to depend on it so much that I often forget that it is there and usually have no problem with most of the software that I use.

Here is the problem as far as I can determine. When I use the cartridge to load a program, there are two symbols that are substituted for the LOAD command: a slash, and the up-arrow. These use some type of an interrupt that sets the Z-flag and the B-flag in the status register. I haven’t seen too much written about the B-flag (break flag). Anyway, when I remove the cartridge I find that the status register returns to zero, and then I can load and use the verifier normally. It’s no big deal to fix the problem; I just add this line to the program:

1080 poke 05.0: end

If anybody else is having problems using Verifier 128, maybe this will fix the problem. I am sure there are a lot of people using the Mach128 cartridge.

Amiga Bits

The Vanishing Workbench

Did you ever slide down the screen of some application, expecting to find the Workbench screen patiently awaiting your return, only to find that it had vanished? This happens when an application closes the Workbench screen in an effort to save memory. Examples of programs that do this are ABasic (the original BASIC shipped with Amigas), and Deluxe Music from Electronic Arts. This may be a fine way to save memory, but life can be difficult with no Workbench screen - it makes it impossible to run anything else until you exit your program! Fortunately, the system won't let anyone close the Workbench
screen if there are any windows open on it. So, if you want to make sure you don’t lose your screen, bring up a window before you launch your Workbench-closing application. A good candidate to have up is a CLI window, since you’ll probably want one around anyway for doing disk operations from time to time.

**Beating The Low Memory Blues**

If you can’t bring a program up because you’re running low on memory, try closing as many windows on the Workbench as possible. You can close every window on the screen, even the one containing the program you’re trying to run, by dragging the program’s icon outside the window onto the Workbench screen (actually, backdrop window) itself. Once you close up everything you can, you can launch the program by double-clicking the icon as usual. Make sure you eventually drag the icon back into the drawer it came from, or it’ll disappear from the disk forever.

If you want to keep some windows around, you can minimize the memory consumed by making sure they don’t overlap. If you’re totally desperate and absolutely need a few more K, you can always unplug your second drive (if you have one), and reboot. If you don’t have the internal 256K RAM expansion module, get it. If you can afford to, get more – it’s amazing what a few extra Megabytes can do for Amiga!

**Modifying The Epson Printer Driver**  
**Peter Inskeep**  
**To Work With Other Printers**  
**Long Valley, NJ**

I use the Scribble! word processor, version 2.0, and a Gemini Star 10-X printer. If Epson is selected as the printer through Preferences, many of the printer commands work properly on the Start 10-X. One command that does not work is the paragraph return, as would be the case when putting a space between two paragraphs. Invariably, the printer will skip not one, but two lines, before starting the next paragraph. Scribble! is not aware of this extra line skip, so it does not add the extra line when calculating where the end of the page really is. The more blank lines between paragraphs, the more the program gets out of line at the end of the page.

Creating a custom printer-driver is a formidable task. But, if the Epson program is to be modified only slightly it can be made to work well with the Star 10-X. I used the “Edit” program on the WorkBench disk to make the modification directly to the Epson printer driver.

The Epson printer driver is located with all the others in the “devs/printer” directory. Using the command, “type dfo:devs/printer/epson.opt h” from CLI to display a hex dump of the file, I located the characters that controlled the linefeed for the printer. This was at location $0430 (hex), where the characters read as follows:

```
$0430: 0000FF00 0A00D0A0 0000FF00 ....5.-
```

The carriage return/linefeed at $0436 and $0437 (in bold above) were causing the double-skip. In preparation for surgery, the file was copied to a new file called “Star” on a spare disk.

Edit was then invoked to make the correction. Because some of the “lines” in the file were very long, it was necessary to specify the unusually long line length to Edit, or the lines would be truncated and the file corrupted. The Edit command used was “edit star opt p65w550”. The 000D0A combination was found in lines 12 and 13. Line 12 contains 000D, but only the 00 shows if the ! Edit command is used. Line 13 contains only 0A (you can’t see it, since it’s a linefeed, but it’s there). Line 12 was deleted and replaced using the control-2 and control-j keys. Here are all the edit commands required:

```
113
CTRL-2 CTRL-j
Z
```

The instructions for using Edit can be found in the AmigaDOS manual. Since the control-2 and control-j key combinations are non-printing, you will not see anything being added, but you are putting the two characters 000A into the file.

Line 13 was deleted, because the 0A at $0437 was really the bug for the Star 10-X. This left the file short one character, so 00 was added at the beginning of line 14 to put 000A at $0436. To add the character to the beginning of the line, before the 0A, a substitute null string command was used:

```
A/CTRL-2/
```

After Edit was wound up, the newly edited file “Star” was checked for size with the list command to make sure it was the same size as “Epson”. Type star opt h’ was used to see if the changes had been made in the right place. The corrected file read as follows:

```
$0430 0000FF00 0A00D0A0 0000FF00 ....5.-
```

“Star” was then copied to the Scribble disk (copy star to scribble! /devs/printers). Preferences would recognize the new printer file, but for some reason Scribble! declared “no printer” when the print menu was selected. To keep Scribble! happy, I renamed “Epson” to “True Epson”, and “Star” to “Epson”. After that, there was no problem.

Although I have not done it, I assume that many of the printer drivers could be similarly modified. It took patience to figure out how to use Edit, especially with non-printing characters. I put the whole mess in the ram disk (“ram:”) while I experimented, which made all the trial and error go much faster.

By the way, I am really enjoying Scribble! Micro-Systems sent the Scribble! 2.0 upgrade, with the spelling dictionary, as soon as it was available, and enhanced Scribble! at the same time. It was a real bargain, because there was no charge for the upgrade.
Letters

We get a lot of mail from readers who have very specific questions about their Commodore and third-party equipment. Many of these are interfacing problems of one kind or another: which software will work with my modem? How can I get my printer and my word processor to work together? How does my system load from drive 13 now that I've installed a Magic Shmoo(tm) Expander box on my game port?

These kinds of questions are often impossible to answer unless you have specific experience with the particular equipment configuration involved. Since we don't have every imaginable piece of equipment on the market, we're often left scratching our heads for a reply.

You, on the other hand, may have recently discovered the necessary incantations to make the Magic Shmoo(tm) box work with your disk drives, and would just love to share your discovery with the rest of the world. That's what this next bunch of letters is all about. Read them over - see if you can help. We'd love to hear from you, and so would the authors of these questions.

Angling for weird parts: Because of a stiff cord on my (replacement!) power pack, and a lot of movement of the keyboard by our four users, the power receptacle broke its solder joints to the PC board. A simple removal of the PC assembly, lifting off the RF shield and careful resoldering have cured the problem. However, it could have been worse, and that leads me to the following subject.

I tried to locate a 7-pin DIN male by female right angle adapter so I could lead my power cable out the rear of the board. This idea carried over to also trying to source an angle adapter for the rear serial receptacle on my 801 printer (the cable interferes with the paper feed). My intention was to fasten these to the cases so flex loads would be taken by the easily repaired cords.

I could not find a supplier in this area (250 km out of Vancouver). Could you suggest a supplier of the following items I would like to purchase: right angle DIN m/f adapters; longer serial cords or m/f serial cord extenders; a simple inexpensive modem.

Doug Hurd, Penticton, British Columbia

Yet another request for Urdu software: I have been a regular subscriber to The Transactor for the last two years. I have owned a faithful C-64 for the last four years! I have recently seen an Arabic version of a word processor that is on a cartridge that converts the C-64 to an Arabic language computer.

I am interested to know if there is any such software for the Urdu language available? I would be grateful to any Transactor readers who could provide me with information on such a product.

M. Sharif Butt, Makkah, Saudi Arabia

C-64 Numerics: I enjoyed the 'extra' languages issue, especially the benchmarks.

What numeric calculation software environments exist for Commodore 64? Any public domain stuff?

The "Apple Numerics manual" (ISBN 17741-2 Addison-Wesley) describes SANE, the Standard Apple Numerics Environment, which runs on the 6502 microprocessor in assembly language. Has something like this been written for a Commodore system?

"Interactive Simulation Language" (Interactive Mini Systems Inc.) with the "ILMPC" compiler is available for Commodore 64. It includes modules for solving non-linear differential equations.

I bought something from Dynacomp written in BASIC with no error trapping and clumsy plotter routines for C64. Runs, but not what I'd hoped to find.

Edward Connors, Frederick, Maryland

Remote line feeds: I have enjoyed The Transactor for many issues, especially Networking and Communications. The program I would like to refer to is Remote 64.

After changing 16 to 0 in line 1000 to give full duplex transmission, this brought my non-Commodore terminal to life and I can interact with my 64 remotely.

The only problem is that my remote terminal expects a linefeed (chr$10), and unless my programs are modified to give one, the terminal will just write over the same line. Program and directory lists seem impossible.

Is there a simple method to send a linefeed after a carriage return by modifying Remote 64?

This problem has been giving me many sleepless nights and an answer would be appreciated.

Tim Rayworth, Kentville, Nova Scotia

Nefarious plots: I am wondering if anyone has noticed any bugs in the 1520 plotter when attempting to plot in relative coordinates.

It seems that when the pen carriage is at certain x (horizontal) positions, the first 't' sub-command after an 'i' sends the pen off to the absolute origin instead. To cure this, it is necessary to insert the apparently redundant subcommand PRINT*1, "R",0,0, after PRINT*1,"I".

The 1520 seems to be a rather unpopular peripheral and I haven't come across another unit to test whether all have this fault or just mine.

Michael Seman, South Perth, Western Australia

Put away those sleeping pills, Tim, and add the following lines to the Remote 64 source file:
I use a Pocket Modem and a terminal program called Dark Term on my C64. Both the modem and the terminal program have autodial capability, but they are unable to sense a busy signal, ringing phone, or an answered voice signal. This meant that I had to monitor the call by holding a telephone to my ear. Many are the headaches caused by a detected carrier. I have overcome this annoying problem with two items found in a local Radio Shack store. The first is called a Phone Pickup Coil, part number 44–533; the second is called an Audio Amp/Speaker, part number 277–1008.

The phone pickup coil is normally connected to a tape recorder and then fastened to the telephone handset via a rubber suction cup, to allow phone conversations to be recorded. However, when connected to the Audio Amp/Speaker, the phone conversations can be heard without listening to the telephone handset. Now, if the pickup coil is placed on the front right corner of the Pocket Modem case, and the modem is turned online, the speaker will broadcast the dial tone, or the telephone conversation, just like it did with the telephone. This means that I can now use my modem and term program without having to hold my telephone to my ear.

I use the above setup every time I use my modem. With it I can check for following:

1) A dial tone when I first turn on my modem.
2) Listen to how my modem dials the phone number.
3) Listen for the phone to ring on the other end, or know immediately if the line is busy.
4) I can hear the other computer send its carrier signal and tell if my computer has answered it.
5) If I don’t get an answer after three rings I hang up and check the number.
6) If I should dial a voice number I will hear them on the speaker and can then pick up the phone and explain or apologize.
7) When I am uploading or downloading, I leave the speaker on and listen to the data being transferred. I have found on a number of occasions, when my computer mysteriously locked up while uploading, that the problem was caused by the BBS hanging up on me. Without the pickup coil and speaker setup, or holding the phone to my ear, I never would have figured out the cause.

I have installed the pickup coil semipermanently on my Pocket Modem with a dab of hot glue (Crazy Glue or Epoxy will also work). The Hot Glue can be peeled off, and it leaves no marks if I should decide to sell my modem. Also instead of removing the rubber suction cup I turned the coil upside down: the coil will still work, and one day I may figure out a use for the suction cup behind the computer.

The coil and speaker should work on any type of modem. Connect the coil to the amp/speaker, turn on the modem and pass the coil over the modem case until the dial tone is heard. It will be necessary to experiment to find the loudest signal and then just glue the coil in that spot. The beauty to this setup is that no permanent modifications are necessary to the computer, modem or phone lines. Also, there is no way you can damage the computer or modem, or mess up your phone service. The only disadvantage to the system is that the coil will pick up a hum if placed too close to the monitor or TV.

Eavesdropping on modems: Please let me take this opportunity to thank you for a truly informative magazine. Many of my projects, programs and most of my experience is directly related to the articles you have published.

This letter was inspired by a couple of articles concerning modems in recent issues of the Transactor.
Note: If you are curious about why people have trouble with disk drives and Datasettes when placed on the left side of the monitor, just pass the coil by the left side of your monitor and listen. I would suggest you keep the volume turned down!

Perhaps the neatest thing about this project is the price. The pickup coil costs $2.19 and the Amp/Speaker is worth $15.95 at Radio Shack. This speaker and coil modification is a lot like my Fastload Cartridge: now that I have used it, I won’t leave home without it.

I hope that this project will prove as useful to you as it has to me.

Kevin Lemon, St. Catharines, Ontario

IEEE for C-128: In response to your query in the March ‘87 issue about whether there is an IEEE interface for the C-128: there is... the Quicksilver 128 from Skyles Electric. I have been using one of these for about two months on a C-128 equipped with a Hewlett-Packard Thinkjet printer and 7470A two pen plotter. The device plugs into the expansion port and requires replacement of a ROM and attaching a jumper clip inside the computer. An important feature is that this unit also works with the C-128 running in C-64 mode.

Unlike the older BusCard II for the C-64, this interface does not have a built-in monitor or enhanced DOS commands. Although not needed for 128 mode, they are really missed in C-64 mode. It also does not have a parallel printer output.

As advertised, the Quicksilver seems to be fully transparent. However, one bug exists if it is to be used in C-64 mode. If you turn on the C-128 and ‘GO 64’ immediately, you will not be able to access the IEEE bus in C-64 mode. The solution is to open and close any IEEE device while passing through C-128 mode. This could be handled with an autostart disk routine in final versions of application programs.

John A. Spencer, Edwardsville, Illinois

Amiga marketing: Thank you for Vol 7, Issue 5, first for the inclusion of articles on the Amiga; and second for the excellent Start Address.

Although the Amiga articles are over my head, at least they are a recognition that you plan to support the model. I am one of those who are still trapped in an 8-bit orientation in a 32-bit environment. I still want to Load and Run with a simple Shift/Run and to call up a directory with a caTd0 or 1. I yearn to be able to print a directory or program listing with an open4:4:cmd4(caT or list):print*4:close4. That was so automatic. Now I am beginning to master cd dfl: dir opt a, but I cannot get it to print even if I type dir opt a to ptr: And I am so tired of getting something about a command lacking a return code and I cannot find anything in the manuals on “return codes”? And the manuals were not written for such an. How about some articles for the alphabetabates in the Amiga audience?

On the second point, I agree with your editorial wholeheartedly. But another way that Commodore has repeatedly shot itself in the foot is in the poor support they have given their distributors. Two local outlets have said that they will never deal with Commodore again. I realize that this is a holdover from the Tramiel era, but the current superstructure at Commodore does not seem to have addressed this problem adequately. Also, Commodore seemingly does not know how to win the press. The PC caught on only because of the familiar three initials. The computer press followed a familiar trail and it established a “standard” with a poorly designed machine. But success breeds success and the press seems intent on burying the name Commodore. Granted that the rumoured 150,000 installed systems may not excite software developers, some good and effective PR effort on the part of Commodore devoted to the seemingly hostile part of the computer press could effect a drastic change. It would not be easy but it could be vital.

One footnote to my first point: the recent rumour that Commodore has abandoned the Amiga 1000 as of the first of the year in favour of the “2000” and a “500” does nothing to inspire confidence in those of us with the 1000. This tactic seems to have been taken out of IBM’s manual of deserting the loyal customers.

Rev. Fr. J.Paul Morris, Long Beach, California

Having experienced it ourselves, we sympathize with anyone going through the vertigo you get when you move up to the Amiga from the 8-bit machines. But as it happens, sending a directory to the printer is easy and natural on the Amiga: typing dir >prt: opt a should do it. Try it — you won’t miss the old Commodore syntax for long.

As we go to press, there’s still no official word on the new Amiga models you mention, so it’s too early to say whether to expect compatibility problems. However, you can take heart from the fact that the Amiga’s operating system software is designed from the ground up with flexibility and expandability in mind — it is light years ahead of all other Commodore computers in this respect. In the old days at Commodore, ‘compatibility’ and ‘alphaalphabetabates’ were catchwords of about equal currency.

No niche for Amiga: Why are Amigas not selling? The short answer is that IBM/MS DOS is the business standard and the cheaper 8 bit computers are more than adequate for most home uses. Amiga would be great for desktop publishing, but the Mac already owns that territory.

Expandability? You’re selling a 1 Meg RAM box for $1035, nearly the price of a two drive clone. Anyone needing this much memory is probably running business programs and would be better off with DOS hardware, kludgy though it is. Ditto for Sidecar. To put it another way, why speak Esperanto then translate everything into English?

Multitasking might be a sleeper, but a lot of us have trouble enough concentrating on one task at a time. Granted, the idea of having a spreadsheet churning away while you write a letter to your MP sounds good on paper. But seriously folks, when’s the last time you had to recalculate the orbits of all the stars and planets? Most real world spreadsheet applications run in a few seconds, even on the lowly PC.
Hi-res Trace Utility SYS Address:

If you've had any slight difficulties with the trace utility on page 69 of Volume 7, Issue 4 - such as the program failing to work - you may wish to change the SYS address in line 1060 from 52096 to 52723. And always be on your guard for sophisticated DATAfiers that cavalierly assume that a program's start address is going to be the same as its entry point.

The trace utility also has a problem with version 2 C64 ROMs - the version that fills colour RAM with the background colour when you clear the screen. To make the program work properly on these machines, make the following changes:

- line 1850: delete the last data item (162)
- line 1890: delete the last data item (169)
- line 1900: delete the entries 147, 32, 210, 255
- Finally, create a new line: 1865 data 169, 147, 32, 210, 255, 162

In the assembler source code, the only change required is to reposition lines 4890 and 4900 to fall between lines 4690 and 4700.

Uncontrolled Sprite Bits:

David Johnson of Gander, Newfoundland, reports an error in the third example of the article 'Bit Addressing of Sprite Controls', from Volume 7, Issue 2. He suggests the following changes:

- Change line 1 to ldx *5, instead of lda *5
- Change line 5 to beq dabble, instead of beq maskor

Tape Verifizzle:

Apparently not too many of you are making use of the Verifizer for cassette users that we've been publishing for the last while, because we've just now found out from a single letter that up until this issue it had a fatal error in the data statement in line 1070. The last number in that line should be 247 instead of 249. Thanks for spotting the error goes to Phil Hoff of Chico, California.

Slashing and Pounding:

In his book, The Home Computer Wars, Michael Tomczyk reported that one of his biggest triumphs as a one-time Commodore marketing executive was the replacement of the standard ASCII backlash (code 92) with the English pound sign. This astounding innovation was first seen on the VIC 20, and was inherited by Commodore's subsequent 8-bit machines. Well, maybe Tomczyk had a point and maybe he didn't, but a side effect of the switch is that we have to be extra careful when we print an article with the pound sign in it. The typesetting machine doesn't know Commodore from coconuts, and code 92 comes out as a backlash every time; to get the pound sign we have to manually substitute a character from a special font. All of which explains why the 'Commodore 64 23K RAM Disk' article in Volume 7, Issue 5, used an incorrect backlash both in the text of the article and in line 180 of the program, instead of the correct British pound sign. Thanks to Jack R. Farrah of Cincinnati, Ohio, and James C. Sanders of Knoxville, Tennessee for pointing out the error.

TransBloopers

EPROM Burns:

Our thanks to everyone who sent in error reports for the Universal EPROM Programmer article in Volume 7, Issue 05. A follow up article appears in this issue.

Jim O'Hare, Victoria, British Columbia

It's a bit ironic that the Amiga, which is arguably the most powerful micro on the market today, should be widely seen as having no natural place. It is even more ironic that its most radical innovation - multitasking - should be so badly undervalued. I think it's a misconception to suppose - as your spreadsheet example implies you do - that there is no benefit to multitasking unless all tasks are continuously running full bore.

Perhaps in your letter to your MP you would like to include some juicy numbers about government waste. No problem - just save your document file from your word processor, exit the program, load your spreadsheet, make your calculation, write down the numbers, exit the spreadsheet, reload the word processor, reload the document file, and you're ready to go. Easy! Or, you can click the word processor out of the way, do your spreadsheet calculation, then go back into the word processor with another mouse click. Easier!

But that's just scratching the surface. The real value of the multitasking environment might not become apparent to you until after you've worked on the machine for a month or two. And that's when for some reason you have to go back to a single-tasking computer for a while. You'll find it frustrating and confining. By this time you've become used to slipping fluidly from one program to another and back again, with almost the same ease as your thoughts shift between topics, or your hands between tools. And you won't want to go back to a machine with a narrow-minded, serially-oriented operating system.

Part of Commodore's marketing problem is that multitasking is hard to promote. Reading about it isn't enough - you have to live with it to appreciate its true power. But time will take care of that, and a few years from now single-tasking machines may be just Mesozoic curiosities. That may or may not help the Amiga, since being too early can be just as bad as being too late, but it's not inconceivable that it could eventually be more successful than anyone now imagines.
Smile! You’re on RLE!

Christopher Dunn
Chicago, Illinois

RLE was first implemented on Compuserve to handle display of the National Weather Service radar weather maps.

But First, A Word From Our Sponsor...

Our regular TeleColumn does not appear this issue as usual. Instead, we have just a few notes here (sorry for the interruption Chris). TeleColumn will be back in the next Transactor, but part of what we intended for the TeleColumn that isn’t here was the command sequence for configuring the DataPac network to allow successful up/downloads on CompuServe. However, Ranjan Bose thoughtfully included this at the end of his online services comparison (see next article).

The Commodore “MagNet” is the tentative name for the online magazine section of the Commodore Network on CompuServe. So far there’s been mixed response over the choice of name, ranging from “terrific marketing idea”, to “catchy”, and on through “too cute”. So, being the democratic blokes that we are, the floor is currently open to suggestions. If you have one, or if you like “MagNet”, please let us know. You can call, write, or even message us on CompuServe (enter GO CBMNET and choose CBMCOM or CBMPRG).

Building the online magazine section for CBMNET is like learning a new language – and the first program you write is a database! However, progress is being made and articles from this issue will be among the first to appear! We’re also considering a sub-section of “Online Exclusives”. We often receive articles containing programs that are just too big to publish in the magazine. These would be ideal in an Online Exclusives section since “entering the program by hand” is no longer criteria for deeming an article unacceptable. We took out our box off unused stuff and found several we could already put to use for this idea. There’s only one problem – our author payments budget can’t handle the load of this extra material, but if a free T-shirt or some other gift would suffice, we’ll be happy to oblige.

Authors who submitted material that wound up in the unused box probably know who you are, and we’ll be getting in touch with you shortly to see how the idea might appeal to you. Naturally, you can feel free to decline and your material will be returned. Further, we’ve already returned many articles that might also have fit into this category, and there are undoubtedly others with articles that would merit a place in this “telespotlight”. So, if you’d like to re-submit them, they’ll be considered for Online Exclusives.

 Christopher and Pamela Dunn

Ok, what is RLE? It stands for Run Length Encoding, and is a system developed at CompuServe to enable them to transmit high resolution graphic images to different brands of computers.

A hires image is made up of pixels, as most of us know, that can be set to different colors to form the picture on our screen. The image can be generated in various ways, most often with a drawing program of some sort, or with a digitizer when converting a real world image. This image can then be stored to disk for recall later. Different computers use different formats for this storage, and even different programs on the same computer use incompatible formats for storing images. Just about the only common item was that all computers used pixels to form the screen image. RLE is a way of encoding the screen image so that the original picture may be viewed on most different types of computers.

In The Beginning...

RLE was first implemented on Compuserve to handle display of the National Weather Service radar weather maps. If you were lucky enough, back a few years ago, to have a terminal program that had the RLE data decoder built in, then you could directly view these hires maps on your screen. If you did not have RLE capability, all you saw was gibberish. As the format of RLE filtered down to various programmers, programs were written to decode the data. Some were built into terminal programs, some were meant to be used off line to decode captured data.
More RLE selections were added to Compuserve, Users Pictures, images of the FBI's 10 Most Wanted, and more recently, images of Missing Children. However, all of these images were being generated at Compuserve, users themselves wanted to translate their own works of art into RLE. Step in the wizards again, and soon programs popped up to transfer hires images on your computer into RLE files. These files could then be uploaded to a database for all to view. (assuming they had an RLE decoder of course.)

Specifications

The format of RLE is really quite simple. It had to be simple because it had to be understandable by a wide range of computers. First off, all RLE images are in monochrome. I won't say black and white, because you could display a RLE image in blue and yellow, or any 2 colors you like, but the RLE data only tells you if a pixel on your screen is on or off, not what color it is. The standard in RLE files is a Black background (Pixel off) and a white foreground (pixel on).

The screen size for an RLE image is 256 pixels across and 192 down. This was chosen because it was the resolution of the screen on the original Radio Shack Color Computer, which was widely in use when RLE was being developed. On the C64, the screen size is 320 by 200, so an RLE will not quite fill the screen. A border is left on the left, right and bottom.

If you look at a hires image on your screen, and starting in the upper left corner, counted the number of pixels that were of one color, then when you hit a pixel of a new color, start a new count, you would have an idea of what Run Length Encoding is. RLE is the count of dark and light pixels across a line on the screen. RLE data is sent in pairs of ASCII characters. The first translating into the number of dark pixels and the second translating into the number of light pixels to follow.

Image Coding/Decoding

Say that the first line of an image had 3 dark pixels, then a light one followed by 3 dark ones and another light one. The RLE encoding program would count the first 3 dark pixels, see that the next one was light, so generate a 3 for the first dark value. Then the program counts the light pixels, finds only one, and 1 becomes the light value. We now have our first pair; a 3 and a 1. This could be used as the RLE data, but a character with an ASCII value of 3 happens to be the Control-C character. Trying to send that out on most networks would cause havoc! So to the raw value an offset of 32 is added. 32 is an ASCII SPACE, so all RLE data is at least a space character or higher. The maximum character usable in ASCII is 127, the Delete character. 127 – 32 = 95 So the largest value for a string of pixels is 95. Getting back you our example, the raw pixel count is adjusted to produce the ASCII characters "*" and "!" (chr$35) and chr$33). The next pixels would then be counted from the image and the process continued.

When the end of a line is reached, you simply wrap around on the next line. If the pixels don't change state from the end of one line to the beginning of the next, you just keep counting. If you hit 95 pixels in a row though, you have to stop. Since the maximum ASCII character usable is 127, and that means 95 pixels in a row, what happens when there are more than 95? If the pixels count was of dark pixels, the first character of the pair, you would generate the character 127. For the light character that follows you would generate character 32 (ASCII for space). At the decoding end, the 127 would give you 95 dark pixels, and NO light pixels since 32 – 32 = 0. You then go on counting the dark pixels where you left off and generate a new pair of characters as needed. The same would hold true if the first pixel encountered in an image was light. The first character of the data pair, the one for the dark count, would be an ASCII 32, meaning NO dark pixels to start with.

You can now see the meaning of Run Length Encoding. You run down the length of the images counting pixels and generating the data from the count of dark and light pixels found. The RLE data is stored in a text file, and text files are easy to transmit via modem.

At the displaying end, everything runs in reverse, the first pair of data is taken and decoded into the number of dark and light pixels to draw on the screen, then the next pair is taken and so on until the bottom is reached.

The only addition to an RLE file I will mention is that it has a 3 byte header; the characters <ESC>GH. The <ESC> is ASCII 27. And the GH stands for Graphic Hires. This code is used to switch terminal programs into RLE decoding mode automatically. The end of a RLE file is marked with <ESC>GN for Graphic Normal. (Normal terminal mode.)

Commodore RLE

I was writing above in general terms because it applies to all computers, not only Commodore equipment. Now I will get into some specific programs for the C64 that create and display RLE files.

As RLE was meant to display pictures while connected on line, the RLE decoding software was added to terminal programs. You can also download or RAM buffer capture the RLE data and display it off line. Presently I know of 2 terminal programs that will display RLE images on line. Compuserve's Vidtext 4.2 and my own CBterm/c64. If you capture the RLE data to a disk file, you may display it off line with programs like RLE2HR. You may generate your own RLE files from your hires images with the program HR2RLE. CBterm/c64, RLE2HR, and HR2RLE may be found for downloading from the databases of the Compuserve CBIG Sig. Vidtext 4.2 is available for purchase from Compuserve. CBterm will also print the hires image if you have a Star or Epson type printer. Both CBterm and Vidtext may be made to save the hires image to disk in a Doodle or Koala compatible file with the use of an overlay subroutine for each program also available in the databases of CBIG Sig.

Note to Vidtext users: Before RLE pictures can be displayed, you must enable hires Memory. Enter Meta-O and select Option 8 HIRES MEM. Enter 'E' for Enable and hit Return twice to exit (Option 8 defaults to 'D' for Disabled).

RLE Image Sources

Now that you have read this article, you may be wondering what kinds of images are available to look at online? As I mentioned in the beginning, you may view the weather radar maps updated
hourly (GO WEATHER), or the FBI's 10 most wanted list (GO TEN) or pictures of missing children (GO MISSING). Other areas on Compuserve are adding RLE images all the time. But that is only the half of it. There are hundreds of pictures that have been generated by users and uploaded to various Forums on Compuserve. Database 3 of CBIG contains over 100 RLE images alone, ranging from very fine detailed drawings by some very skilled artists to some equally interesting computer room "Nudes".

The Florida Forum uses RLE files to display maps and images of attractions from that state. Then there is the Picture Support Forum (GO PICS) that was formed for the purpose of helping people with displaying and generating RLE files. It has a vast database of user contributed images.

In DL12 of The Commodore Arts and Games Forum (GO CBMART) you'll find a program called CAD30.BIN. Also in DL12 are some 40 RLE Figure files that go with this CAD package. Since this is in fact the "Arts" forum for Commodore users, their DLs will contain other interesting RLEs, and probably many more to come.

An RLE file is highlighted in a forum's database by the addition of "/TYPE:RLE" printed after the filename. If you Read that file and are running a terminal program that will do RLE, then the image will display on your screen. When you go to upload your own RLE file to a database, you should also add "/TYPE:RLE" to the upload command so Compuserve will mark the file as an RLE image.

New files are being added to the forums all the time. Ask the forum's SYSOP if there is a special place for RLE images. Since RLE is a universal format, your Commodore generated image can be displayed on lowly IBMs and Apples. Its about time those folks can see what a real computer can do...

If you have any questions I might be able to help with, leave them to me at the CBIG sig (GO CBIG, leave a message to SYSOP) or Eazyplex me at 76703,717. I will help if I can. There are also a number of graphics utilities for conversion between various hires formats available in CBIG's databases 2 and 5. The RLE images are in database 3.

Editor's Note: As a bonus, the programs "RLE2HR" and "HR2RLE" will be on The Transactor Disk that goes with this issue (Disk #17).

From FBI Ten Most Wanted Section (GO FBI)

Fillmore Ray Cross, Jr.
AEG: 44
Born: Aug. 11, 1942
San Fran., CA
Height 6'
Height: 175 lbs.
Build: Slim-Red.
Eye: Hazel
Hair: Brown.
Complexion: Fair.
Nationality: Amer.
Tattoo: Snake & Rose, upper rt. arm.
Snake & panther, upper rt. arm, muscular arms & forearms.

From The Missing Childrens Section (GO MISSING)

Christopher
Enoch Abeyta
Age: 12 months.
Born Nov. 28, 1985
Eye: Blue.
Height: 12.5".
Hair: Black.
Weight: 22 lbs.
Hair: Blonde

The Transactor 18
May 1987: Volume 7, Issue 06
Telecommunication is a fascinating and extremely useful application of computers. Few of those who have tried it ever get over the sheer magic of the ability to ‘talk’ with remote computers that are usually so much bigger and powerful than their own. Commodore equipment owners have generally been fortunate in this respect because CBM has always been actively involved in promoting this branch of computing among their patrons. They have introduced four modems so far, each a step better than the last. Their first modem, popularly dubbed the ‘Vicmodem’, which was introduced with the VIC-20, was a very basic piece of hardware. Next came the 1650 automodem, soon to be followed by the 1660 with touchtone dialing, then finally, the 1670, which not only made 1200 bps access available, but also many other convenient features including Hayes-like ‘smart’ commands. The modem comes packaged with several telecommunication programs of high quality, and all this without one having to trade in that ticket to Hawaii.

**Beyond BBSs**

A typical telecommunications neophyte usually teethes on bulletin board systems, which often vary a lot in features and capabilities. The real fun and power of telecommunicating, however, extends beyond reading E-mail or messages or even exchanging programs. In the present age of ‘Infomania’, one needs to know a lot of varied bits and pieces of information. These billions and billions of characters can be handled efficiently only by a mainframe computer, in fact, several of them working together. Several such huge information databases exist, providing the following categories of services:

a) Pure information – Encyclopedias, research libraries, news items, stock prices, etc.
b) Public-domain software, freeware and shareware.
c) Electronic mail (E-Mail)
d) Interactive games involving single or more players.
e) Special interest groups (SIGs) or clubs catering to a wide variety of interests from Astronomy to Zen.
f) Real time interactive conferencing (electronic CB).
g) Commercial services – Banking, stock investments, books, software, travel arrangements, etc.

Some of these services are provided through ‘gateways’ to other mainframes and often incur additional surcharges.

One big difference between bulletin boards and information services is that one is charged by the minute when accessing the latter. For most people living outside the continental U.S.A., there are additional telecommunication charges. Since one may be paying from 17 to 75 Canadian dollars per hour, one really has to plan an online session ahead of time, and familiarity with the system helps.

Three of the many services which support the interests of Commodore users are Compuserve (CIS), Delphi (DEL) and Quantum Link (QL). All three permit access up to at least 1200 bps, with Compuserve having differential rates for 300, 450 and 1200 bps access. Of these, Quantum link caters exclusively to owners of the C-64 and C-128.

**Software Requirements**

Quantum link requires special telecommunications software which is distributed free with several soft/hardware packages (e.g. GEOS, 1670, etc.). A monthly minimum service charge of $9.95 (US) is levied, which includes one free hour of special services (almost anything of interest including Email!) costing $3.60/hour (US). Compuserve recommends its ‘Videx’ software system which supports its proprietary error-checking “B” protocol, a protocol that is more efficient and faster than Xmodem, high resolution graphic displays (RLE), and other special and standard features. Several public-domain programs also support some or all of these features. In addition to the ‘B’ and Xmodem protocols, CIS also supports file transfers using Xon/Xoff, and special software is not absolutely essential. Delphi permits Xmodem transfers and is compatible with most software. There are no minimum monthly charges for regular members of CIS and DEL.

**Access**

QL is the easiest to access because its special software automatically dials and connects with the system. The software supports practically all popular modems. Accessing CIS or DEL is a bit more involved depending on whether you are connecting directly, or through a telecommunication network such as Tymnet, Telenet or Datapac. Some terminal programs support user-definable keys and autostart features which, once programmed, make the log-on procedure easy. CIS and DEL identify a user account by a user id and password; the latter should be changed periodically to protect one’s account from unauthorized access. QL handles account identification automatically.
Features

Although all three services provide all of the basic features, the range of services on CIS is the largest. QL will be of greater interest to beginners because everything is menu-driven, and the menu-screens are colourful. The system, however, is not flexible, and advanced users cannot but help feel caged and slowed down. CIS and DEL permit menus for beginners, and short prompts or command mode for advanced users. These save a lot of time ($5) in the long run.

The quality of games on QL is much better, because of the interactive, colourful high-resolution graphic displays, when compared with the primarily text-oriented games available on the other two services.

On the other hand, both CIS and DEL provide access to research databases such as DIALOG, MEDLINE, etc. Both of these services also offer searches made on one's behalf by experienced librarians ($$).

Electronic mail is easier and more flexible on CIS than on DEL, which, however, provides services such as Telex, Globalink, etc. Both permit forwarding of mail to other information services.

QL is entirely Commodore-specific and is currently supported by CBM. There is a GEOS SIG, Commodore hotline, product reviews, AHoy and RUN forums and a unique program preview section from where one can download good quality demo versions of commercial software and their abridged documentation. Several public domain program libraries are spread all over the system. The Commodore SIGs on CIS, which previously were supported by CBM, still continues to support programmers, telecommunications, graphic and sound artists, and provides technical information. It is now managed by the staff from The Transactor and others. Several data libraries chock-full of public-domain programs exist on the system. CIS SIGs also support most other computers and electronic editions of several popular magazines. DEL is the poorest in this respect. It has a ‘Flagship Commodore’ forum manned by ex-SYSOPs from CIS, and has limited fare for Commodore enthusiasts.

All three services have sections for placing and reading Classified ads, which are free on CIS and QL. CIS and DEL assign personal disk space to users for storing their files and other information.

Navigating

CIS is by far the most flexible system, with the easiest and fastest navigating commands, permitting one to jump around the system. One can even custom-design the first menu to include only areas of primary interest. Excellent manuals, charts and the famous book by Bowen and Peyton, “How to Get The Most Out of Compuserve” (Bantam) are recommended reading. One should try to gain familiarity quickly and switch to Command mode as soon as possible for increased speed, thereby saving time and money. Navigating commands on CIS are logical and consistent between similar sections. In addition, CIS transfers information appreciably faster than the other two, which are prone to delays even during non-rush hours. QL can be painfully slow at 300 bps and has a very slow SEARCH operation. This almost wipes out the differences in fee structure, and CIS comes out ahead of the other two. Moving files between your personal disk area and public areas, or sending them as E-mail is much more advanced yet easy on CIS, which provides a choice of two different editors with distinctive features (actually a third kind of editor is also available for special use). Two special commands on CIS, namely GO and FIND, are really helpful in moving around the system efficiently. All three provide feedback services for answering specific questions and problems. CIS’ feedback service is more human, and reads least like a computer-generated form letter!

In summary, for Commodore equipment owners, both CIS and QL offer a lot of useful services and information. QL is somewhat limited in features but extremely easy to use, with some unique and interesting features such as commercial software demos, GEOS forum, educational resource center and fantastic games. Compuserve, on the other hand, has a huge, dynamic spectrum of features, easy, flexible and fast navigation, and a lot more to offer, especially after one gains some familiarity. Delphi, in my opinion, hardly offers anything uniquely attractive to patrons of Commodore equipment at this time.

Special note for CIS users accessing via DATAPAC:

For years now, Canadian users, accessing CIS via DATAPAC, were not able to transfer programs or other files using ‘B’ or Xmodem protocols. The problem is due to DATAPAC’s aggressive interference which blocks transmission and hangs up the systems involved. The following procedure can, however, soothe DATAPAC into letting such a transfer (B or Xmodem) proceed. This routine was worked out by some user and was provided to me by Compuserve. I have used it frequently with 100% success:

1. After dialing DATAPAC and entering one or two periods (300 or 1200 bps) and receiving the DATAPAC node address message, enter P 29400138 <Return>

2. At the Compuserve user id prompt, enter: tPPAR <RETURN> Control-PPAR

You will next see a large list of numbers following “PAR”.

3. Enter: PROF 1 <Return>

This causes a global change and you will lose echo from DATAPAC. Type the following blindly, or use predefined macro keys, or switch into half-duplex if your modem has a switch to do so.

4. Enter: SET 126:004:003:000:004:004:001:000 <Return>

5. Exit DATAPAC by entering a <Carriage Return> followed by: GOODBYE <Return>

6. The Compuserve user id prompt reappears. Type on as usual.
Commodore in Europe:
An International Comparison of Price and Availability
Mikos Garamszeghy, Toronto, Ontario

(VIENNA, Austria: Transactor International News Service)

Vienna is one of my favourite cities: home of music, art, cafes, pastries, chocolates and Commodore computers. Commodore computers? On a recent business trip to Vienna, I had a chance to do some window shopping for computer equipment. This article relates some of my experiences in Vienna and other European countries.

Commodore machines are quite popular in most parts of Europe, even in certain Eastern Bloc countries such as Hungary. (A Hungarian colleague who works in a large engineering and scientific research institute tells me that most of their word processing and scientific computing is done with C-64 and C-128 computers). Computer equipment is generally available in mass market retail shops such as department stores (even Harrod’s in London, where the Queen and her family shops, sells Commodore equipment) and specialty stores such as photo/electronics stores and computer stores. Most serious software is sold at the specialty stores, while the mass market stores generally only sell games programs. Technical documentation is plentiful in the German speaking countries (most of the Abacus book series is translated from the original German version for use in North America). There are also some very good technical magazines. The German version of RUN magazine rivals TRANSACTOR for its no-nonsense technical content (all in German, of course). Several magazines are also available in English and French dealing with general and Commodore specific computing. Sadly, I hear, the French version of RUN is to cease publication at the end of 1986, leaving that country without a native Commodore specific magazine. Perhaps Transactor should fill the gap with a French version?

Much of the hardware familiar to North Americans is available in most European countries. The C-128, C-64 (both old and new versions) and AMIGA are generally available along with their usual peripherals. In addition, the IBM clone PC-10 and PC-20 and even an AT clone (PC-40) are also available. Several machines that have long been dropped in North America, such as the Plus/4 and C-16 (as well as an enhanced model, the C-116) are all popular items. Third party hardware and software support for the Plus/4 – C-16 type machines is fairly strong. Memory expansion cartridges and RS-232 ports are even available for the C-16.

I saw several items that I would like to see in stores on this side of the pond, but Commodore does not seem to want to bring them over. The most interesting is the 128-D. This is a two piece C-128 compatible unit with a detachable keyboard, a built in 1571 disk drive and, what every Commodore owner dreams about, a built in fan cooled power supply. The main computer and disk drive unit is styled much like the Amiga and is the same color as a regular C-128. The detachable keyboard is a very nice feature. It has the same number of keys and layout as the conventional C-128 board with perhaps a slightly better tactile feedback. The keyboard cable is about 1/2 inch in diameter and terminates at the main computer unit in a RS-232 or IEEE type “D” plug. Although the cable may be a bit short and stiff for some people who like keyboards on the lap, I found it quite comfortable to work with. Judging by the diameter of the cable and the number of pins in the plug, I would say that it supported all of the connections used on the internal plug of a normal C-128 keyboard cable. Therefore, it should be possible to connect this detachable keyboard to a North American style C-128 (if you could get your hands on one of these keyboards). The main computer/disk drive unit is about 430 mm (17 in) wide x 405 mm (16 in) deep x 100 mm (4 in) high and weighs about 9.3 kg (20.5 lb). It also has a fold down carrying handle on the left side. Although it does not have a built-in monitor like the SX-64, it might still be considered to be a semi-portable.

The European C-128s and 128-Ds have two character sets in ROM. They are selected by the North American “CAPS LOCK” key which becomes the ASCII/DIN key. The first set (ASCII) is identical to the North American set. The second set, DIN (abbreviation of the German national standards board), includes a few extra characters and accents not found in the English language. In appearance, the DIN set resembles the characters used in the older Commodore PET series and has a much crisper appearance. The lines in the characters of the standard C-128 are two pixels wide each. The DIN characters have one pixel wide lines. This gives it the crisp image on a good quality monitor. It is also suitable for European TV standards which have a higher resolution than North American NTSC.

The 128-D supports all three of the C-128 operating modes and had no trouble running any of my C-128 and CP/M software. (But then again, I would be surprised if there were any incompatibilities because it is essentially the same machine in a different box.) The 128-D is available in Austria, Germany, England and most other European countries. It is also available in Australia.

The Transactor  21  May 1987: Volume 7, Issue 06
The Europeans also have a second C-128 compatible disk drive to choose from: the 1570. This drive resembles a 1541 in outward appearance but is actually a single sided version of the 1571. It supports the fast serial bus, burst mode as well as single sided MFM type CP/M formats. The price is about midway between a 1541 and a 1571.

The special 1551 parallel drive for the Plus/4 and C-16 (a very hard to come by item in North America) is relatively easy to find in most parts of Europe. Several third party manufacturers offer 1541 compatible drives and other mass storage devices, such as a 200k wafer tape drive that works on the cassette port. Speaking of cassettes, most casual European users seem to prefer tape units over disk drives. Judging by the relatively high cost of the equipment, I am not surprised!

There were several fairly “popular in North America” items that I did not see in all my wanderings around Europe. Perhaps the most noticeable was the 1700 and 1750 RAM expansion cartridges, although several third party RAM expanders were available for the C-64. Modems were also very scarce and expensive.

Table 1 is a summary of some of the more common computer prices in selected countries. All prices are shown in Canadian dollars, converted from the appropriate national currency at the prevailing exchange rate. It should be noted that most of the European prices include all taxes (up to 30% or more, which in most cases are refundable to non-resident purchasers upon leaving the country) while North American prices generally do not. Europeans also tend to sell computers as part of a package with a monitor and a tape or disk drive, sometimes even with a printer, usually with bundled software. The prices in the table have been separated to the extent possible into the component items. Some items in the list may not be familiar to North American readers such as the Sinclair and Amstrad computers. These are both quite popular in most parts of Europe and are aimed at a similar type of market. They are included to give an indication of the price of Commodore equipment relative to similar equipment. In addition, a few of the Commodore product numbers mean different things in different countries. For example, in North America a 1901 monitor is a monochrome monitor while in Austria and Germany it is a color monitor equivalent to the North American 1902.

**Commodore 128 D**
<table>
<thead>
<tr>
<th></th>
<th>Canada</th>
<th>U.S.</th>
<th>Austria</th>
<th>W. Germany</th>
<th>U.K.</th>
<th>Hungary</th>
<th>France</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commodore: (Computers)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-16</td>
<td>99</td>
<td></td>
<td>199</td>
<td>161</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+4</td>
<td>199</td>
<td>120</td>
<td>298</td>
<td>209</td>
<td>250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-64 (old)</td>
<td>249</td>
<td>199</td>
<td>298</td>
<td>321</td>
<td>298</td>
<td>735</td>
<td>329</td>
</tr>
<tr>
<td>C-64 (new)</td>
<td>299</td>
<td>249</td>
<td>398</td>
<td>349</td>
<td>390</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-128</td>
<td>450</td>
<td>385</td>
<td>599</td>
<td>475</td>
<td>538</td>
<td>1911</td>
<td>599</td>
</tr>
<tr>
<td>128-D</td>
<td></td>
<td></td>
<td>1198</td>
<td>979</td>
<td>998</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amiga</td>
<td>1995</td>
<td>1500</td>
<td>2300</td>
<td>2250</td>
<td>1950</td>
<td></td>
<td>2800</td>
</tr>
<tr>
<td>PC-10 II</td>
<td>1800</td>
<td></td>
<td></td>
<td>2300</td>
<td></td>
<td></td>
<td>2785</td>
</tr>
<tr>
<td>PC-20 II</td>
<td>2500</td>
<td></td>
<td></td>
<td>3200</td>
<td></td>
<td></td>
<td>4289</td>
</tr>
<tr>
<td><strong>(Disk drives)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1530 Datassette</td>
<td>49</td>
<td>56</td>
<td>79</td>
<td>65</td>
<td>65</td>
<td>225</td>
<td>75</td>
</tr>
<tr>
<td>1541</td>
<td>349</td>
<td>210</td>
<td>399</td>
<td>360</td>
<td>398</td>
<td>882</td>
<td>440</td>
</tr>
<tr>
<td>1551</td>
<td></td>
<td></td>
<td>459</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1570</td>
<td>499</td>
<td></td>
<td>499</td>
<td>425</td>
<td>450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1571</td>
<td>425</td>
<td>350</td>
<td>599</td>
<td>509</td>
<td>518</td>
<td></td>
<td>650</td>
</tr>
<tr>
<td>SFD-1001</td>
<td>399</td>
<td></td>
<td>399</td>
<td>400</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>(Monitors)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1702</td>
<td>399</td>
<td></td>
<td>499</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1801</td>
<td></td>
<td>295</td>
<td>599</td>
<td>390</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1901</td>
<td></td>
<td></td>
<td>799</td>
<td>650</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1902</td>
<td>549</td>
<td>399</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>(Printers)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS 1000</td>
<td>499</td>
<td>350</td>
<td>699</td>
<td></td>
<td>518</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1525</td>
<td>199</td>
<td></td>
<td></td>
<td>279</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS 803</td>
<td>499</td>
<td></td>
<td>279</td>
<td>298</td>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1520</td>
<td>399</td>
<td></td>
<td>175</td>
<td></td>
<td>175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS 802/1526</td>
<td>399</td>
<td></td>
<td></td>
<td></td>
<td>700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPS 1101</td>
<td>499</td>
<td></td>
<td></td>
<td></td>
<td>550</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Atari:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800 XL</td>
<td>159</td>
<td></td>
<td>199</td>
<td></td>
<td>140</td>
<td></td>
<td>320</td>
</tr>
<tr>
<td>130 XE</td>
<td>249</td>
<td>199</td>
<td>399</td>
<td>210</td>
<td></td>
<td></td>
<td>1290</td>
</tr>
<tr>
<td>520 ST (mono)</td>
<td>1099</td>
<td>910</td>
<td>1499</td>
<td></td>
<td>1190</td>
<td></td>
<td>2580</td>
</tr>
<tr>
<td>1040 ST (color)</td>
<td>1699</td>
<td>1599</td>
<td>2399</td>
<td></td>
<td>1800</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sinclair:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL</td>
<td>599</td>
<td></td>
<td>398</td>
<td></td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectrum</td>
<td>249</td>
<td></td>
<td>169</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectrum +</td>
<td>299</td>
<td></td>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectrum 128k</td>
<td>280</td>
<td></td>
<td>280</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectrum 128k + 2</td>
<td>320</td>
<td></td>
<td>429</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Amstrad:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>464 monochrome,tape</td>
<td>649</td>
<td></td>
<td>400</td>
<td></td>
<td>579</td>
<td></td>
<td></td>
</tr>
<tr>
<td>664 color,3.5 disk</td>
<td>1199</td>
<td></td>
<td>800</td>
<td></td>
<td>1289</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6128 color,3.5 disk</td>
<td>1399</td>
<td></td>
<td>800</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1) All prices are quoted in Canadian Dollars based on the following exchange rates:
$1.00 US = $1.40 CDN; 1 AS = $0.10 CDN; 1 DM = $0.70 CDN; 1 UK = $2.00 CDN; 1 forint = $0.029 CDN; 1 FF = $0.215 CDN
2) Prices based on average of several typical retailers in each country; European prices include applicable VAT (sales tax).
Provoking Thought

The thing that keeps me plugging away at this keyboard, besides a profound aversion to real work, is the deep-seated belief that what lies before me is not a tangle of inanimate circuitry, but a new order of life evolving, and that some day it will thank me for the faith I had in it and the help I gave it way back in its infancy as a species.

I had a conversation with a woman at a party once who appeared to believe the same, but as it turned out was only trying to shed herself of an incredible nerd without getting into an argument. Still, it’s the closest I’ve come to finding someone with my “religion.”

Life . . .

The tired phrase “user friendly” turns my stomach as quickly as the next; still, it does illustrate the significance of the living element in software. Good programs do not give users the feeling that they are dealing with something inanimate.

And Recursion

I would like to discuss a glorious technique for breathing life into your programs. It is standard fare in many languages; unfortunately, BASIC is not one of them. Therefore, many Commodore hackers quite possibly remain oblivious to its power and even its existence. I refer to recursion, likened by some to a snake swallowing its tail.

What Is Recursion?

“Recursion is recursion is recursion.

(recursive definition of a recursion)"

Actually, if you look up “recursion” in the dictionary, you will see it defined as,

“…common coding mistake of novice and student programmers when the use of GOTO has been forbidden. See stack overflow.”

Recursion involves either a subroutine calling itself, or two or more routines calling each other. To clarify, imagine THIS routine calls THAT routine, and then THAT routine calls THIS routine, and then THIS routine calls THAT routine, and on THIS routine has returned from its original call to THAT routine in the first place, and so on, and so on but not indefinitely, of course. That would be recursion. The opposite of recursion is iteration, in case you ever need to impress someone at a party.

Where Can I Get Some?

Languages like C and PASCAL support recursive activity very nicely. Languages like BASIC and COBOL (ick) and RPG (double ick) do not. Assembler supports everything; it just takes a little more work.

For a language to recurse properly it must support local variables, which means that every function, procedure, subroutine or whatever, must set up its own personal variable space each time it is called. These variables cannot be chewed on by the rest of the program or even by other calls to the same routine unless you, the programmer, expressly say so.

Mid-Term Test

What would happen if you ran the following Basic program, then typed “12345q”?

80 gosub 100
90 end
100 get k$: if k$ = “” then 100
110 if k$ <> “q” then gosub 100
120 print k$;
130 return

(A) “RETURN WITHOUT GOSUB?” error message
(B) The system would crash
(C) “q” would be printed
(D) “qqqqqq” would be printed
(E) “q54321” would be printed

The correct answer is D, not only because previous values for K$ are lost each time you press a key and Basic would RETURN
through the PRINT once for each time you pressed a key, but because the correct answer is almost always D on multiple choice questions (same as my mark).

If you pressed too many keys before the Q, you would blow up the stack by nesting GOSUBs too deeply.

If a similar program were written in Pascal then E would be the correct answer. That again would be recursion.

So What's It Good For?

Recursion is useful for a lot of things besides printing input backwards; nested expression evaluations, binary searches and quick sorts, just to name a few. One of the most traditional applications of recursive algorithms is in analyzing board game positions, where a program must check every possible line leading from a given position for a specified number of moves before choosing the best.

The two Commodore 64 programs included with this article are the board games REVERS1 and my very own personal version of this game, FENCE. The following short routine provides the brains for each of them, and indeed could be used in any situation in which one wanted a program to "look" down a decision tree.

Don't bother typing it in and trying to run it unless you're in love with your assembler's UNDEFINED SYMBOL error message. See if you can follow in a general way what is happening, so that when the need arises you'll be ready.

```assembly
find'best'move = *
 lda #0
 sta level ;initialize level to present position
 jsr setpointers ;to variable arrays for current level

checklevel = *
 ldx level ;current depth of analysis
 cpx difficulty ;how deep to go
 beq return ;if equal dont go any deeper

lda #0
 sta movemove,x ;# of moves tested on this level
 jsr findmoves ;returns # of moves and array of moves

check'next'move = *
 ldx level ;to access table values for this level
 lda movenum,x ;number of moves to look at
 cmp movemove,x ;see if all moves are accounted for
 beq return ;if so return

inc movemove,x ;otherwise bump counter
 jsr makemove ;change the board array
```

The above passage admittedly smacks more of pseudo-code than of a useful subroutine that one can plug in, black box fashion, to a program. It is not something a beginner may want to cut teeth on. It does display a fairly powerful and concise machine language method for traversing trees and, hopefully, will provide food for thought for programmers who've never considered the possibilities of controlled recursion in their programs.

Notice how X is used to access non-array variables relating to the level of recursion. SETPOINTERS sets up the table data for level so that recursion is completely supported.

Over-the-board possibilities are structured like a decision tree. Each node represents a position with the root being the current position. The branches off each node represent possible moves. The above algorithm zips to the bottom of the tree then works its way back up, exploring every possible combination of moves. Only the final (leaf) positions are actually evaluated with results compared and passed back up the tree via the PICK routine.

When this routine finally exits, the outcome of every possible line to the specified depth has been evaluated. The computer then simply chooses the best (or least worst) alternative. If computers were fast enough or if the universe was going to be around longer every game could be analyzed right to the end; the machine would never lose.

The Fun Side Of All This: Reversi

Reversi is a good game except that it's a pain to play over the board. A single move can entail flipping 20 or 30 pieces over, and klutzes like me tend to knock them all over the place, losing the position.
Reversi is a democratic game. All pieces are equal. Reversi is very positional and strategic, but not terribly tactical until late in the game.

The rules are simple: lay down a man so that you out-flank your opponent along rows, columns or diagonals. If there are no out-flanking moves then your opponent moves again. The game is over when no one can move. The winner controls the most squares.

If you don't quite understand yet don't worry. Just LOAD "REVERSI",8 and RUN it. Help is available.

If you want to play Flip (the computer) select 1 player. Selecting 2 players tells the program to monitor and referee a game between two humans.

Select a low level of difficulty when prompted, say 1 or 2, unless you really want to get creamed. INFINITE level will analyze ever deeper until a key is pressed (you won't live long enough to crash the stack). Anything over level 4 may take a few minutes, especially in the middle game. If you choose to play level 9 your grandchildren may have to finish the game for you.

A heart beats in the upper-right corner of the screen while the computer thinks so that you don't have to wonder if it has passed away or something. Flip will make a tweedle noise after moving, in case you fell asleep or were doing something else.

When you choose your colour, keep in mind that in REVERSI, black moves first.

**Get Help**

If you aren't sure what your moves are, press H for help and they will be shown to you. Use the CRSR keys (or joystick in port 2) to position the big X over the square you want, and press RETURN (or fire). The appropriate pieces will be turned and then the computer will play its move.

There are buzzers and horn and poignant messages for stuck positions and attempts at illegal moves. If you beat the computer. . . well, see for yourself.

**No Cheating**

There are keys for removing and adding pieces of either colour and for trading sides (B,W,E,T). These are for problem composition and special opening positions only -- not for cheating. No Cheating!!!

**Instant Replay**

When the game is over, you will be asked if you want to replay, start over or quit. Games are recorded in memory. Replay allows you to single step, using the space bar through a game just completed. By pressing P you can re-enter the game at any point and play through those "if only I hadda . . ." situations.

Full screen colour graphics are used for the board.

**The Game Of Fence**

If you like Reversi, you will love FENCE. Even if you don't like Reversi you may get off on it. Fence is like Reversi except that instead of being played on an 8 by 8 square checker type board, it is played on a larger 11 by 11 board, and only on the diagonals.

LOAD "FENCE",8 then RUN

**I prefer FENCE:**

It is possible to see farther ahead into a position because there are fewer legal moves – unfortunately, the same is true for the computer (Gnash).

The game is shorter and much more tactical than Reversi. It is possible (though not necessarily advisable) for the player moving first to force a stuck position on his/her/its opponent after only a few moves. The all-powerful corner squares no longer exist, but are replaced by 22 irreversible side squares which come into play early in most games. It is not unusual for a game of FENCE to end long before the board is filled.

Score board, features, commands, graphics, whistles, bells and buzzers are the same as with the Reversi game. The only difference is that if you beat the computer you get . . . well, see for yourself.

Where a number of moves have equal value, the computer will randomly select one, so memorizing a winning line (when you get lucky) is probably not going to help you the next time.

Unless you find extreme tedium therapeutic, it is unlikely that you would want to type in the thousands of data lines that these programs would generate even if Transactor were to see fit to print them. If you are interested in checking FENCE and REVERSI out this might be a good time to go for the Transactor disk.
Random Number Generation
In Machine Language

Gregory D. Knox
Wurtsmith AFB, Michigan

...some applications are more critical than others, but the point here is that you sometimes need sequences of random numbers that are well behaved (in a random sort of way)...

A variety of applications require generating a sequence of random numbers. Many games, for example, use a randomly generated number to simulate the rolling of dice or to change some feature of the program's operation in an unpredictable way. If you are doing any kind of simulation with a computer, you are almost sure to need these numbers in one way or another. Maybe you need to simulate some kind of input to a modeled system but only have a statistical description of the input "signal", or perhaps you're working with the simulation technique called Monte Carlo analysis. These applications and more can all make stringent demands on the sequence of random numbers you use. Of course some applications are more critical than others, but the point here is that you sometimes need sequences of random numbers that are well behaved (in a random sort of way). This article describes a technique to produce long sequences of random numbers.

I've been using the term "random" here without much regard for its rigorous meaning. In reality, the sequences we can generate with a computer are properly called pseudo-random sequences. The computer is a deterministic device and therefore produces deterministic results. Truly random numbers would have to be generated using some kind of nondeterministic process, for example, sampling the electrical noise voltage across a diode or resistor. Of course you could use such numbers in your computer, and there are instances where you might need to, but in general there are good reasons for not doing so. First of all, it's inconvenient, but beyond that you sometimes want random-like data but need to use the same sequence more than once. If it's a very long sequence, storing it could be a pretty big waste of memory. Another reason is that you can exercise a degree of control over the statistical properties of sequences you generate in a deterministic fashion.

Most high-level languages have some type of instruction that lets you generate sequences of pseudo-random (hereafter called random) numbers. Depending on the particular language and its implementation, these random numbers may or may not be very "good ones". There are several factors we need to consider when talking about "goodness". Two of the more important ones are: Any sequence we generate with an algorithm will eventually repeat itself. This is an unavoidable property and of some importance. Secondly, the relative frequency of occurrence for each number is not generally the same.

In almost all cases, we would like the number of elements contained in a random sequence to be very large before the sequence begins to repeat itself. There are methods of producing random numbers that result in the repetition of the sequence after only a very few numbers have been generated. At the other end of the spectrum, there are methods that yield sequences that are non-repeating for lengths of truly astronomical magnitude. The random number generating facilities provided with most high-level languages are not always as sterling in this regard as you might expect, though some are quite good.

Usually when you generate a sequence of random numbers, you want them to be uniformly distributed. That is to say that there should be the same frequency of occurrence for all numbers. There are lots of applications that require some other distribution, but the starting point for the generation of these non-uniform sequences is often the uniform distribution. A number of other more involved considerations also exist where the evaluation of random sequences are concerned, but we don't really need to deal with them here.

Rolling Your Own

When programming in assembly language, you often don't have access to a routine that lets you generate random numbers. Nevertheless, as we've seen, such a need may well arise. You may be able to use the Basic machine language subroutine contained in ROM, but this isn't always possible and might not
be just what you need anyway. There is a way to produce, in machine code, random sequences that are non-repeating for astronomically long intervals. In addition, these sequences are as uniformly distributed as is possible to obtain. Although we haven’t examined other figures of merit for random sequences, it turns out that these particular sequences possess many of those other desirable properties as well. The random sequences we’ll look at here are called Linear Maximal Length Shift Register Sequences. This sounds pretty intimidating, but using these sequences is actually quite easy.

Before we take a look at the technique, let’s digress and talk about a little hardware for a minute. As the name suggests, one of the earlier implementations of these things involved digital shift registers. The shift registers were connected so as to take the value of the output and one or more other shift register cells, add them together MOD 2, and feed the result back to the input or first shift register cell, then initiate another shift operation.

![Figure 1](image)

Figure 1 shows what this looks like for a shift register with five cells. Note the bit positions, or cells, where the taps must be applied to obtain the values that get fed back to the input cell. These connections weren’t chosen randomly. This particular arrangement of taps for a five cell shift register gives us the maximum length sequence we are looking for. Some other feedback arrangement would certainly generate an output sequence, but it could be a very short one. Consequently, you really don’t want to consider any but the particular connection of taps that produces the maximum length sequence. Using such a connection, you would get an output stream of 1s and 0s that appeared very random indeed. If your shift register had, say, 30 cells, the sequence would contain (2^30)-1 numbers before it started to repeat. If you were generating these numbers at the rate of a thousand per second, very easy to do, the sequence would not repeat for almost 300 hours.

The problem of how to select the proper taps still needs to be addressed. You can build a shift register with any number of cells you want, registers with two cells or a thousand are equally possible. Each has some configuration of taps that makes it a Linear Maximal Length Shift Register. How do you know what taps to use? There’s an involved mathematical technique that will give them to you but that’s pretty messy. I look up the required connection in a table that lists the tap positions required for a shift register of given length. Figure 2 is such a table for shift register lengths up to 34 cells.

Shift register sequences of this sort have many and varied uses in the communication electronics field as well as in other areas. We can use the basic theory ourselves to generate random numbers using machine language. The hardware technique described above used binary shift registers to store the 1s and 0s. Each cell of such a shift register could contain but a single bit and, of course, that is the reason the shift register output consists of a serial bit stream only one bit wide. The theory that applies to these binary shift registers, however, can be more generally stated. Although it seems to have been given little attention, you don’t have to restrict yourself to using binary shift registers. You could design a shift register with cells that can take on any of a number of discrete values, for instance, any integer between 0 and 9.

It turns out that all the theory applicable to the binary version of a Linear Maximal Length Shift Register also applies to these non-binary versions. As always, there is a proviso: with the binary shift register, we did our addition MOD 2; with any other base (B) we must do our addition MOD B. For example, if your shift register works with the integers from 0 to 9 (that’s base 10), then your addition must be done MOD 10. One other thing: the base you select must be smaller than the number of cells in your shift register or the sequence that results won’t be one of maximum length. For example, if you have an eight cell shift register, you have to operate it base 7 (integers from 0 to 6) or less.

**Doing it in Software**

Now that we’ve looked a little at this hardware stuff what can we do with it?

Remember how long the 30 cell binary shift register produced 1s and 0s before repeating. If you take this same 30 cell shift register, but operate it base 10, say, instead of base 2, the output stream will consist of integers in the range of 0 to 9. If you generate this sequence at the same rate as the binary sequence we worked with earlier, it won’t repeat for 36,533,877 years. That’s right, millions of years.
Of course we’re much more interested in producing random numbers with software. It is very easy to program the computer to mimic the operation of a shift register. Each of the cells of the shift register is represented by a memory location. Since each memory location is an eight bit byte, each cell of the shift register could hold any of 256 values, a number far larger than you need for this technique. What you do is decide on the number of cells you want to use then set up that many memory locations in RAM. Before you can use the shift register you have to fill each of the cells with a number. Remember that this number has to be less than the modulus B you’ve decided on (from 0 to 9, for example, if it’s MOD 10). Any combination can be used except all the cells can’t be zero or no output will result. Next, consult Figure 2 to determine what taps are required.

To make this software shift register actually shift, you perform an addition MOD B on the values in the cell locations indicated as taps in Figure 2. Save this result somewhat temporarily. Next, load the highest numbered cell (memory location) with the next highest numbered cell’s value and so on down to cell 2. At this point, all cells will have the value their predecessor had with the exception of cell 1. Now take the stored result of the addition and load it into cell 1. This completes a single shift operation and results in the generation of one number of the random sequence. You can use the value contained in any cell as your source of output but always use the same cell.

About the Programs

Program 1 (RNDGEN1) is an assembly language listing of a program that implements a 21 cell shift register operated MOD 10. Note that before you can use this routine you have to fill each of the 21 cells with some integer value less than 10. Program 2 is a BASIC program that loads the object code for RNDGEN1 and then lets you work with it from BASIC. It will ask you to type in a 21 digit number (the seed). Once you do this, the machine language routine will start producing random numbers. These will be printed on the screen as they are generated. You might observe the effect of different seed values on the output values.

This machine language routine could be the basis for a random number generator in lots of applications, but is presented here mainly to show how easily the concept is programmed. It might be inconvenient to load 21 numbers every time you needed this routine.

Program 3 (RNDGEN2) is a more extensive development of the idea. This routine already has 16 of the seed values pre-positioned in the shift register cells and you only need to supply five others. They go in the first five cells of the shift register (locations $232C through $2330). Then, before any numbers are produced as output, the shift register is shifted 2560 times. This ensures that the original seed has long since disappeared. Once this is done, shifting occurs normally. Notice that the first time you enter this program you do it at INIT (location $2341). The program then performs the 2560 shift operations described above. When this is completed, an RTS is executed returning control to the calling program. From here on you enter the routine at SHIFT (location $234A) each time you want to retrieve a random number.

Program 4 does the same thing to demonstrate RNDGEN2 as program 2 does for RNDGEN1 except that you only enter a five digit number. All of the programs will operate in both C-64 and C-128 mode (40 or 80 column display in C-128 mode).

Both of these assembly language programs produce as output a single integer from 0 to 9 for each shift operation. Of course, you may want a random number of some other length, say four or five digits in width. You can do this easily by taking successive values and assembling them accordingly. The exact details would depend on how you were using the random number generator, whether from machine language or BASIC, etc. As you can see, it is very easy to produce astronomically long sequences of pseudo-random numbers, (this 21 cell implementation generates 1021 numbers before it repeats). The technique is relatively straightforward to program at the assembly language level; even so, it is a powerful method for producing random numbers.

**PROGRAM 1: Source Code For RNDGEN1**

```assembly
base = $232b
cell1 = $2340
cell2 = $232d
cell1 = $232c
tmp = $232a
*= $2341
;
shift clc ;entry point
   lda cell1
   adc cell2 ;add these cells
   cmp #10
   bcc temp ;acc <10: already mod 10
   sbc #10 ;if not, make result mod 10
   temp sta temp ;store until shift is done
   ldx #20 ;# of times to shift
   loop lda base,x ;shift from here...
      sta base +1,x ;to here
      dex ;next lower cell
      cpx #0 ;done yet?
      bne loop ;no, then loop
      ldx base,xy ;get mod 10 addition result...
      sta cell1 ;and put in the first cell
      rts ;back to main prgm.
.end
```

**PROGRAM 2: Basic Demo/Loader For RNDGEN1**

```
CE 100 rem save "0:pgm2 *.8
OM 110 rem --- program 2 ---
IM 120 rem loads and runs rndgen1
NE 130 rem to demo random # generator output
HD 140 rem --- basic loader ---
```
PROGRAM 3: Source Code For RNDGEN2

```assembly
base    =  $232b
cell21  =  $2340
cell2   =  $232c
cell1   =  $232c
tmp     =  $232a
flag    =  $2329
hcnt    =  $2328

…….partial seed……
* = $2331
.byte 3,5,9,4,7,1,4,6,3,2,0,5,7,4,9,8
* = $2341

: init lda #0 ; entry point to reinitialize
sta hcnt    ; zero counter most sig byte
sta flag    ; clear flag
sta hcnt    ; zero counter least sig byte

: shift clr cell21
adc cell2
cmp #10
bcc temp    ; acc<10: already mod 10
sbc #10     ; if not, make result mod 10

temp sta temp ; store until shift is done
ldx #20     ; # of times to shift

loop lda base,x ; shift from here...
st base +1,x  ; goto here
edx #1,x     ; next lower cell
cpx #0       ; done yet?
bne loop    ; no, then loop
lda tmp     ; get mod 10 addition result…
```

PROGRAM 4: Basic Demo/Loader For RNDGEN2

```assembly
EE 100 rem save "0:pgm4 ".8
GN 110 rem -- program 4 --
JM 120 rem loads and runs rndgen2
NE 130 rem to demo random # generator output
HD 140 rem -- basic loader --
GA 150 for a = 0 to 90
EH 160 read b
HJ 170 poke 9009 + a,b
IL 180 next
MI 190 rem -- rndgen2 demo from basic --
GO 200 rem
IH 210 print " = = = 5 digit seed = = =
NH 220 for a = 0 to 4
KH 230 get k$: if k$ = ** then 230
IJ 240 $j = $j + k$

PN 250 poke 9004 + a, val(k$): rem cells 1 thru 5 get
    seed loaded
IE 260 print chr$(147);$§
CB 270 next
CM 280 sys 9025: rem = $2341
LF 290 print peak (9004): rem = $232c
EO 300 goto 280
EF 310 rem
DK 320 rem -- load data --
CF 330 data 24, 173, 64, 35, 109, 45, 35, 201
NG 340 data 10, 144, 2, 233, 10, 141, 42, 35
BH 350 data 162, 20, 189, 43, 35, 157, 44, 35
AF 360 data 202, 224, 0, 208, 245, 173, 42, 35
GF 370 data 141, 44, 35, 96
```

The Transactor 30
May 1987: Volume 7, Issue 06
N-Body Simulator
For The Commodore 64

Richard Lucas
W. Los Angeles, CA

The solar system is an example of a multiple-body system. Each planet is affected by the gravitational attractions of the other planets and, mostly, the sun...

Introduction

The motions of multiple astronomical bodies, affected only by the gravitational pull they exert upon one another, trace complex paths that usually defy easy analytical solution. Finding the motions of an arbitrary group of N bodies is called the N-Body Problem, which, in general, has no analytic solution.

The solar system is an example of a multiple-body system. Each planet is affected by the gravitational attractions of the other planets and, mostly, the sun. Jupiter and its many satellites are another multiple body system. Many stars in the galaxy come in clumps of two, three, or more suns orbiting around one another.

If one wants to compute the trajectories in an n-body system, two basic options are available: an analytical approximation, which usually requires some experience with mathematics and celestial mechanics; or numerical integration on a digital computer. Numerical integration is completely general, and requires a modest effort to set up and study a system.

N-Body Simulator (NBS) is a program for the Commodore 64 that solves the N-Body Problem by numerical integration. You can specify any system up to forty bodies and watch their motions as they are plotted in the Commodore 64's high resolution bit map mode. Besides having value as a display tool for astrodynamists, the program is educational since it allows you to reach in and manipulate celestial systems. (The screen displays are also very attractive, but this is not the main point of the program.)

Simulator Mathematics

The motion of each body is described by seven attributes: the body's location in xyz coordinates; the body's velocity in xyz coordinates (which are referred to as u, v, and w); and the mass.

NBS uses a coordinate system compatible with the Commodore 64 high resolution screen to display body motions. The x-axis is horizontal and positive going to the right, the y-axis is vertical and positive downward, and the z-axis, which really can't be seen in the display, goes directly into the screen. The upper left-hand corner of the screen is the coordinate (0,0,2), and the lower right-hand corner is the coordinate (319,199,2).

Body positions and velocities are stored in terms of "pixels" and "pixels per unit time". The distance each pixel represents depends upon the scale selected. Three scales are available in NBS:

1. 1 pixel = 10^7 kilometers
   1 mass = 1000 kilograms
   1 time = 1 day

2. 1 pixel = 1 astronomical unit (one AU equals the mean distance from the earth to the sun)
   1 mass = 1 earth mass (5.9742x10^24 kilograms)
   1 time = 1 day

3. 1 pixel = 10^6 meters
   1 mass = 1 kilogram
   1 time = 1 second

Scale 1 is appropriate for displaying the inner solar system out to Mars. Scale 2 allows the entire solar system to fit on the screen (though the orbits of the inner planets won't be very distinguishable). Scale 3 is intended for displays of near-Earth space.

Simulation Operation

The NBS command screen shows the attributes of one body from the system currently in memory and a menu of commands. The body attributes are displayed in a window. You can determine which body is displayed in this window by pressing the F1 or F7 keys. All commands can be executed from the main menu. Simply press the highlighted letter corresponding to the desired command.
### Commands

- **Exit** - End the program.
- **Plot** - Start the simulation.
- **New System** - Input the parameters for one or more bodies. The program prompts for the number of bodies in the system. NBS doesn't erase any values presently stored, so pressing RETURN at any prompt without entering anything simply retains the previous value.
- **Scale** - Change the scale used.
- **Display** - Change from sprite mode to high resolution mode, or vice versa. In high resolution mode the points representing the bodies are not erased, so each body gradually leaves a tracing of its path across the screen. Sprite mode can display a maximum of eight bodies.
- **Load** - Load an n-body system from disk.
- **Save** - Save the n-body system presently in memory to disk. NBS automatically adds the suffix "\*.nb" to the file name.
- **Previous Body** - Display the parameters of the previous body in the system list.
- **Next Body** - Display the next body.
- **X, Y, Z Position** - Enter a new value for the body currently in the data window.
- **U, V, W Velocity** - Change the velocity of the body currently in the data window.
- **Mass** - Change the mass of the body currently displayed.
- **Time** - Change the time step interval.
- **Rename** - Change the name of the body in the data window.

Pressing any key stops the simulation, but note that the current positions are lost when the display is interrupted in this manner. Pressing the F1 key interrupts the simulation and stores the current positions so that you can choose to pick up where you left off.

### Simulation Strategies

Accuracy depends on the size of the integration time step. Accuracy is achieved by using a small time step. On the other hand, a small time step makes the simulation proceed more slowly. The best compromise is the largest time step that gives acceptable accuracy.

For new situations trial and error will reveal the best time step. Start with a large time step (which takes less computation time), then decrease the time step until good results are achieved.

### Some Situations To Try

#### Inner solar system:

*For June 1, 1986 in heliocentric coordinates based on the ecliptic of 1950.*

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sun</td>
<td>1.989e27</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Mercury</td>
<td>5.97e20</td>
<td>147.86</td>
<td>104.25</td>
<td>0.5459</td>
<td>-0.4605</td>
<td>-0.1720</td>
<td>0.027607</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Venus</td>
<td>4.87e21</td>
<td>140.62</td>
<td>105.21</td>
<td>0.6143</td>
<td>-0.1482</td>
<td>-0.266</td>
<td>4.75e-3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Earth</td>
<td>5.97e21</td>
<td>144.74</td>
<td>85.771</td>
<td>1.22e-3</td>
<td>0.2372</td>
<td>-0.09</td>
<td>5.5e-6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Mars</td>
<td>6.42e20</td>
<td>147.84</td>
<td>78.184</td>
<td>-0.4083</td>
<td>0.2165</td>
<td>-2.76e-3</td>
<td>5.33e-3</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Outer solar system:

*For June 1, 1986 in heliocentric coordinates based on the ecliptic of 1950.*

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sun</td>
<td>332931.6</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Jupiter</td>
<td>317.867</td>
<td>154.63</td>
<td>98.144</td>
<td>-0.09663</td>
<td>2.719e-3</td>
<td>7.369e-3</td>
<td>-9.020e-5</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Saturn</td>
<td>95.243</td>
<td>145.96</td>
<td>90.827</td>
<td>0.31820</td>
<td>4.789e-3</td>
<td>-2.26e-3</td>
<td>-1.53e-4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Uranus</td>
<td>14.5459</td>
<td>149.67</td>
<td>80.722</td>
<td>-0.0684</td>
<td>3.903e-3</td>
<td>-2.477e-4</td>
<td>-5.143e-5</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Neptune</td>
<td>17.2408</td>
<td>151.71</td>
<td>69.756</td>
<td>0.5789</td>
<td>3.108e-3</td>
<td>1.942e-4</td>
<td>-7.629e-5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Pluto</td>
<td>2.176e-3</td>
<td>126.77</td>
<td>83.627</td>
<td>8.443</td>
<td>2.044e-3</td>
<td>-2.858e-3</td>
<td>-3.062e-4</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Trinary system:

Scale = 1

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sun #1</td>
<td>3e27</td>
<td>150</td>
<td>90</td>
<td>0</td>
<td>-15</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Sun #2</td>
<td>3e27</td>
<td>150</td>
<td>135</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sun #3</td>
<td>1e27</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Klemperer's Rosette

Scale = 1

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sun #1</td>
<td>1.8e29</td>
<td>120</td>
<td>70</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Sun #2</td>
<td>1.8e29</td>
<td>180</td>
<td>70</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sun #3</td>
<td>1.8e29</td>
<td>180</td>
<td>130</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Sun #4</td>
<td>1.8e29</td>
<td>120</td>
<td>130</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Binary star system:

Scale = 1

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sun #1</td>
<td>3e27</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>-15</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Sun #2</td>
<td>3e27</td>
<td>150</td>
<td>125</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Simple System

Scale = 3

<table>
<thead>
<tr>
<th>Body#</th>
<th>Name</th>
<th>Mass</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Heavy</td>
<td>1e30</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Light</td>
<td>1e10</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
N-Body Simulator

IK  100 rem n-body simulator
OO  110 rem version 6.09
IF  120 rem by richard lucas
GP  130 :
NI  140 rem initialize
FO  150 poke2038,peek(55):poke2039,peek(56)
DC  160 poke 56,62: poke 55,0: clr
NF  170 dim x(40),y(40),z(40),u(40),v(40),w(40),
x1(40),y1(40),z1(40)
EB  180 dim m(40),gm(40),e2(7)
MM  190 dim xo(40),yo(40),zo(40),uo(40),vo(40),
w(40),ao(40),bo(40),co(40)
OM  200 dim ex(7),un(3),un$3(3),tu$3(3)
CG  210 for i = 1 to 3:readtxt$(i),un$(i):next
ID  220 data "days", "1017 kilometers–tons–days", "days", "AU–Earth mass–days", "seconds"
GN  230 data 100km/kg/sec
CB  240 gosub 2820
ME  250 g = 6.67e-11: sy = 1: cf = 40: c4 = 504: c7 = 7
: c8 = 248: vi = 53248: hi = 16: c5 = 255
LG  260 hr = 16: t0 = 124: s = "epncdls" + chr$(133)
+: chr$(136) + "xyzvwrmt": sp = 0
AA  270 nb = 0: dt = 10: d2 = dt*dt/2: d3 = d2*dt/3
: d4 = d3*dt/4: cb = 7
MA  280 for i = 0 to 7: ex(i) = 2*(i-1):next
PC  290 for i = 0 to 7: e2(i) = 2*i:next
FI  300 un(1) = 1000 + 8640002: 1e1013
FP  310 un(2) = 5.9742e24 + 8640012: 1.4959789e1113
NJ  320 un(3) = 1/1e613
PL  330 poke 53280,0:poke 53281,0:print chr$(14)
+: chr$(8) + chr$(151);:
NO  340 b$ = "": s$ = "": for i = 1 to 38: b$ = b$ + chr$(32)
+: s$ = s$ + chr$(192): nexti
NH  350 b$ = chr$(29) + b$ + chr$(29)
MN  360 :
PG  370 rem main loop
CJ  380 gosub 2460
AK  390 gosub 2620
IL  400 print "": ;
AP  410 get a$: if a$ = "": then 410
OD  420 for i = 1 to 18: if a$ = mid$(o$,i,1):print a$
+: goto 450
CL  430 next
KF  440 gos 410
EL  450 on i goto 490,510,1970,2900,3070,2110,
+: 2250,3190,3230
JL  460 if i = 18 then input " New name of body ": n$(cb)
+: goto 370
NJ  470 input " new value ": nv
JK  480 on i = 9 goto 2390,2400,2410,2420,2430,
+: 2440,2380,2450
BD  490 poke555,peek(2038):poke56,peek(2039)
+: clr:end
IG  500 :
OB  510 rem plot trajectories
IJ  520 if nb = 0 then print " No bodies in current
+: system.": goto 370
JI  530 if sp then 610:rem skip hires for sprites
BC  540 rem set up hires screen
GI  550 poke 56578,peek(56578)or3:rem switch to vic
+: bank 1 (16k–32k)
MC  560 poke 56576,peek(56576)and252)or2
GO  570 poke 53272,peek(53272)and15)or128
+: :rem char screen is in 9th k
MP  580 poke 53265,peek(53265)or32:rem turn on
+: hires screen
HC  590 poke 820,0:poke 821,64:poke 822,0
+: poke 823,96:poke 251,0:sys 49152
AL  600 poke 820,0:poke 821,96:poke 822,231
+: poke 823,99:poke 251,1:sys 49152
FN  610 ifspthenfori = vti0:pekei...next
MN  620 ifspthen a1 = 0:fori = 1to8:a1 = a1 or
+: (i<nb)*2*(i–1):next: poke vi + 21.a1
AK  630 ifspthenpoke 53281,0:printchr$(147)
OP  640 f0 = 0
OP  650 :
NE  660 rem move start parameters to working arrays
OK  670 for i = 1 to nb
OH  680 x[i] = xo[i]
NI  690 y[i] = yo[i]
MJ  700 z[i] = zo[i]
NI  710 u[i] = uo[i]
MJ  720 v[i] = vo[i]
KL  730 w[i] = wo[i]
IO  740 next
IM  750 rem compute accel at time dt before start
IA  760 for i = 1 to nb
NM  770 a0[i] = b0[i] = c0[i] =
AB  780 next
GC  790 for i = 1 to nb
DE  800 ax = ay = az =
ND  810 for j = 1 to nb
EC  820 if j = then910
AQ  830 dx = x[j]–x[j–1]
AP  840 dy = y[j]–y[j–1]
AA  850 dz = z[j]–z[j–1]
MK  860 r = sqrt(dx*dx + dy*dy + dz*dz)
HK  870 r3 = r*r*r/gm(i)
FD  880 ax = ax + dx/r3
FE  890 ay = ay + dy/r3
FF  900 az = az + dz/r3
CJ  910 next
AK  920 x(j) = x(i)–u[i]*dt + ax*d2
DL  930 y(j) = y(i)–v[i]*dt + ay*d2
GM  940 z(j) = z(i)–w[i]*dt + az*d2
KL  950 next
AN  960 for i = 1 to nb
NO  970 ax = ay = az =
HO  980 for j = 1 to nb
KB  990 if j = then1080
KB  1000 dx = x(j)–x(i)
LC  1010 dy = y(j)–y(i)
MD  1020 dz = z(j)–z(i)
GF  1030 r = sqrt(dx*dx + dy*dy + dz*dz)
BF  1040 r3 = r*r*r/gm(i)
PN  1050 ax = ax + dx/r3
PO  1060 ay = ay + dy/r3
PP  1070 az = az + dz/r3
MD  1080 next
BA  1090 a0[i] = ax

The Transactor
33
May 1987: Volume 7, Issue 06
A Two-Button Mouse

Anshol Bryant
Winnipeg, Manitoba

...experimenting with the C-1350 Mouse, and more...

If you are presently using the old digital joystick or paddles, to move a cursor around the screen, or draw with, then join the mouseketeers - try this new mouse!

On the C-128 (or C-64 with Super Expander cartridge) from BASIC, you can read the mouse with the JOY() function. The left button is read like the "fire" button. Only the left button! But, you say (I sure did!) this mouse, which has the same physical appearance as the Amiga mouse, has two buttons - left and right!
(Hmmm)

As no mention is made of the right button in the manual, I decided to dissect this little critter to see why.

Mouse Pinouts

Internally, this is a state-of-the-art mouse. Two optically-encoded discs, set in motion by a rolling ball, generate phase-quadrature pulses. These are decoded in hardware (using op-amps and comparators) and four outputs are generated - UP, DN, LFT and RHT. Two active pushbuttons are also output! My findings are tabulated in FIGURE 1.

FIGURE 1: Control Port Pinouts

<table>
<thead>
<tr>
<th>Pin</th>
<th>Joy</th>
<th>Mouse</th>
<th>Stick</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Up</td>
<td>Up</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Down</td>
<td>Down</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Left</td>
<td>Left</td>
<td>Button 1</td>
</tr>
<tr>
<td>4</td>
<td>Right</td>
<td>Right</td>
<td>Button 2</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>Pot Y</td>
</tr>
<tr>
<td>6</td>
<td>Button 1</td>
<td>Button 1</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>Button 2</td>
<td>Pot X</td>
</tr>
</tbody>
</table>

FIGURE 1 shows the pinouts for three types of Control Port input devices for comparison - the digital JOYstick, the MOUSE, and the analog joySTICK. On the C-1350 mouse, the left button comes out on Pin 6 - same as the JOY "fire" button and the right button comes out on Pin 9 - same as the STICK's POT X line. POT X and POT Y are inputs (READ ONLY) to the A/D converter used to digitize the analog position of potentiometers. The diagram below of register $8DC00$ of CIA 1 shows the bit distribution for each device's digital logic lines.

In order to use the right button output, which like the left button output, is simply a switch closure to ground, a combination of digital and analog techniques is needed.

FIGURE 2: Analog Joystick Schematic

FIGURE 2 is the schematic of a two button analog joystick. It could also be for a graphics pad, or other homebrew configuration type of input device. It graphically shows that all that is required on the POT X line is a potentiometer between Pin 9 and Pin 7 (+5v).

CIA 1 - Register $8DC00$

<table>
<thead>
<tr>
<th>Joy</th>
<th>Stick</th>
<th>Mouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 2</td>
<td>Port 1</td>
<td>Port 2</td>
</tr>
<tr>
<td>Port 1</td>
<td>Port 1</td>
<td>Port 1</td>
</tr>
<tr>
<td>BTN 1</td>
<td>BTN 1</td>
<td>BTN 1</td>
</tr>
<tr>
<td>Right</td>
<td>Left</td>
<td>Down</td>
</tr>
<tr>
<td>Left</td>
<td>Down</td>
<td>Up</td>
</tr>
</tbody>
</table>

May 1987: Volume 7, Issue 6
FIGURE 3 is a simple schematic of the C-1350 mouse. The left button is BTN1 and the right button is BTN2. The Control Port connects BTN2 with register $D419. This register allows the microprocessor to read the "position" or, in this case, the "logic state" of the POT X line, with values ranging from $00 at minimum resistance (=logic high) to $FF at maximum resistance (=logic low). Switch closure generates the logic low, and what is needed to generate the logic high is a "pull-up" resistor between Pin 9 (BTN2) and Pin 7 (+5v).

Mouse Modification

Experimenting with a variable potentiometer and scope, I found a value of 47k to be about right for the "pull-up" resistor. It's not critical, but should range between 22k and 100k.

There is room inside the mouse for one resistor. I used a tiny 1/8 watt 47k resistor. Also required is a Phillips-head screwdriver and a fine point low-wattage soldering iron.

Two screws on the underside of the mouse hold the case halves together. Two screws inside hold the p.c. board assembly (and pushbutton sub-board) to the bottom case half.

FIGURE 4 shows the layout of the p.c. board. The mouse cable plugs into a Block Plug on the board.

The colour code of the Block Plug is as follows:

<table>
<thead>
<tr>
<th>Block Plug Pin</th>
<th>Colour</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yellow</td>
<td>RHT</td>
</tr>
<tr>
<td>2</td>
<td>orange</td>
<td>LFT</td>
</tr>
<tr>
<td>3</td>
<td>red</td>
<td>DN</td>
</tr>
<tr>
<td>4</td>
<td>brown</td>
<td>UP</td>
</tr>
<tr>
<td>5</td>
<td>white</td>
<td>GND</td>
</tr>
<tr>
<td>6</td>
<td>blue</td>
<td>+5v</td>
</tr>
<tr>
<td>7</td>
<td>green</td>
<td>BTN1</td>
</tr>
<tr>
<td>8</td>
<td>black</td>
<td>BTN2</td>
</tr>
</tbody>
</table>

Solder the resistor between pins 6 and 8 of this Block Plug on the foil side and re-assemble the mouse.

Mouse Machine Code

Now that we have a right button with two discernable logic states, some code is needed to use it. This is typical coding, modularized so that you can adapt it to your specific needs.

The object is to maintain x and y coordinates and report button status.

```
;define variable labels
XPOS .BYT 0 ;save x position
YPOS .BYT 0 ;save y position
BTNS .BYT 0 ;save btn status

;subroutine to read Control Port 2
RDPORT SEI ;lock out keyboard
LDA #$C0 ;
STA #$C02 ;set ddr to read
LDA #$80 ;
STA #$C00 ;read Control Port 2
LDX #$00 ;allow time for
INX:BNE +1 ;lines to settle
LDX #$419 ;read POT X
LDY #$41A ;read POT Y
LDA #$FF ;
STA #$C00 ;reset Port 2
RTS ;
```

You could now simply store the 'absolute position' x and y coordinates just read, for the analog joySTICK, but imperfections in the pots results in jitter on the screen. Some finesse is in order! A moving average algorithm smooths out the rough spots.
; subroutine for moving average algorithm
AVRG BCS AVRGP ; if sign positive
AVRGN EOR #=1 ; if negative, do
ADC #1 ; reverse subtr
LSR ; allow half-weight
EOR #=1 ; invert byte
CLC ; to preserve sign
ADC #1 ;
CLC:RTS ;
AVRGP LSR ; allow half-weight
CLC:RTS ; to the byte

Putting this altogether yields a simple, smooth routine for getting the 'absolute position' from the STICK.

; main routine to read the STICK
STICK JSR RDPORT ;
LDA $D0C0 ; read Port 2
AND *$OC ; filter BTN1 & 2
EOR *$FF ; invert logic
STA BTNS ; save BTN1 & BTN2
TXA:SEC ; store x in XPOS
SBC XPOS ; using a simple
JSR AVRGN ; moving average
ADC XPOS ; algorithm
STA XPOS ; update XPOS
TYA:SEC ; store y in YPOS
SBC YPOS ; using the same
JSR AVRGN ; algorithm
ADC YPOS ; and update YPOS
STA YPOS ;

; now test the buttons and exit
TEST LDX *$FF ;
STX $D0C2 ; reset ddr
CLI ; finished with Port
LDA #4 ; test bit 2
BIT BTNS ; of BTN5
BNE BTN1 ; if BTN1 pressed
ASL ; test bit 3
BIT BTNS ; of BTN5
BNE BTN2 ; if BTN2 pressed
RTS ; exit Z = 1 no btns

BTN1 LDA #=1 ; flag for BTN1
.BYT $2C ; skip over
BTN2 LDA #1 ; flag for BTN2
RTS ; exit Z = 0

Mouse Moves

Getting 'absolute position' from the MOUSE is not possible. We have only 'relative position' to work with - incrementing or decrementing XPOS and YPOS as the mouse moves.

; main routine to read the MOUSE
MOUSE JSR RDPORT ;
LDA $D0C0 ; read Port 2
AND *$10 ; filter BTN1
STA BTNS ; save bit 4
TXA ; read POT X
BMI LOW ; discern state
HIGH LDA *$20 ; set bit 5
.BYT $2C ; skip over
LOW LDA *$00 ; clr bit 5
ORA BTNS ; combine bits 4 & 5
LSR:LSR ; shift to bits 2 & 3
EOR *$FF ; invert logic
STA BTNS ; save BTN1 & BTN2

; now we have our left and right buttons!
LDA $D0C0 ; read Port 2
AND *$0F ; filter directions
CMP *$0F ; any movement?
BEQ EXIT ; no, finish up
TAX ; yes, mouse rolling
UP AND #1 ; check up
BNE DN ;
INC YPOS ;

DN TXA ;
AND #2 ; check down
BNE LFT ;
DEC YPOS ;

LFT TXA ;
AND #4 ; check left
BNE RHT ;
DEC XPOS ;

RHT TXA ;
AND #8 ; check right
BNE EXIT ;
INC XPOS ;

EXIT JMP TEST ; test buttons and exit

Use a BEQ to test for buttons pressed, after a JSR MOUSE call. Use a BPL or BMI to check which button is active.

The 'relative movement' of the MOUSE routine is very responsive. The speed of movement and the area of movement, i.e., the "feel" of the mouse depends on how often the MOUSE routine is polled. Possibilities, I leave to the mouse programmer!
Universal EPROM Programmer Update

The Universal EPROM Programmer project that appeared two issues ago (Volume 7, Issue 04, "Gizmos and Gadgets") has generated response beyond all expectation. Unfortunately, most of the response resulted from some unforeseen difficulties that prevented many from completing a fully operational programmer. This follow-up provides corrections (including a couple described last issue) and also gives test procedures, some enhancements to make the burner compatible with more EPROM types, and instructions for a 2764 adapter.

A Few Notes

For those of you who were brave (ambitious) enough to build the EPROM programmer that appeared in the Jan 86 issue of TRANSACTOR this article is for you. And for those that didn’t build the programmer because it seemed too hard, I hope the following information will inspire you to try it. This article will attempt to clarify some unclear areas in the first article and will point out some errors that crept into the schematic (Murphy does live!)

Corrections to the Schematic

The schematic shown in figure 1 looks similar to the original but contains the necessary corrections. The circled areas indicate the corrections as well as some needed changes.

1. The 8255 to the left of the schematic shows two pins numbered 14. The CGN pin should be pin 7, same as the other 8255.
2. On the ZIF socket, pins 14 through 18 should be relabelled pins 15 to 19. Pin 14 is GND.
3. On U3, the NAND gates, Pin 14 goes to +5V, pin 7 is GND.
4. The emitter of Q1 goes to GND.
5. In lines 2760 and 2770 of the program, the ’’ should be replaced by a ’’ (i.e. null string).

A wire was added from pin 13 of U1 to pin 4 of the personality socket. This signal becomes OE4 that is 0 (or low) for read, 0 for standby, and 1 during programming. This is used for some versions of the 2716 EPROM that did not work with the personality socket wiring supplied with the original article. To support this new signal requires the modification of three lines of the program. (Note: changing these lines won’t affect the operation with other EPROMs.)

\[
\begin{align*}
1680 & \text{pake } 16384,239 : \text{for } t = 1 \text{ to } 1000: \text{next } t \\
1770 & \text{pake}16384,21 \\
1780 & \text{pake}16384,239
\end{align*}
\]

An additional wire was added from pin 22 of U1 to pin 3 of the personality socket. This adds A2 to the personality socket for use in reading masked ROMs of 8K or larger.

Please note the wiring corrections for the personality sockets. I am sorry if this caused any problems for those of you who built the programmer. This wiring has been re-checked and verified for proper operation.

It was found that for some transistors used for Q1 the 220 ohm resistor was too high in value and needed to be changed to 75 ohms. This value is not critical and can be anything close to 75. Make sure you wire the transistors correctly! The relay shown must be a small DPDT. The relay used in the prototype, purchased at Radio Shack, has a coil voltage of 5 volts and a coil resistance of 150 ohms. A DPDT switch can be used in place of the relay but subtracts from the automatic operation a bit. Besides that, if you forget to flip the switch, and leave programming voltage (25 volts, for example) on Vpp, accidental erasure could destroy your EPROM. (Ah, experience is a tough and expensive teacher!)

The Circuit Board

I am sure many of you are wondering about the circuit board used. It is a Radio Shack catalog #276-166. It was cut in half and trimmed from the 25/50 .100 inch fingers to 22/44 .100 inch fingers. The board is alas unavailable but still might be in the junk box at your local Radio Shack dealer. (Note: Jameco in CA sells a C64 cartridge port compatible perf board for about $8.00) The other alternative is to use the fingers from an old discarded cartridge. By trimming the foil back and carefully cutting the board the assembly can then be attached to larger perforated board such as a Radio Shack catalog #276-147 or #276-191. Make sure that the cartridge used has all the needed fingers. Necessity is the mother of invention!

Information Please

Information about the pinouts of the expansion bus for the C64 was found in the ‘Programmers Reference Guide’ published by Commodore. Detailed information about the 8255’s is found in Intel’s ‘Component Data Catalog’ available from any Intel distributor. Check your local electronics supply houses for a copy too.

Vpp

Considering the vast number of types of EPROMs available, the original article left it up to you to determine the correct programming voltage (Vpp). Programming voltages can vary from 12.5 volts for certain 27256s to 25 volts for most garden variety EPROMs. Intel makes a version of the 2732 called the 2732A that programs at 21 volts. Please verify the voltage that your EPROM requires from the data sheets supplied with your particular EPROM. Programming at a voltage higher than recommended WILL result in destroying the chip. Try using a slightly reduced voltage first (for example 22 volts for a 25 volt EPROM) and raise it up only if it doesn’t work. Using the variable power supply shown in the original article will allow you to adjust for any programming voltage you may encounter.

Testing the Completed Programmer

One time consuming but essential step in testing the circuit is to use an ohmmeter to check continuity. Unplug the chips and do not plug
the programmer into the C64. Now, using the schematic, verify every
connection. This finds 99.9% of all problems with the circuit. Verify also that pins (2,3) and (1,2,2,A,Z) of the board are not shorted
together. Fingers 2.3 are the +5 volt supply and 1,2,2,A,Z are the
ground. Verify also that no other fingers on the board are directly
shorted to ground. With that complete the next test should be done.

With the board plugged in and power turned on, the 64 should power
up with the usual message on the screen. If this is not successful, re-
check all connections and solder joints. Once you can get the board
powered up, the battle is half over. Make the following checks with a
voltmeter set to read 5 volts.

<table>
<thead>
<tr>
<th>POSITIVE</th>
<th>NEGATIVE</th>
<th>READING</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin 26 U1</td>
<td>pin 7 U1</td>
<td>5.1 v</td>
</tr>
<tr>
<td>pin 26 U2</td>
<td>pin 7 U2</td>
<td>5.1 v</td>
</tr>
<tr>
<td>pin 14 U3</td>
<td>pin 7 U3</td>
<td>5.1 v</td>
</tr>
<tr>
<td>pin 28 ZIF</td>
<td>pin 14 ZIF</td>
<td>5.1 v</td>
</tr>
<tr>
<td>pin 13 Pskt</td>
<td>pin 1 Pskt</td>
<td>5.1 v</td>
</tr>
</tbody>
</table>

If any of the readings above are incorrect check the wiring of the 5 volt
supply (pins 2,3 and 1,2,2,A,Z, on the fingers of the board).

If you have gotten this far you are almost there! With a voltmeter set to
read 5 volts connect the positive lead to pin 35 of U1 or U2. Connect
the negative lead of the voltmeter to pin 7 of U1 or U2. You should
read close to 0 volts (less than .8 volts). If not, start over with the
ohmmeter check; it is also possible that you have a defective 7400. If
that test was successful, with the voltmeter still connected as above,
depress the reset button and hold it. The voltmeter should read close
to 5 volts (greater than 3.5 volts). If not, check the wiring of the reset
circuit and the IN914 diode. With this test done, you are ready to
proceed to the software test.

A short program appears at the end of the article that will assist in the
testing of the completed programmer. This program will allow you to
selectively turn on certain pins of the 8255A and check them with the
voltmeter. Connect the negative lead of a voltmeter set for 5 volts to
pin 7 of U1. Then connect the positive lead to the pin indicated by the
program. Referring to the schematic, there are four ports to test: port A
of U1, port B of U1, port C of U1, and port B of U2. The first menu of
the program allows you to select the bit (or pin) to test. For example, if
you wish to test port B of U2, select 4 from the main menu. The
screen will then indicate that you are testing Port B of U2. To turn a bit
(or pin) of the 8255 'on', enter the desired bit number and press return.
That bit only will be turned on. The 'on' state is represented by a
voltage greater than 3.5 volts. Selecting another bit turns the last bit
off and the new one on. Entering '8' for a bit number returns you to
the main menu. After testing all four ports of the 8255A you are ready
to try your first EPROM.

Using the Programmer

Let's examine a few ways to use the programmer. Suppose you wish to
make a modification to the Kernel rom in your C64. First, load your
favorite monitor program into the C64 and enter the monitor. To
modify the Kernel you must first relocate it. Let's assume you transfer
from $E000-$FFFF to $6000-$7FFF. Now, using your monitor, make
the desired changes in memory at the new locations inside $6000-
$7FFF. When you are done save the entire 8K block to disk as a
program file. That's all that is needed to use it with the EPROM
programmer.

To make the EPROM, install the programmer into the cartridge port
and turn power on. Load the EPROM programmer program supplied
in the January 87 issue of Transactor and place the disk with your new
Kernal in the drive. With the program running, select menu item #2
(PROGRAM EPROM). The program will ask you to select the size of
EPROM you are going to program; enter '3' for an 8K EPROM. Next
the program will ask for a file name, so type in the name of the new
cernal that you just made. If the file is found the program will ask you
to press a key when ready. Place a blank EPROM in the programmer
socket (2764 in this case) and the proper personality plug for the
EPROM used. Connect or turn on your source of Vpp programming
voltage (25 volts for a standard 2764). Pressing any key will start the
programming process. First you should notice the LED associated
with the relay and the relay turning on. As the location number on the
screen counts up you should notice a slight blink of the CE LED. The
CE LED is on most of the time and is off for only a very short time.
Programming will take a while (about 10 minutes?). When the program
is done the main menu will re-appear. Select the option to verify
EPROM with disk and answer the questions as they appear. If all is
well the program will return to the main menu.

An 8K 2764 EPROM has 28 pins and the Kernal ROM socket has only
24 pins. Figure 2 shows how to make an adapter from a 28 pin socket
and a 24 pin DIP header. This will work for the Basic ROM also.

Now let's try just copying another EPROM. First install the program-
mer and load the program as described previously. Place the EPROM
to be copied into the EPROM socket. Make sure that you have already
installed the proper personality socket for the EPROM. Select the
option to copy the EPROM to disk. Answer the questions asked by the
program and give the file a name. What you are doing is creating a
disk file to later burn into a blank EPROM. Reading an EPROM doesn't
take as long as programming one. When the program is done copying
the EPROM to disk, follow the same procedures as described previ-
ously.

For a third example, let's assume you have a machine language
program you want to put on an EPROM. First, assemble the file in
memory and save the file as a program as described in the first
example. If you assemble directly to disk you will have to load the
object file into memory and re-save it. This is required if your
program file is not exactly the same size as your EPROM. If your file is,
for example, only 1K long and you are using a 2K EPROM, before
loading your program file, fill a 2K block of memory with the value
$FF (255 decimal). Then load your file to this block and re-save the
entire 2K block. This will allow you to add to that EPROM later
without erasing the whole EPROM.

Figure 3 shows an additional personality socket for reading the C64
Kernel and Basic ROMs directly. This allows you to create a file to be
loaded, modified, and recopied to an EPROM. An interesting project
might be to modify the character generator ROM in the C64 and create
your own set of characters.

The last example is for those of you that like to program on the rock
(right in hex code). To program an EPROM for use in a different
computer or as a character generator or logic array, use your favorite
monitor and use the display or memory dump command. Fill an
entire block the size of your EPROM with the value $FF (255 decimal)
first. As you should know, an erased EPROM contains all $FFs as the
stored value. When you are done entering the values for your EPROM
in memory, save the entire block as a program file and follow the first
example for programming the EPROM.

The uses of the programmer (as in any tool) are limited only by your
imagination.

If you require a faster programming time, you can use a Basic
compiler. Keep in mind that the programming pulse time for an
EPROM is at least 50 milliseconds and you must add the following line to your program:

1775 FOR CC = 1 TO 30: NEXT CC

One last word of caution, do not erase an EPROM for longer than just required (about 15 minutes for most erasers), and use a good controlled voltage supply for Vpp.

If you get your programmer working, drop me a line and let me know. Or if you have improvements, that's nice to know too.

Editor's Note: Since publishing his EPROM burner, Tim has been invited by several user groups to give presentations... and has accepted, if for no other reason than to cut down on phone time spent assisting callers who built the burner. Tim has invited anyone to call or write. Also, send an SASE and disk, and Tim will return it with the testing and burner programs. Write to: Tim Bolbach, 1575 Crestwood, Toledo, Ohio, 43612

Figure 2: 2764 (28 pin) TO 2364 (24 pin) Adapter

Use a 28 pin WW socket and a 24 pin ribbon cable header (male). On the 28 pin DIP, cut down pins 1, 2, 20, 23, 26, 27 and 28 to about 3/8". Using wire wrap, short pins 1, 26, 27 and 28. Connect pin 27 to pin 24 of the cable header, pin 23 to pin 18, pin 20 to pin 12, and pin 2 to pin 21. The long pins will plug directly into the cable header such that pin 3 of the 28 pin WW goes to pin 1 of the 24 pin header (pin 4 to pin 2, 5 to 3, etc., 25 to 23, 26 to 24).
Figure 1: EPROM Programmer Schematic with corrections and changes
Help! Help!
by Nick Sullivan and Chris Zamara

Instant Help from an unexpected place!

Here is a utility that will provide you with instant, on-line help, from just about any application you may be running. The program can eliminate the need for a pile of manuals by your side, and can speed up the learning process when it comes to using a complicated new piece of software. Creating such a utility – one that can work alongside other programs – can be a most tortuous endeavour. Come join us in our adventure as we try to build a transparent background task on the Commodore 64!

The question is: how do you write totally transparent code?

We wanted to write a program that, at the touch of a key, would bring up a menu of help topics. The touch of another key would choose a topic, and a corresponding sequential file would be brought in from disk and printed to the screen. At the touch of yet another key, the original environment would be restored, and whatever was going on beforehand would continue as if nothing had happened.

These operations require a fair amount of code. We have to:

1) trap the first special keypress 2) save the user’s low RAM and screen in some secret place, to be restored later on 3) print the file menu 4) wait for a selection 5) open a file, or exit the help utility and restore the original context 6) print the file, pausing after each screenful until the user wants to continue 7) close the file and go back to step 2

At the same time, we want this code to be undisturbed by most programs and to consume a minimum amount of precious RAM. How?

The answer is: you don’t.

If your help program is on a cartridge, you can make it almost completely transparent to other applications. But cartridges are too expensive and difficult to manufacture for everyday purposes.

If your help program is running under a multitasking operating system like the Amiga’s, transparency is again not hard to achieve. But we’re talking about the Commodore 64 here and, the last time we looked, multitasking wasn’t available.

So we do what programmers have always done on the 64: look for some space for our program where the traffic is especially light, and hope that nobody dumps on it. It used to be that short utilities would normally be placed in the cassette buffer. This was an inheritance from the PET days, when there were few other places for machine language programs to go. But our program is not that short and, in any case, the cassette buffer is busier now.

The next choice was the 4K of RAM at $C000, which was either a design quirk or an intelligent decision on the part of Commodore, depending on how you look at it. Naturally, it soon became impossible to depend on anything surviving in the C-block for any length of time.

Next came the top of BASIC, also popular in PET days. This had some advantages. BASIC memory in the 64 is fairly large, so you can take away some without inflicting a great penalty on the user. It is also less used than the C-block, so your program has a correspondingly greater chance of surviving (as long as you protect it from BASIC itself by adjusting a couple of pointers). But it also has drawbacks. For one thing, you never know in advance just where the top of BASIC is, so you have to make your code relocatable. For another, this area of memory gets more use now than it did once upon a time (POWER, PAL and Supermon all use it, for instance) so the risk of inconveniencing BASIC by using up too much RAM has grown over the years.

That leaves us with the ‘hidden’ RAM underlying the ROMs. In the early days, this RAM was a godsend – nobody used it for anything. If you could get your code in there, and get at with some wedging scheme or other, you were home free. No longer. Programmers discovered that the video chip can look at that RAM without problems, and started using the RAM at $A000 and $E000 for high resolution screens, character sets and sprite data. They also found that machine language can be run quite nicely out at $A000 (just kick out BASIC for a little while) and even at $E000 if you don’t need the Kernal for I/O (and you can still load into that RAM without special precautions).

Like some peaceful tribe driven into increasingly inhospitable territory by hostile neighbours, utility writers were inevitably forced into these unfriendlier regions of memory in order to have any hope that their code would survive the competition for RAM. One by one, these newly-mastered areas just as inevitably became overcrowded in their turn.
The only place that was generally ignored was the one we haven’t yet mentioned: the D-block of RAM that underlies not just ROM (in this case the character set ROM) but also the input/output registers for the VIC II, SID and CIA chips. You can’t load into this area - it writes over your I/O registers, which is a pretty sure-fire means of crashing the system. You can’t save from it, because the RAM and the I/O chips needed to communicate with the disk drive are not simultaneously accessible. What’s worse, any code executing in the D-RAM cannot directly access any I/O (the video chip, SID chip, serial port, etc.), because when the D-RAM is selected, all of the I/O is switched off. If there is a use for this RAM, it’s certainly not obvious at first glance.

Where, then, do we put our help utility? Reluctantly, we put it in the D-block.

Having made this fundamental, and possibly misguided design decision, let’s follow the programming steps through in more detail. First, it’s obvious that we can’t put everything under ROM - if we did, we couldn’t reach our code to execute it in the first place. So we need a bit of link code that will switch out the overlying stuff in the D-block (by poking a $34 into the memory configuration register at location 1) then jump to the real program. Oh yes, we also need another link to bring the ROMs back in (with a $37 in location 1) so we can continue life as normal after the code has finished executing.

This brings back the same old question - where can we put our code (the link code this time) so that it will get in the way of the fewest other programs? Well, we’re only talking about two very short routines now, so the top of BASIC space seems like a natural: the user is not likely to notice the difference, and most programs that inhabit this area are relocating, as mentioned above, so we probably won’t get written over. Sometimes, though, the top of BASIC won’t be safe, so we’ll include an option to put the link code somewhere else in memory if the user desires.

To recap: the main program will go in the D-block, where it should be safe from most competition. The short routines will go either at the top of BASIC, where considerate programs like POWER and others will leave them alone, or at some other place to be determined by the user.

Now back to the seven steps described above:

1) To trap the first keypress, we’ll use the vector at $028F. This vector (known as KEYLOG in the memory maps), is the stepping stone to the ROM routine that converts raw keycodes to real PETSCII characters, and is invoked during every IRQ interrupt just after the keyboard has been scanned. It is not very well-known, so not many programs bother to ensure that it is set to its default value as part of their initialization sequences, which is less true of the more popular vectors in page 3 (it is reset by RUN/STOP-RESTORE, however). We’ll substitute for this vector the address of our own routine in ‘open’ RAM, whose function will be simply to switch out the ROMs (all the ROMs – we’ll have a pure RAM computer with no I/O registers at this point) and call our main program in the D-block.

2) Before it does anything else, that program has to save the user’s current environment so that it can be restored later on. We’ll want to save all parts of memory that the help utility itself will subsequently affect. These include: zero page, the stack (page 1), a few bytes in page 2 ($46 through $48), the low-res screen area we’ll be using (pages 4 through 7), colour RAM (pages $DB through $DB), and the video registers from $D011 through $D021. We’ll store all this data in the safest place we know about – right alongside our program itself in D-block RAM.

Unfortunately, this presents a new problem. To write stuff into D-block we need to have the I/O registers switched out. But to save the colour RAM and the video registers we need the I/O switched in. The answer? More link code. To minimize our demands on the user’s RAM, we’ll copy that code into one end of page 1 (the stack page) every time the help utility is invoked, and execute it there. At the same time, we’ll move the stack pointer to the other end of the stack so there’ll be no conflict. (The stack will be restored to its original state after the help utility has finished executing.) But now that we’ve broken this new ground, we may as well use it for more than just copying I/O registers to RAM. We’ll also put here our routines for copying the RAM back to the registers (which we’ll need when we’re ready to exit), and our disk I/O routines (for opening, getting bytes from, and closing the disk files, for keyboard input and for screen output).

3) Now we have to print the file menu. No problem. We already have the data in the D-block, so we just make repeated calls to our print character routine in page 1 till the printing is done.

4) Now we wait for the user’s selection. Again the routine we need is going to be on the stack page. Rather than do a JSR GETIN here, we preferred to do a JSR SCNKEY and interpret the keycodes ourselves. This avoids complications with interrupts, and also lets us look at the logo key (our escape key) and the main keyboard separately.

5) To open the file we call another of our routines on the stack page. By this point in the project, we’ve settled on standard filenames of the form ‘help-a’, ‘help-b’, and so on, which means that we only have to pass the distinguishing final character of the name to our open routine.

6) We fetch and print the file byte by byte, waiting for a signal to either continue or abort after each screenful. Along with our routines for printing to the screen and scanning the keyboard, we need a new one to get bytes from disk; this will go on page 1 with the others.

7) Finally we have to close the file (also via a routine in page 1) and go back to step 2.
Okay, all this stuff has been worked out, and the code has been written. Now all we need to do is find some way to assemble it all, not forgetting that we're going to need some BASIC at the front of it so that the menu selection strings can be changed to fit the actual help files available. We want to use PAL (the Transactor's default assembler), but PAL doesn't have a good way of handling this scattered code in one assembly. The TSDS assembler (Total Software Development System by Kevin Pickrell), on the other hand, has the "$= " pseudo-op, which is specifically designed to allow you to assemble discontiguous blocks of code in one piece. TSDS also lets you bring in binary data (our BASIC front end, in this case) from disk, as though it had been coded with .BYTE statements. After a bit of fooling around, the thing is done.

For publication, though, the program really should be in PAL—after all, that's the assembler most of our readers are using. PAL has the .BAS pseudo-op, which lets you incorporate BASIC code directly into your program, using SYS <label> to invoke the machine code. But it doesn't let you assemble those discontiguous blocks in one piece. The answer this time is to assemble in two pieces, appending the second piece of object to the first in an assembly to disk. A couple of chunks of the program need to be accessed both before they're copied into D-block RAM and after (the VECSET routine, the help utility video preferences and some other data) which means some fancy footwork with labels, but it turns out to be possible.

To create the HELP program, you can either assemble the source code in listing 1 or type in the BASIC data loader in listing 2. If you choose to assemble the source, you'll have to use PAL 64 or else face a very tricky porting job. Otherwise, type in listing 2 and run it. Either way, this will create the file called "help", on disk, which is the final program, and all you'll need except for the sequential text files themselves.

Run the HELP program, and the help is thereafter available at any time (at least until the next RUN-STOP/RESTORE). To get the help, press CTRL-<left-arrow>. Your current screen will be replaced by a screen containing the menu choices, which you select by letter. (Exception: if you have more than one file open, the help utility will refuse to work, and will just do a brief series of screen flashes instead.) Each menu selection corresponds to a disk file, which will then be printed as described earlier (if the file is not on your current disk, you'll get a garbage character and a space bar prompt— we didn't have room for fancy error checking). After you've finished looking at help files, you'll be returned to wherever it was you came from, which might be direct mode or might be a running program. Your screen colours and so on will be the way they were before. If you want to disable the help for any reason, press SHIFT-CTRL-<left arrow>; you'll get a long series of screen flashes, and the special keypress will no longer work. You'll have lost exactly 17 bytes of open RAM in the process. To re-enable the help, you'll have to load and run the original program file all over again.

To install your sequential help files, first rename them as "help-a", "help-b" and so on. Then load the help program, and write your menu selections into the strings in the DATA statements between lines 50 and 69 (you can have up to 20 selections, hence 20 different help files). Make sure that you do not alter the length of the strings or, for that matter, any line in the program, after assembly—this is very important. When the program encounters a DATA statement beginning with a blank, it assumes that there are no more menu selections to come. When you've entered all of these strings into the DATA statements, enter RUN 100 to install them, and resave the program (before doing a regular RUN) to make your changes permanent. When the program is subsequently run, the new selections will be displayed in the menu. Besides changing the names of menu items in the help utility, you can control the border, background and character colours by changing the POKEs in lines 230 through 250.

If you want to set the location of the 17 bytes of link code that go in normal RAM, change the 5-digit address in line 10 (assigned there to the variable A). If this address is zero, the help utility puts the link code at the top of BASIC. If it is non-zero, the link code goes to the specified address. If you do change this number, just make sure you don't change the length of the line (I.E. the number must always be 5 digits long).

Note for PAL users: After you assemble, the statement SYS "INIT".A in line 11 will have been changed to SYS3298:;A (the actual number may vary depending on just how you type the BASIC portion in). The colon after the number is an error, of course; it results from an oversight in PAL. Change the colon to a space (don't just delete it!) before you save the program.

One more thing before we close. There's undoubtedly more you can do with the D-block RAM than just print sequential files. It can sometimes be really useful to be able to do something while in the midst of another program, in order to change the program's behaviour in some way. (The HELP utility even works from many commercial programs.) A simple example is changing screen colours; a more difficult thing would be adding features to a program. All that's required to adapt this program for your own purposes is to modify the central part of the help routine, and replace the file printing routines, the string storage and the stack-page subroutines with code of your own devising. The other parts of the program should not need to be changed. Please let us know if you come up with anything interesting that you want to share with other Transactor readers.

Help! BASIC Loader

| PM  | 10 rem data generator for "help"  |
| FM  | 20 rem creates file on drive 0    |
| FJ  | 30 cs = 0                        |
| JJ  | 40 for i = 2049 to 4718:read a   |
| NH  | 50 cs = cs + a.next i            |
| BO  | 60 if cs(>225125 then print "data error": end |
| OF  | 70 restore                       |
| FD  | 80 open 1,8,15,"io":open 2,8,1,"0:help! |
| BM  | 90 input#1,e,s,t;if e then print "disk error": goto 150 |
| IO  | 100 print#2,chr$1;chr$8;       |
| PN  | 110 for i = 2049 to 4718:read a |

The Transactor 45
May 1987: Volume 7, Issue 06
GH 4140 data 1, 32, 156, 1, 166, 144, 208, 37
OJ 4150 data 201, 13, 240, 5, 206, 228, 221, 208
OA 4160 data 237, 206, 227, 221, 208, 227, 32, 131
BF 4170 data 221, 169, 183, 160, 221, 32, 135, 221
CH 4180 data 32, 138, 1, 192, 2, 240, 16, 201
FJ 4190 data 32, 208, 245, 240, 194, 32, 131, 221
GH 4200 data 32, 138, 1, 201, 32, 208, 249, 76
IK 4210 data 120, 1, 169, 205, 160, 221, 133, 34
NK 4220 data 132, 35, 160, 0, 177, 34, 240, 6
JI 4230 data 32, 156, 1, 200, 208, 246, 96, 13
OB 4240 data 8, 147, 200, 69, 76, 80, 33, 32
KD 4250 data 200, 69, 76, 80, 33, 13, 18
CI 4260 data 211, 69, 76, 69, 67, 84, 32, 65
DB 4270 data 32, 84, 79, 80, 73, 67, 18, 32
NC 4280 data 40, 204, 207, 199, 207, 32, 75, 69
ME 4290 data 89, 32, 84, 79, 32, 69, 88, 73
KO 4300 data 84, 41, 146, 0, 13, 18, 211, 208
CD 4310 data 193, 195, 197, 32, 84, 79, 32, 67
MO 4320 data 79, 78, 84, 73, 78, 85, 69, 146
IP 4330 data 0, 0, 0, 2, 2

Help! PAL Source

PL 10 open 1,8,15, "s0:t-help": close 1
GJ 11 open 2,8,1, "0:--help":
NN 12 sys 700
OB 13 .opt 02
HI 14 .bas
AB 15 rem the transactor help utility
FA 16 rem nick sullivan and chris zamara
JG 17 rem october 1986
IG 18 rem (c) 1986 the transactor
CJ 19 rem okay to copy, not to sell
II 20 :
HN 21 rem do not alter the length of any
DB 22 rem line after program is assembled
LI 23 :
LP 24 a=00000 rem 5 digit link base addr
PA 25 sys "init": a=crl
FM 26 goto 340
PI 27 :
JO 50 data "Getting Started"
DN 51 data "Printing Your Text"
HD 52 data "Avoiding Spelling Errors"
IE 53 data "Getting Finished"
KC 54 data 
KP 55 data "The first four data"
MM 56 data lines above are a sample
AF 57 data "menu that will be shown"
MC 58 data "when help is invoked.
LO 59 data "Up to 20 menu items may"
JN 60 data "be used. For each one,
MO 61 data "there must be a matching
GK 62 file on disk, with a
HP 63 data "filename of the form:
AN 64 data help-?
FC 65 data where the ? represents
ND 66 data an alphabetical character.
BN 67 data "The above menu items"
OI 68 data "require filenames help--a"
GO 69 data "through help-d.
KL 70 :
HM 100 si = 25 + 1: rem 25 chars per menu item
1571 RAM Disk Copy

GULP.COPY is a short BASIC 7.0 program with a machine language loader for the C-128 which uses burst mode read and write routines along with the 512k memory module for copying disks on the 1571 drive. The program will make an exact duplicate of your single or double sided GCR disks (either normal Commodore DOS or CP/M) in one gulp, with no bothersome disk swaps. The program is relatively fast (about 6 minutes for a single sided disk or 12 minutes for a double sided disk) and very easy to use. Just follow the prompts on the screen.

Because the copy is exact, there is no way to distinguish between the original (source) and copy (target) during the copy process (the i.d. code is also duplicated). I recommend, therefore, that you cover the write protect notch on the source disk to prevent disaster from striking if you accidentally mix up the original and copy disks during the copy process.

While 6 minutes may not seem particularly fast (some 1541 disk copy programs can do it in much less), the program is very simple (therefore reliable) and does not resort to sophisticated reprogramming of the disk drive. (In addition, with most 1541 fast copy programs, the screen is blanked out and all extra devices must be removed from the serial port. Neither is necessary for this program). With a full disk, 1571 GULP.COPY is more than twice as fast as the 1571 DOS shell “copy a disk” utility. (The DOS shell routine only copies allocated blocks on the disk while GULP.COPY will copy everything. Even so, GULP.COPY will be faster for all but an almost empty disk!) It is also far more versatile. Because it copies everything on the disk, GULP.COPY can also be used to copy C-128 CP/M disks (GCR format only – but it can be easily modified to copy MFM disks also) and disks with unallocated random files (I admit to being sloppy because I don’t always allocate the blocks for my random files) neither of which can be copied with the DOS shell program.

Although the target disk is formatted on both sides, only one side is used if the source disk was single sided thus maintaining full compatibility with the 1541 drive. (The DOS shell writes everything to a 1571 double sided disk which may not always work in 1541 mode because a file might be stored partly or completely on the inaccessible flip side).

The six minutes for a single sided disk breaks down approximately as follows:

40 seconds to read the entire disk
40 seconds to format the new disk
and the rest to write the new disk.

GULP.COPY uses 1571 burst mode to read and write, thus it will not work with other drives, such as the 1541. For a full description of the burst mode transfer protocol, see the three part series “A Layman’s Guide to Burst Mode” which appeared in TPUG magazine May to July, 1986. GULP.COPY also uses the 512k memory expander module as a RAM disk. The code to access the expansion is written in machine language and located at address $0BB8 of the cassette buffer along with the burst mode code. The ML routine is used because of a minor flaw in BASIC 7.0’s STASH, FETCH and SWAP commands which are normally used to access the RAM disk. These commands cannot “see” the RAM beneath the I/O block at $D000 to $DFFF of bank 15 because the direct memory access chip (DMA) is memory mapped into a portion of this area at $DF00 and the commands do not switch out this block. This presents a problem for programs such as GULP.COPY which uses all of Bank 0 RAM, including the part hidden by the I/O block. Fortunately, the DMA chip registers can be programmed directly to allow it to access other RAM configurations, including the RAM under the I/O block.

The DMA registers are outlined in table 1. If bit 4 of the command register is off, the DMA chip will use the configuration specified by the memory management unit (MMU) configuration register at $FF00 as the source or target of the DMA. Bit 5 of the command register is the execute flag. This must be on for any type of DMA to take place. If both bit 4 and bit 7 are on, DMA takes place as soon as the command register is written to. If only bit 7 is on, DMA takes place when the MMU configuration register at $FF00 is written to. Bits 0 and 1 of the command register determine the operation to be performed. The other registers and their typical values are outlined in the table.

Table 1:

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Bit*</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DF00</td>
<td>Status (read only):</td>
<td>7</td>
<td>interrupt pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>transfer complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>block error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>128k total size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>512k total size</td>
</tr>
<tr>
<td>$DF01</td>
<td>Command:</td>
<td>7</td>
<td>execute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reload addr. regs with same nos. used last time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>decode MMU configuration at $FF00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1, 0</td>
<td>transfer C-128 &gt; RAM disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>transfer RAM disk &gt; C-128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>swap C-128 &amp; RAM disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>verify C-128 &amp; RAM disk</td>
</tr>
<tr>
<td>$DF02</td>
<td>C-128 address, lo byte</td>
<td></td>
<td>value ranges from 0 to 8 for 512k or 0-1 for 128k</td>
</tr>
<tr>
<td>$DF03</td>
<td>C-128 address, hi byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DF04</td>
<td>RAM disk address, lo byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DF05</td>
<td>RAM disk address, hi byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DF06</td>
<td>RAM disk bank</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$DF07</td>
<td>Transfer length, lo byte</td>
<td>7</td>
<td>enable interrupts</td>
</tr>
<tr>
<td>$DF08</td>
<td>Transfer length, hi byte</td>
<td>6</td>
<td>interrupt at end of transfer</td>
</tr>
<tr>
<td>$DF09</td>
<td>Interrupt masks</td>
<td>5</td>
<td>interrupt on error</td>
</tr>
<tr>
<td>$DF0A</td>
<td>Address control</td>
<td>7</td>
<td>do not increment C-128 address during transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>do not increment RAM disk address during transfer</td>
</tr>
</tbody>
</table>
C128 Gulp Copy Program Listing

EA 1000 rem save "0:gulp copy".
AC 1010 print chr$(147); print " ** 1571 gulp copy **
II 1020 print: print "512k ram version"
DC 1030 print: print " by m. garamszeghy": print: print
FP 1040 e1 = 2816: e2 = e1 + 3: e3 = e2 + 3: rem three asm entry points
IL 1050 dim sn(35): ch = 0
PI 1060 for i = 2616 to 3038: read x: poke i, x: ch = ch + x:
    next
OO 1070 if ch <> 28600 then print " checksum error": stop
DN 1080 for i = 1 to 17: sn(i) = 21: next
LG 1090 for i = 18 to 24: sn(i) = 19: next
OG 1100 for i = 25 to 30: sn(i) = 18: next
EH 1110 for i = 31 to 35: sn(i) = 17: next
AP 1120 graphic clr
GF 1130 gosub 1360: bank15: open 15,8,15,"10":
    :open 8,8,8,"#": o = 0: o2 = 0: b = 0
MG 1140 print#15, "u1:";18,0,18;0: print#15, "b-p:";8,162:
    :get#8,i8,h8
EL 1150 si = 1: sd = 1: print#15, "u1:";8,0,42,0: if ds then
    sd = 0: si = 0
AF 1160 print: print " copying.sd + 1 sides...": print
ID 1170 close8: print#15, "u0" + chr$(4)
DM 1180 print " reading..."
PL 1190 a = 52: for i = 1 to 9: gosub1410: a = a + 21: next:
    :sys#3, e3, (o2), 128
NE 1200 a = 52: for i = 10 to 17: gosub1410: a = a + 21: next:
    :sys#3, e3, (o2) + 1, 128
GO 1210 a = 52: for i = 18 to 27: gosub1410: a = a + sn(i):
    :sys#3, (o2) + 2, 128
BP 1220 a = 52: for i = 28 to 35: gosub1410: a = a + sn(i):
    :sys#3, (o2) + 3, 128
MF 1230 if sd then sd = 0: o2 = 4: o = 35: goto 1190
KN 1240 gosub 1390: print: print " formatting..."
KJ 1250 print#15, "u0" + chr$(6) + chr$(0) + i8 + h8
GO 1260 print#15, "u0" + chr$(4): print " writing..."
    :o2 = 0: o = 0
DL 1270 sys#3, e3, (o2), 129: a = 52: for i = 1 to 9: gosub 1440:
    :a = a + 21: next
HB 1280 sys#3, e3, (o2) + 1, 129: a = 52: for i = 10 to 17:
    :gosub 1440: a = a + 21: next
LJ 1290 sys#3, e3, (o2) + 2, 129: a = 52: for i = 18 to 27:
    :gosub 1440: a = a + sn(i): next
KK 1300 sys#3, e3, (o2) + 3, 129: a = 52: for i = 28 to 35:
    :gosub 1440: a = a + sn(i): next
KM 1310 if si then si = 0: o2 = 4: o = 35: goto 1270
MJ 1320:
DN 1330 print chr$(147): print " ** done **": print#15, "10":
    :do close
ME 1340 print: input " copy another (y/n) " :ca$: if ca$ = "y"
    :then 1130: else end
KL 1350:
ML 1360 print: print " insert source disk...then press return"
BC 1370 getkey as: if as<>chr$(13) then 1370: else return
IN 1380:
MD 1390 print: print " insert target disk - then press return"
    :goto 1370
MO 1400:
MJ 1410 print#15, "u0" + chr$(64) + chr$(i + o) + chr$(0)
    + chr$(sn(i)) + chr$(i + o + 1)
MG 1420 sys e2, a, sn(i), 0: return

C128 Gulp Copy PAL Source Listing

AK 1430:
OL 1440 print#15, "u0" + chr$(66) + chr$(i + o) + chr$(0)
    + chr$(sn(i)) + chr$(i + o + 1)
KG 1450 sys e1, a, sn(i): return
IC 1460:
CH 1470 data 76, 9, 11, 76, 113, 11, 76, 182
FC 1480 data 11, 133, 251, 134, 252, 120, 169, 64
CN 1490 data 133, 254, 160, 0, 56, 32, 71, 255
JM 1500 data 173, 0, 221, 205, 0, 221, 208, 248
LB 1510 data 69, 254, 41, 64, 240, 242, 162, 63
ON 1520 data 142, 0, 255, 177, 250, 162, 0, 142
ON 1530 data 0, 255, 141, 12, 220, 165, 254, 73
DE 1540 data 64, 133, 254, 169, 8, 44, 13, 220
DH 1550 data 240, 251, 200, 208, 211, 24, 32, 71
AD 1560 data 255, 44, 13, 220, 173, 0, 221, 9
IL 1570 data 16, 141, 0, 221, 169, 8, 44, 13
DJ 1580 data 220, 240, 251, 173, 0, 221, 41, 239
HB 1590 data 241, 0, 221, 198, 252, 240, 5, 230
BG 1600 data 251, 76, 20, 11, 88, 32, 204, 255
FH 1610 data 96, 133, 251, 134, 252, 132, 250, 160
JD 1620 data 0, 120, 44, 13, 220, 92, 170, 11
BE 1630 data 32, 163, 11, 32, 163, 11, 162, 63
KE 1640 data 142, 0, 255, 145, 250, 162, 0, 142
GF 1650 data 0, 255, 200, 208, 238, 198, 252, 240
GJ 1660 data 5, 230, 251, 76, 128, 11, 88, 32
NL 1670 data 204, 255, 96, 168, 8, 44, 13, 220
BJ 1680 data 240, 251, 173, 0, 221, 73, 16, 141
IJ 1690 data 0, 221, 173, 12, 220, 96, 141, 6
EK 1700 data 223, 142, 1, 223, 169, 0, 141, 2
KJ 1710 data 223, 141, 4, 223, 141, 5, 223, 141
HL 1720 data 7, 223, 169, 52, 141, 3, 223, 169
MG 1730 data 200, 141, 8, 223, 162, 63, 142, 0
PN 1740 data 255, 162, 0, 142, 0, 255, 96
MD 2020 more2 = *
HO 2030 jsr dsfget ;get data from disk
OG 2040 ;
FP 2050 getmor = *
FA 2060 jsr dsfget ;get data from disk
GM 2070 ldx #00111111 ;flip to ram 0 and kernel
ND 2080 sty mmucon ;mmu control
MP 2090 sta (ramptr),y ;store byte in ram
BP 2100 ldx #0 ;back to normal
DM 2110 sty mmucon
IC 2120 int
PB 2130 bne getmor
CN 2140 ;
NN 2150 dec numsec ;decrease # sectors to go
HI 2160 beq back2 ;if sectors fini then return
AP 2170 ;
BM 2180 inc ramptr + 1
NI 2190 jmp more2
OA 2200 ;
KG 2210 back2 = *
KD 2220 clc
IE 2230 lsr clrc ;clear all channels
MK 2240 rts
AE 2250 ;
KI 2260 dsfget = *
JN 2270 lda #8
OF 2280 ;
DC 2290 wait3 = *
EI 2300 bit dlicr ;interrupt control register
FJ 2310 beq wait3
GI 2320 ;
DS 2330 get1 = *
DD 2340 lda d2pra ;serial port 6526 cia 2
KD 2350 eor #00010000
PP 2360 sta d2pra
OJ 2370 lda disdr ;serial data register
EO 2380 rts ;got data - return
MM 2390 ;
DP 2400 ; ** entry point #3 – initialize configuration **
PD 2410 entry3 = *
FE 2420 sta dmaenkr ;64k external ram bank (0 or 4)
ID 2430 sty dmacon ;dma controller status register
(128 or 129)
DH 2440 lda #0
OA 2450 sta dmaadr ;lsb of internal address to access
CJ 2460 sta dmal ;lsb of internal expansion ram to access
GN 2470 sta dmal+1 ;msb of expansion ram to access
NA 2480 sta dmaid ;lsb of byte count
GP 2490 lda #$34
AG 2500 sta dmaid+1 ;msb of internal address to access
KE 2510 lda #$c8
KH 2520 ldx dmaid+1 ;msb of internal address to access
UK 2530 ldx #00111111 ;set for ram 0 and kernel
JK 2540 sty mmucon ;mmu control
AK 2550 ldx #0 ;then back to normal
NB 2560 sty mmucon ;mmu control
GP 2570 rts
KI 2580 ;
KP 2590 .end

The Transactor 3rd Quarter 1987

May 1987: Volume 7, Issue #6
TEXTSCAN was developed for viewing Z80 source files. You can view in either direction, control the number of lines you scroll, turn on line numbering, search for a character string, and more.

We'd all grown accustomed to the 64, even our pet budgie! But I wanted an 80-column screen. On that I was quite adamant. So we traded in the system, printer and all, for a C128 (minus printer).

One of the things I plan to do is develop C128 and CP/M mode programs using the Z80 chip. This makes a lot of sense, but you need the tools. This could prove expensive, and there's not much around for the C128. But for under thirty bucks (Canadian) you could send away the form from the System Guide and receive the complete DRI CP/M System: source, utilities and manuals. That's a real bargain considering you now possess a complete Z80 development system. You can use CP/M mode to develop in Z80 and later port the code to the C128 environment. Porting is a separate exercise that I'll tackle when the time arises. For now I need to understand what it takes to control the hardware through Z80 code. The DRI package includes complete source code and this will provide a valuable insight into Z80 interfaces. That's one reason why I developed TEXTSCAN. I can examine an entire source file very easily, right there on the screen, and not have to print it (even if I did have a printer).

CP/M does have a built-in command called "TYPE", but it is only really adequate for cursory examination of text. It displays a screen at a time, allowing you to go forwards through the text in a sequential manner. TEXTSCAN gives you a viewport into your entire document. You can view in either direction, control the number of lines you scroll, advance rapidly through the text, turn on line numbering, and even search for a character string (great for examining source code!). And it is fast.

The only limitation is that it will only load the first 54K of a file into memory for viewing. I can't imagine someone creating files that large, but if you need to view one, then split it up into smaller parts.

Creating The Program

Assuming of course that you have received the mail-order package listed in the System Guide, then generating the program is fairly straightforward. The first step is to create a bootable CP/M disk to work with. Format a disk and use PIP to copy over CPM + SYS and CCP.COM as explained in the System Guide. Also copy over ED.COM unless you are lucky enough to have another editor. (Almost any other editor is better than ED!). Next copy Z80.LIB from the DRI Source disk, and the files, MAC.COM and HEXCOM.COM, from the DRI Additional Utilities disk.

Reboot the system from your work disk and use ED (or alternative) to type in the source of TEXTSCAN.ASM, saving it under this name. Here you cannot use the Verifier to help you, so be especially careful. In any case, double check your input for the SUMCHECK routine which begins the program. Of course you needn't enter the comments (anything beginning with ' ').

When you are satisfied with the file, run MAC TEXTSCAN. This will generate three output files with extensions - .HEX, .PRN and .SYM. Correct any assembly errors that are listed and repeat the process. You may need to run TYPE TEXTSCAN.PR to identify where some of these errors are, as the macro process may display errors which are difficult to place. If error free, then as a further check on your input, make sure that the display shows '0656' (last assembled address) followed by an '023h USE FACTOR'.

Now run HEXCOM TEXTSCAN which will convert the .HEX file into an executable program file, TEXTSCAN.COM. It should list a first address of 0100, last address of 05BB, bytes read 04BC and records written 0A.

Using Textscan

Start by running TEXTSCAN TEXTSCAN.ASM to view your source file. If all is well, the entire source file will be read into memory and the first 24 lines displayed. If the program crashes, then most likely a source error in the SUMCHECK routine itself caused the crash. Check this source again. However, if the program prints a '?' and then gracefully exits, then a bad sumcheck has been generated, meaning that there is a typo somewhere in your source. Correct any errors and reassemble your program.

With your text file displayed on the screen, you have several options, all of them clustered around the Numeric Keypad and the top row (except for the function keys). You'll find that the keys give you immediate response, i.e., you can instantly modify the action even as text is being displayed.

(ENTER) - Outputs the number of lines last defined by the (0 - 9) and (. ) keys.
(0 - 9) - Defines 1 to 10 lines for output. (0) counts as 10. Also acts as if (ENTER) was pressed.
( ) - Defines 24 lines for output (default). Also acts as if (ENTER) was pressed.
(ALT) - Toggles Double mode which allows keys (0 - 9) to define 11 to 20 lines for output.
(+ ) - Changes scroll direction upwards. If held down, will continuously scroll text in the forward direction.
(- ) - Changes scroll direction downwards. If held down, will continuously scroll text in the reverse direction.
(NO SCROLL) – Toggles Scroll Mode where text is continuously scrolled in the current direction.

(TAB) – Skips 24 lines in direction of scroll and acts as if (ENTER) was pressed. By repeatedly tabbing you can quickly advance through the text.

(LINE FEED) – Toggles Line Number mode where line numbers are displayed before each line.

(CRSR UP) – Goes to the beginning of text, sets a forward direction and then acts as if (ENTER) was pressed.

(CRSR DOWN) – Goes to the end of text, sets a reverse direction and then acts as if (ENTER) was pressed.

(HELP) – Erases the current line – top line in reverse scroll, or bottom line in forward scroll. Now you can enter a string of up to 40 characters. The line editing functions supported by CP/M will work normally. Pressing (RETURN) causes a search to be initiated in the current direction of the scroll, i.e., either to the beginning or end of file. The first match will result in a new set of lines output, starting with the matched line. If the search fails, the original line that was erased will be re-displayed.

(ESC) – Exits back to the CP/M.

Lessons Learnt

Although I had previously coded in Z80, this was my first attempt with CP/M and with a C128. So obviously some lessons were to be learnt.

The CP/M Macro Assemblers, MAC and RMAC, support all the Z80 instructions through the Z80.LIB library file. By using this file you can code all the Z80 instructions. TEXTSCAN uses this library extensively. Examine Z80.LIB to understand the syntax of the mnemonics used.

I liked the CP/M+ facility to read up to 16k at a time into a specified area using Multisector I/O. TEXTSCAN does this in a loop with a count of 3, each iteration reading up to 128 sectors (128 times 128 bytes, or 16K). Then if there is still data to be read, the loop is executed once more to read in a further 6K. On each read, register H returns the actual number of sectors read if the file is exhausted. Otherwise register H contains 0. This is not too obvious from the manuals.

I couldn’t find any method to make keys repeat in CP/M mode. Because I wanted the (+) and (-) keys to repeat, (in order to scroll), I decided to bypass the BDOS console routine and scan the keys myself. As the keys I’d chosen were C128 mode keys, I had to unravel how these keys were scanned. The result is the KSCAN routine which scans the C128 keys.

Scanning the C128 keys means using the I/O registers in the DXXX block of memory. But CP/M transient programs do not have I/O turned on in Bank 1 where they are run! Yet TEXTSCAN works even without manipulating the Configuration registers. This leads me to conclude that if you use the Z80 Input/Output instructions, then you do not need I/O switched in to access the I/O registers – a bonus for Z80 programmers!

**TEXTSCAN Z80 Source Code**

```
..TEXT SCAN' FOR CP/M+ ON THE C128
Aubrey Stanley, Nov 1986

maclib z80  ;Z80 macro library
org 100h ;Start address

;CP/M Functions
boot eq 0 ;Warm start
bdos eq 5 ;CP/M Function Vector
dcof eq 6 ;Direct console
print eq 9 ;Print string
conif eq 10 ;Read console buffer
open eq 15 ;Open file
close eq 16 ;Close file
read eq 20 ;Read sequential
sdmaf eq 26 ;Set DMA address
mscof eq 44 ;Multi-sector I/O
scrmn eq 109 ;Set console mode

;File Control Block
tcb eq 5ch
tcbr eq 12

;Console Buffer for Find string
dmbuf eq 80h
mx eq mx +1 ;Max chars
nc eq nc +1 ;Num chars
rchar eq rch +1 ;Char string

;Character equates
rawin eq 0ah
rawout eq 0dh
crsy eq 1ah
frq eq 0ah

;Print file
lxx pflags
lxh h.0
shelf eq 128
stm eq 1
fi eq 0
mki eq 0
opp eq 48
stx eq 1
mx eq 0
mcof eq 0
mssc eq 0
msof eq 0
mscof eq 0
```

The Textscan 56 May 1987: Volume 7, Issue 06
The Transacter Volume 7 Issue 06 May 1987
That Guru Does Have A Message!
Betty Clay, Arlington, Texas

Has anyone escaped the GURU? At some terribly important point, you see the requester box that says:

"Software error - task held.  
Cancel ALL disk activity.  
Select CANCEL to reset/debug."

When you press the button, the black and red alert box appears with the 'Guru Meditation' message:

"SOFTWARE FAILURE! PRESS LEFT MOUSE BUTTON TO CONTINUE"

followed by a number of many digits, such as #03000007.000057A3 or #83010009.00002463. For the average user, the only choice is to reset the computer. The messages seem totally meaningless; sometimes they even appear to be a cruel joke played on helpless users by the creators of the Amiga. Why?

There truly is meaning in those guru meditations. We can all learn what most of them mean if we are provided with the necessary information. Serious programmers can attach a terminal to the Amiga with a null modem cable, set their workbench programs to debug before they load the program, and can work on their programs through the terminal without changing the state of the Amiga. They can use the ROMWACK debugger that exists on all KickStart disks, or the GRANDWACK debugger that is part of the developers' packages. And they will pore over the ROM KERNAL MANUALS as they work. Those volumes are not written in what most of us would call English, but they contain more information than some of us ever expected Commodore to release to us, and it is there that we find the explanations of the Guru Messages, more properly called 'Alerts.'

It is not the purpose of this article to explain how to debug programs using all of the needed equipment. It's purpose is to give some meaning to the alerts, and to assure the reader that there are programmers out there to whom these messages are really useful.

First, let's break the messages into manageable pieces. Then we'll try to make sense of each piece. Take a (hypothetical) Guru Message number like:

#07060006.00008321

Break it up like this:

07 06 0006. 00008321

*07060006.00008321

The first part (07) denotes the device or library from which the error message came. There are twenty of these devices, libraries, etc., and they are listed in numerical order in the reference page that accompanies this article. If you look there, you will see that '07' indicates that the error occurred while using the DOS library. The zero at the beginning indicates that it would be possible to recover from this error, if we have the proper equipment and knowledge to do the debugging. For now, we must be content to know that if the beginning digit is zero, the error might be correctable; a beginning digit of eight or more indicates an error from which there is no hope of recovery. Thus, you might have a DOS library error that began with '07' (recoverable) or with '87' (hopeless). These are hexa-decimal digits, so you might find A, B, C, D, or E in the left column.

To put it a bit more technically, if the error is irreversible, the high bit is set. (Fatal errors also cause the entire screen to go black, while recoverable ones only push the normal screen down a bit.)

*07060006.00008321

The second piece of information (06) may be one of the six "general error codes." In this case, I chose the "I/O Error" because it seemed appropriate for a disk error, though in reality it is more likely to occur in other libraries. The six general error codes are also listed in the reference page. The one you will encounter most often is '01', insufficient memory. If the general error is not attributable to one of these six codes, there should be two zeros in this field of the error message.

*07060006.00008321

The four digits immediately to the left of the decimal point tell exactly what it was that went wrong. In our sample message, the digits are '0006', and the reference chart will show that, in the DOS Library, a code of six indicates a disk block sequence error.

You probably would not memorize these numbers, for the same number will mean different things for different libraries/
devices. The reference chart includes a list of these codes as they appear on Version 1.0 of the WACK disk, a debugging disk that was provided for the developers. Commodore is now offering this disk for sale, but the quoted price is $99 U.S. Remember that the Guru Meditations were intended to help the developers, so the explanations of the codes may not be adequate for those of us with less knowledge, but this information from the 'exec.alerts' file provides strong clues about what is going wrong. To learn much more would require learning most of the ROM KERNAL MANUAL.

*07600006.0008321*

The eight digits to the right of the decimal point give the hexadecimal notation for the address of the task (program) that reported the error. Since the Amiga is a multitasking machine, all software must be relocatable, and these locations may be different each time you load the program.

The codes in the chart have been broken into sections to make them more readable than they are in the 'Guru Meditations'. You can see the library/device number in the first two digits; the general error code in the second pair of digits is always either '01' (not enough memory) or '00' in these examples; and an attempt has been made to give at least some meaning to the four digit code at the end. Notice, also, that few of these are called fatal errors. Most of the errors in the Exec, Graphics, and Intuition Libraries are fatal. Most others are not.

Here are a few examples:

*#81000003.0000A325*

81 - a fatal error in the Exec Library
00 - not one of the general errors
0003 - an error in the library checksum
.0000A325 - of the task starting at location $A325

*#21000001.00001234*

21 - recoverable error in the Disk Resource
00 - not a general error
0001 - the unit already has a disk
.00001234 - the task starts at location $1234

*#84010005.00002345*

84 - fatal error in Intuition Library
01 - caused by lack of memory
0005 - needed to open a new window
.00002345 - called by a task at location $2345

To a skilled programmer, armed with literature and an intimate knowledge of the Amiga, this information could save hours.

There is another type of error that can bring a "Guru Meditation." These have all zeros except for the last digit or two before the decimal point. They are errors reported by the 68000 microprocessor itself. For example,

Meditation #00000002.xxxxxxxxxx

...would indicate that the microprocessor had received an illegal instruction. For most of us, it is adequate to know that if the first six or seven digits are zeros, the error is irreversible, and it was reported by the 68000 microprocessor. We can't do anything about it, but we can recognize it.

For those of us who have yet to learn C and 68000 assembly, making the corrections must wait - but it is reassuring to know that we can look at the message and say to ourselves, "Well, I've ruined my data and lost my program, but at least I know it was because the Graphics Library didn't have enough memory left to draw my text." That can be solved by adding more memory. As our knowledge grows, the other problems will become correctable, too.

Reference Chart Notes

There are a very few guru meditations that cannot be interpreted using these tables. One was reported recently by Claudio Nieder, a computer science student in Switzerland. He kept getting a guru meditation number 84000009.48454C50. Noticing that the address part bore a strong resemblance to ASCII code, he translated it:

48 45 4C 50
H E L P

Soon he received a reply from Dale Luck of Commodore-Amiga. It seems that the Amiga exec uses the word HELP as a catchword. When Intuition finds the system to be in so bad a state that the error message cannot even be displayed, Intuition places the word HELP in location 0 and then resets the system. As the system is brought back up, location 0 is checked for the word HELP, and if it is there, this message is displayed.

Carolyn Scheppner of Commodore Technical Support recently elaborated on some of the guru messages that are made by the 68000 itself. The two microprocessor codes that appear most often are 000000003 and 00000004. Code 3 (Address error) may be caused when an invalid, no-longer-valid, or zero pointer is passed to a system routine, causing an attempt to do word or longword manipulations on an odd address. Code 4 (illegal instruction) could be caused by poor coding, but it can also happen when memory containing necessary code or vectors has been overwritten.

It seems that some of the exec.alerts can even be used to diagnose hardware problems. The examples given by Ms. Scheppner were 870000008 and 87000000B, which indicate "key already free" and "key out of range". She says that these error codes could indicate problems with the keyboard or with static.
## Reference Chart for Guru Messages

### Devices, Libraries, and Resources:

<table>
<thead>
<tr>
<th>Device/Resource</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec Library</td>
<td>01</td>
</tr>
<tr>
<td>Graphics Library</td>
<td>02</td>
</tr>
<tr>
<td>Layers Library</td>
<td>03</td>
</tr>
<tr>
<td>Intuition Library</td>
<td>04</td>
</tr>
<tr>
<td>Math Library</td>
<td>05</td>
</tr>
<tr>
<td>CList Library</td>
<td>06</td>
</tr>
<tr>
<td>DOS Library</td>
<td>07</td>
</tr>
<tr>
<td>RAM Library</td>
<td>08</td>
</tr>
<tr>
<td>Icon Library</td>
<td>09</td>
</tr>
<tr>
<td>Audio Device</td>
<td>10</td>
</tr>
<tr>
<td>Console Library</td>
<td>11</td>
</tr>
<tr>
<td>GamePort Device</td>
<td>12</td>
</tr>
<tr>
<td>Keyboard Device</td>
<td>13</td>
</tr>
<tr>
<td>TrackDisk Device</td>
<td>14</td>
</tr>
<tr>
<td>Timer Device</td>
<td>15</td>
</tr>
<tr>
<td>CIA Resource</td>
<td>20</td>
</tr>
<tr>
<td>Disk Resource</td>
<td>21</td>
</tr>
<tr>
<td>Misc. Resource</td>
<td>22</td>
</tr>
<tr>
<td>BootStrap</td>
<td>30</td>
</tr>
<tr>
<td>Workbench</td>
<td>31</td>
</tr>
</tbody>
</table>

### General Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Insufficient memory</td>
</tr>
<tr>
<td>02</td>
<td>MakeLibrary Error</td>
</tr>
<tr>
<td>03</td>
<td>OpenLibrary Error</td>
</tr>
<tr>
<td>04</td>
<td>OpenDevice Error</td>
</tr>
<tr>
<td>05</td>
<td>OpenResource Error</td>
</tr>
<tr>
<td>06</td>
<td>I/O Error</td>
</tr>
</tbody>
</table>

### Specific Error Codes

**EXEC LIBRARY: 01 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExctpVect</td>
<td>81 00 0001</td>
</tr>
<tr>
<td>BaseChkSum</td>
<td>81 00 0002</td>
</tr>
<tr>
<td>LibChkSum</td>
<td>81 00 0003</td>
</tr>
<tr>
<td>LibMem</td>
<td>81 00 0004</td>
</tr>
<tr>
<td>MemCorrupt</td>
<td>81 00 0005</td>
</tr>
<tr>
<td>IntrMem</td>
<td>81 00 0006</td>
</tr>
</tbody>
</table>

**GRAPHICS LIBRARY: 02 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopDisplay</td>
<td>82 01 0001</td>
</tr>
<tr>
<td>CopInstr</td>
<td>82 01 0002</td>
</tr>
<tr>
<td>CopListOver</td>
<td>82 00 0003</td>
</tr>
<tr>
<td>CopListOver</td>
<td>82 00 0004</td>
</tr>
<tr>
<td>CopListHead</td>
<td>82 00 0005</td>
</tr>
<tr>
<td>LongFrame</td>
<td>82 01 0006</td>
</tr>
<tr>
<td>ShortFrame</td>
<td>82 01 0007</td>
</tr>
<tr>
<td>FloodFill</td>
<td>82 00 0008</td>
</tr>
<tr>
<td>TextTmpRas</td>
<td>02 01 0009</td>
</tr>
<tr>
<td>BltBltMap</td>
<td>82 01 000A</td>
</tr>
</tbody>
</table>

**LAYERS LIBRARY: 03 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GadgetType</td>
<td>84 00 0001</td>
</tr>
<tr>
<td>GadgetType</td>
<td>04 00 0001</td>
</tr>
<tr>
<td>CreatePort</td>
<td>84 01 0002</td>
</tr>
<tr>
<td>ItemAlloc</td>
<td>84 01 0003</td>
</tr>
<tr>
<td>SubAlloc</td>
<td>84 01 0004</td>
</tr>
<tr>
<td>PlaneAlloc</td>
<td>84 01 0005</td>
</tr>
<tr>
<td>ItemBoxTop</td>
<td>84 00 0006</td>
</tr>
<tr>
<td>OpenScreen</td>
<td>84 01 0007</td>
</tr>
<tr>
<td>OpenScrRast</td>
<td>84 01 0008</td>
</tr>
<tr>
<td>SysScrType</td>
<td>84 00 0009</td>
</tr>
<tr>
<td>AddSWGadget</td>
<td>84 01 000A</td>
</tr>
<tr>
<td>OpenWindow</td>
<td>84 01 000B</td>
</tr>
<tr>
<td>BadState</td>
<td>84 00 000C</td>
</tr>
<tr>
<td>BadMessage</td>
<td>84 00 000D</td>
</tr>
<tr>
<td>WeirdEcho</td>
<td>84 00 000E</td>
</tr>
<tr>
<td>NoConsole</td>
<td>84 00 000F</td>
</tr>
</tbody>
</table>

**INTUITION LIBRARY: 04 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartMem</td>
<td>07 01 0001</td>
</tr>
<tr>
<td>EndTask</td>
<td>07 00 0002</td>
</tr>
<tr>
<td>QPktFail</td>
<td>07 00 0003</td>
</tr>
<tr>
<td>AsyncPkt</td>
<td>07 00 0004</td>
</tr>
<tr>
<td>FreeVec</td>
<td>07 00 0005</td>
</tr>
<tr>
<td>DiskBltSeq</td>
<td>07 00 0006</td>
</tr>
<tr>
<td>BitMap</td>
<td>07 00 0007</td>
</tr>
<tr>
<td>KeyFree</td>
<td>07 00 0008</td>
</tr>
<tr>
<td>KeyRange</td>
<td>07 00 0009</td>
</tr>
<tr>
<td>DiskError</td>
<td>07 00 000A</td>
</tr>
</tbody>
</table>

**TRACKDISK DEVICE: 14 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDCalibSeek</td>
<td>14 00 0001</td>
</tr>
<tr>
<td>TDDelay</td>
<td>14 00 0002</td>
</tr>
</tbody>
</table>

**DISK RESOURCE: 21 00 0000**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRRHasDisk</td>
<td>21 00 0001</td>
</tr>
<tr>
<td>DRIntNoAct</td>
<td>21 00 0002</td>
</tr>
</tbody>
</table>

**WORKBENCH: 31 00 0000**

**CPU Error Codes:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>Bus Error</td>
</tr>
<tr>
<td>03</td>
<td>Address Error</td>
</tr>
<tr>
<td>04</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>05</td>
<td>Division by zero</td>
</tr>
<tr>
<td>06</td>
<td>CHK instruction (result fell outside designated bounds)</td>
</tr>
<tr>
<td>07</td>
<td>Trap overflow</td>
</tr>
<tr>
<td>08</td>
<td>Privilege violation</td>
</tr>
<tr>
<td>09</td>
<td>Instruction trace</td>
</tr>
<tr>
<td>0A</td>
<td>Line A Emulation – 68000 encountered an instruction word with values between A000–AFFF</td>
</tr>
<tr>
<td>0B</td>
<td>Line F Emulation – 68000 encountered an instruction word with values between F000–FFFF</td>
</tr>
</tbody>
</table>

**TRAP instructions – operating system call instructions**
The Transactor Amiga Structure Browser
by Chris Zamara and Nick Sullivan

- Your guided tour through the system

With this program and the information given in this article, you will gain knowledge about your Amiga that you yearned for, but couldn't find. This knowledge will make you a better programmer, almost certainly making you more prosperous than you would have been otherwise.

With all of your extra money, you can buy more expensive computer hardware and advance yourself even further. You will be successful, and that success will give you greater self-confidence. People will turn to you for advice and you will help out many others who would like to be as clever and successful as yourself. By helping these people, not only will you gain deep personal fulfillment, but you will become quite popular and well-liked, which certainly can't hurt your sex life. Since so many people will be looking up to you and following your advice, you will wield quite a bit of power in the computer community and in your business dealings in general. By feeling good about yourself, you will certainly get into better physical shape and will definitely live longer - maybe up to three hundred years!

In short, reading this article and typing in the program at the end will give you knowledge, power, success and prosperity. It will improve your sex life and give you greater self esteem and a feeling of usefulness. You will be able to buy lots of expensive computer hardware, and fast cars as well if you wish. You will live longer. But the choice is yours - read this article and make those changes in your life, or skip past it and possibly never realize the enormous potential that you know is inside of you.

Okay, maybe you won't get all of those things but, if you want to learn more about the Amiga's system structures, or you'd like an easy way to look at vital data about programs while they execute, you'll make good use of the Transactor Structure Browser.

If you read our Amiga programming article in the last issue, you learned how important the various system structures are, since they are the key to finding out about all entities known to the system, like tasks, screens, gadgets, and the like. Structures in the Amiga are the equivalent of a memory map on a single-tasking machine, since there are no fixed memory locations on the Amiga.

The structure has a direct C-language implementation, but we can speak of a structure template (the definition of the structure) independently of a language. A structure is just the particular way that some data are grouped. For example, a structure called 'X' might be defined as a word, followed by a pointer to another structure 'X', followed by a pointer to a structure 'Y', followed by a long word. By knowing a structure's template and where in memory such a structure is stored, it is a simple matter for the system or an application program to extract its data.

What kind of information can we find in various structures throughout the system's memory? Well, some interesting things are to be found, for example, in the Intuition-managed structures. With the program presented here, you'll be able to look at Screen, Window, and Gadget structures, among others. Information about any program in the system can be gained by looking at these structures, since everybody's windows, screens and gadgets are maintained in a kind of network where they can be accessed via pointers from other Screen, Window and Gadget structures.

A pointer to the first Screen is all that is needed to find everything else, and that is found in the "IntuitionBase" structure, which, in turn, is found when the Intuition library is opened by a call to the OpenLibrary() function. (OpenLibrary() is in the Exec library, a pointer to which is found in memory location 000004, the only fixed location in the entire Amiga memory map. C-language startup code opens the Exec library.) All programs that use Intuition have their information in these structures, so through them you can learn things about almost any program currently running in the system.

A screen structure contains everything Intuition needs to know about a screen, like its size, title, number of bit-planes, a pointer to its gadgets, flags describing what kind of screen it is, the colours it's rendered in, etc. The screen structure points to a window structure for the first window on the screen. A window structure has a pointer that can point to another window structure, so that all windows in a screen are linked together. If there is more than one screen in the system (unless you have an application running that opens its own screen, the only screen will be the Workbench screen), there will be a pointer in the first screen structure to the next screen; any number of screens can be linked together in this way.

In a window structure you'll find the usual size and colour information, along with an IDCMP Flags variable describing what messages Intuition is getting from that window. A pointer to the first gadget in a linked list of gadgets can also be found in the window structure. A gadget structure totally describes a gadget (a simple way to get input from the user through the mouse) - its position, size, type, imagery, etc. The gadget structure is set up by an application before the structure is submitted to Intuition, so the values within represent those that the programmer actually put into his code - in other words, a peek into a program's gadget structure can give some insight into how a program was written.
Other structures of interest that are not implemented in Version 1.0 of Structure Browser: Menu, Image, Requester, View, ViewPort, Layer, Interrupt, MemChunk, MsgPort, Message, Task, VSprite, AnimOb. (Many of these structures, like Task and Interrupt, are difficult to monitor since the data within are constantly changing.)

The key to finding the memory location of any structure is the Library Base structures. Every shared library ("graphics.library", "intuition.library", "layers.library", to name a few) has a library base structure that contains pointers to structures of interest to routines in that library. As mentioned above, the structures form a kind of network, since you can reach structures through pointers in other structures. In this network can be found nearly every item of data that the system cares about anything: a direct link to the system’s state of mind at any given moment. If we know the definitions for the system structures (there are over a hundred of them altogether), we can snip around and discover anything we wish to know about what the system knows. And in the Amiga, the system knows a lot, because application software has to go through the system for just about anything it wants to do.

The Structure Browser Program

The Structure Browser (SB from now on) makes it easy to snip through many of the structures in the system. You start with a list of the library base structures; in the current version of the program, only IntuitionBase is available. To display the contents of the library base structure, just move the pointer to the library base name on the screen and press the left mouse button. The library base structure will immediately be displayed, showing each structure member’s name (as defined in the standard Amiga header files) and its data type. In the case of the IntuitionBase structure, all members are either important structures in their own right, or else pointers to such structures; to display any of these, just click on its name as usual. This process can be repeated again in the new structure, to bring you to other structures of interest.

You can always get to the structure you came from by clicking on the “Previous Level” gadget at the bottom left of the window. You can trace your steps as far back as you want, since the program works recursively (the number of levels deep you are is displayed in the heading at the top of the window). Structures that contain more members than will fit on the window show the first 16 members and display a gadget that says “(MORE)”. You can go to the next page of structure members by clicking on (MORE), and go to the previous page by clicking on the “Previous Page” gadget.

By going from structure to structure in this manner, it is easy to view information about programs that would be difficult to find out otherwise. For example – what kind of gadgets are being used in that neat commercial program? Are they gadgets at all? Did the programmer use a requester or a window for that prompt? Just by mousing around with SB, you can find it all out!

The data type for each structure member is shown in the same way that the members are declared in the C header files (the appropriate header file must be compiled – using the #include statement – with any C program that wants to refer to a system structure).

These data types are standard in Amiga code, because they are ‘typedefs’ that are set up in the header file “exec/types.h”. The basic types are:

- BYTE - signed 8-bit value (char)
- ubyte - unsigned 8-bit value (unsigned char)
- short - signed 16-bit value (short)
- ushort - unsigned 16-bit value (unsigned short)
- long - signed 32-bit value (long)
- ulong - unsigned 32-bit value (unsigned long)

Pointers to any of the above are denoted as in standard C syntax, with an asterisk (*). For example, a pointer to a short would be denoted by short * in the type field. Structure members that are pointers to other structures also use the C syntax; a pointer to a 'Window' structure, for instance, is denoted by 'struct Window *'. A structure member that is an actual structure, not a pointer to a structure, has no asterisk; SB does not display an address for such a member, since it is contained within the structure being viewed, whose address is already shown.

Structure members that represent not pointers to other structures, but actual data that may be of interest, are displayed in various ways depending on their nature. These data can be simple numbers, like those giving a window’s LeftEdge, TopEdge, Width and Height. The item of interest might be a byte, word or longword containing flags of some sort, like the flags indicating a window or gadget type. Or the data you’re concerned with might just be an area of memory that contains a table of sorts or describes a graphic image.

Simple data values, like LeftEdge, etc. are shown in black on the SB window. Such structure members cannot be used to bring up any further information by pointing on them and clicking – nothing will happen if you do so. They are just what they appear to be: a number for you to see. The values in black are often just what you want to find out after getting to a structure of interest. When one of these values is used for containing flags of some sort, it is treated differently: you can click on it to get a list of what flags are set. An example might be a “smart refresh” window with sizing, close and drag system gadgets. The “Flags” member of such a window’s structure, when selected, would show:

- WINDOWDRAG
- WINDOWSIZING
- WINDOWCLOSE
- SMART_REFRESH

(Like the member names, the flag names are those defined in the standard header files used for C or assembler program development.)

Another form of specialized output in SB is the hex dump. When a structure member is an array of values or a pointer to an area of memory like a bit-plane or image data, clicking on the member name causes a hex dump of the data to be displayed. If the data type is specified in the structure template, the data size is taken into account in the hex dump – the hex values are shown as either bytes, words, or long words.)
A Sample Tour

On paper, the process may sound rather involved, but using SB is a breeze. Here's an example of how you could find out the type of gadget used by a program that is currently running in the system.

Let's use as an example the Structure Browser program itself - we'll take a look at the structure for the program's "Previous Level" gadget.

First, we have to run SB. That's not hard: type 'sb' from the CLI (or 'run sb' if you want to keep the CLI available for launching other tasks while SB is running). SB will bring up a window that is the full width of the screen, but isn't full height, leaving a bit of the WorkBench screen visible at the top. On the window you will be prompted to select a library structure. Since only one choice is available in this version of the program, the decision isn't difficult: select "Intuition" by pointing with the mouse and clicking the left button. After clicking, a list of the "IntuitionBase" structure members will be displayed, with their types and values shown on the right. All members are either whole embedded structures or else structure pointers. The top two are parenthesized, meaning that those structure types are not available in this version of the program. To keep the magazine listing reasonably short, we only implemented a few key structure types; references to data types not handled by the browser are parenthesized, and clicking on them has no effect.

Now, on with our travels to the gadget structure of interest. Since the program we're interested in has a window on the Workbench screen, we must get to that screen's structure first. We can get there by clicking on the "ActiveScreen" member in the IntuitionBase structure, since the active screen - the one we're currently working on - is the WorkBench screen. One click, and up comes the first page of the structure members in the Workbench screen structure. You can tell you've got the right screen by looking at the member called "Title"; to the right it should say "Workbench Screen". To see more screen members, you can click on the "(MORE)" gadget that has appeared at the bottom of the window, and then click on "Previous Page" to get back. The second member of the screen structure is called "FirstWindow", and it points to the first window structure in a linked list containing all windows on the screen.

Click on "FirstWindow" and a window structure will be displayed. More than likely, this is the window we're looking for, the window belonging to SB itself. The "Title" member will tell you what window you're looking at, since it will say exactly what is on the title bar of the corresponding window. If the window isn't the one you're looking for, you can get to the next window structure in the list by clicking on the first member of the structure, the one called "NextWindow". The structure for the next window will then be displayed - you can keep chaining through all the windows in the screen in this way. Stop when you get to the Structure Browser window (check "Title"). You'll also notice other items of interest in the structure, like the window's dimensions and the flags set for the window. Try clicking on the "Flags" member to show the window flags. Even if you're not familiar with Intuition programming, the names of the flags should suggest their purpose.

Now that we've found the window structure we want, let's look for a pointer to the gadget list attached to the window. There's nothing like that on the first page, so click the "(MORE)" gadget to see more window members. About half way down is the member "FirstGadget", which points to a gadget structure (its type is 'struct Gadget *', meaning a pointer to a Gadget structure). Click there, and up comes the first in a linked list of gadget structures for that window. By looking at the LeftEdge and TopEdge variables, you can see the position of the gadget on the screen. To go to the next gadget structure in the list, just click on the first member, "NextGadget". You can chain through the program's entire gadget list this way, stopping when you wish to examine the flag variables "Flags", "Activation", or "GadgetType". GadgetType will tell you if the gadget is a BOOLGADGET (boolean), PROPAGADGET (proportional), or STRAGADGET (string).

Eventually, we get to a gadget with its TopEdge at -12, and its LeftEdge at 10. Since the GRELBOTTOM flag is set in the Flags variable, we know the TopEdge value indicates that the top edge of the gadget is 12 pixels above the window's bottom border. That's the "previous level" gadget that we're looking for. We can examine the gadget structure to see exactly how that gadget was set up in the Structure Browser program. (Using the same imagery as someone else's gadget in your programs might be considered a violation of copyright, but the current version of SB doesn't support Image structures anyway so there'll be no "Don't try this at home, kids!" warning here.)

Structure Browser doesn't support all the system structures as it stands now - there are just too many and the program would fill the entire magazine. Rather, we have put in some of the more interesting Intuition structures and we plan on adding more for version 2.0 of SB, which will be found on The Transactor's first Amiga disk, and will be uploaded to CompuServe's AmigaForum. The program is designed so that it is easy to add new structures as you wish - if you have access to the "include" files for development (or the listings of them in the ROM Kernal manual), it should be easy for you to add whatever structures you're interested in to the program. Information about doing this is given later in the article. SB is public domain, so you're free to use and modify it as you wish; if you do add new structures, though, please use the standard Amiga member names and types exactly as they appear in the include files. We plan to see how far the SB concept of system structure browsing can be taken - a few ideas for enhancements are given later in this article.

Applications for SB

By now you've probably seen that SB is a great tool for learning about the Amiga's internal data structures. But there are actually quite a few good reasons to have SB around on your most-used disk - maybe even in the C directory of your usual Workbench or development disk.

For one thing, SB makes a very handy reference to the structure templates themselves. It's easier to bring up SB and click to a structure than to look up that structure definition in the ROM Kernal manual. And if you don't have the manual, it's easier than
looking through lots of include files. If you don’t have the include files, it’s the only way!

SB can also be used as a debugging tool: if your program is acting strangely, you can peek at Intuition’s view of your windows, gadgets, etc., to make sure they are properly set up and linked together the way you intended. You can look at these things at any time during your program’s execution to see what happens when certain actions are taken; kind of like a trace, but only pertaining to the program’s interaction with the operating system.

Another benefit of SB is that you can use it to learn programming techniques by looking at how other applications have set up their structures to achieve certain effects. By investigating the structures of known system entities, you can see what effect certain flags and variables have on their behaviour. If you’re curious about the tricks that some program is using, just take a browse through its gadgets and windows. For instance, did you know that Workbench uses a borderless, backdrop window that lies on the Workbench screen just below the screen title? We didn’t either, until we used SB to look at the Workbench window structure. You can get some good ideas from other people’s use of Intuition’s resources.

**Future Versions of SB**

Our main goal for SB is to eventually have it encompass nearly all system structures and flags. The program can be expected to grow in size considerably, but it will be worth it for a total map of the system’s inner thoughts. But besides expanding in scope, there could also be more flexibility in the ways that lowest-level data types are displayed. For example, graphic data could be displayed not as a hex-dump, but as the actual image that it defines. Some values, like the mouse’s X and Y coordinates that are found in a window structure, could be updated frequently to give current readings. The hex dump should also display ASCII equivalents of the bytes. Perhaps a less immediately-implemented feature will be the saving of image and BitMap data to disk as IFF-format files. There should be a way to save any structure and its current values to a file as well, perhaps as C or assembler code for easy inclusion in your own programs. If anyone has any good ideas, let us know. Or better yet, implement them – we don’t want all the glory for ourselves! (read: we don’t want to do all the work ourselves!)

**Program Listing Notes**

The C source for SB is broken into several files, each with its own purpose. The mainline and general functions are in the file “sb.c”. The Intuition calls and all functions dealing with the program’s user interface is in the file “sbio.c”. The other files contain the individual routines to handle different structure types. “sbwindow.c”, “sbscreen.c”, and “sbgadget.c” contain the functions that handle the structures their names suggest. “sbgfx.c” handles the graphics library-related structures, RastPort and BitMap. In addition, a header file, “sb.h”, contains various #define statements and a structure definition – it should be stored in a subdirectory called “header”. All of these separate files can be compiled with Manx Atec C using “make” and the makefile provided. The makefile takes advantage of Manx’s pre-compiled header file capability to speed up compilation by not having to re-compile “intuition.h” in each of the six source files. If you are using this method, you can delete the #include at the top of each source file as indicated in the comment. For users of Lattice C, just compile all six files and link them normally – don’t worry about the makefile.

Since a requirement of this program was that it didn’t use up too many valuable magazine pages, we had to make a few compromises in the style department. For one thing, the program has far too few comments – we hope to clear up main points regarding the program’s operation in this article. There are also very few blank lines that might have made the program easier to look at. Function parameter declarations are put on a single line, and on the same line as the function declaration itself, if possible. The code isn’t too unreadable, though, and the disk/Compuserve version of SB (V2.0) will be much prettier.

When entering the program, take extra care in entering the the ‘structdata’ arrays in each structure print function. The first character of each member name is either a space, minus, or left parenthesis; it is important to use the correct one. Also, if you use the wrong data size constant (like putting in INTSIZE where it should be PTRSIZE), you will get bad results when you try to display that structure, or worse yet, a software error.

**Program Notes**

SB is quite useful as it stands, but you may want to modify it to incorporate some other structures that you’re interested in. If you do, here is some explanation of the program to help you out. The specifics are up to you; have a ball, hackers!

The principle behind SB is very simple. A specialized function exists for each kind of structure. If a structure contains more members than will fit on one page, there is a function for each subsequent page. The function is passed a pointer to the structure (and an offset, in the case of multiple-page structures), and it prints the structure’s members, types and values, then waits for input. (The actual output and input is handled by functions in “sbio.c”, but more on that later.) Depending on the member selected by the user, an appropriate routine is called to print that structure’s members and again wait for input. It may be that a routine calls itself, as when chaining through a linked list of like structures.

In any case, each selection brings the program one level of nesting deeper, and the only way to go up a level is by the user clicking the “Previous Level” or “Previous Page” gadget, causing the return from a function. (Since each extra level you visit represents an extra level of function call nesting, you face a theoretical possibility of stack overflow if you go too deep. Even with the system default stack of 4000 bytes, though, we estimate you’d have to go well over 100 levels deep before there was any danger of this happening.)

Besides functions that handle specific structures, there are the more general output functions that display hex dumps or flags – these are in the file “sb.c”. The function used to tell the output function in “sbio.c” what to display is called put(), and is also in
“sb.c” — it is called by all the structure output functions and is passed a pointer to the structure and a pointer to an array of special “StructData” structures. Setting up this array is the key to adding more structures to SB’s repertoire.

A “StructData” structure is defined in the file “sb.h” seen in the listing. It contains information that SB needs to know about a structure member: its name, type, print option, and size. The name and type are just pointers to strings for display. The print type is needed so that put() knows how to print the member’s value — as a byte, short, or long, signed or unsigned, or as a string or pointer. See the put() function to see how the print codes are interpreted. The other member of a struct StructData, the data size, indicates the number of bytes used by the structure member. The constants BYTESIZE, INTSIZE, and PTRSIZE are defined in “sb.h” to specify the common sizes 1, 2, and 4. When the member is a structure (not a pointer to a structure, but an instance of the structure itself), use the SZ macro defined in “sb.h” to calculate the size of the structure, e.g. SZ(View) instead of sizeof(struct View).

By looking at the functions PrWindow(), PrScreen(), and PrGadget() as examples, you should have no difficulty adding the functions to display system structures of interest to you. If you add a new structure, you’ll have to allow the user to select it from another structure by removing the parentheses from around the member name and type. An opening parenthesis at the start of a member name instead of a space prevents that member from being selected by the user (doesn’t make it a gadget). Just remove these parentheses from all functions whose structure contains a pointer to your newly-implemented structure, and add a call to your new function in the SWITCH statement. Again, refer to the functions mentioned above for clarification.

In the print function for any new structure you’ve added to the program, set up a static ‘structdata’ array of StructData structures (try saying that three times quickly!) as in the other functions. Members’ names should begin with a space for flags, arrays, or pointers to structures that SB can handle (i.e. anything for which a print function exists), a minus (−) for simple data elements that will be rendered in black and will not be selectable by the user, or a parenthesis for pointers to structures that are not supported by the program.

The output of members to the window and the input of a response from the user is handled by the same function, GetChoice(). Selectable structure members are implemented as Intuition gadgets that have no rendering other than the associated IntiText. An array of 16 IntiText structures is set up to contain the text for each line printed to the window. An array of 16 boolean gadgets, each pointing to a different IntiText structure, is also declared. Gadget structures also exist for the “Previous Level” and “(MORE)” gadgets that appear on the window.

When GetChoice() is called, it calls Redisplay() to put up the structure member names and build the required list of gadgets. Redisplay() first removes all existing gadgets except the first in the gadget list, the “Previous Level” gadget (BackGadg). Depending on the number of structure members to be displayed, up to 16 passes through a loop are made to either print an IntiText or add the associated gadget to the gadget list. After the loop, the “(MORE)” gadget (MoreGadg) is added if there are more than 16 structure members to display. Finally, a call to the Intuition function RefreshGadgets() displays the gadget text on the window.

After Redisplay() has done its job, GetChoice() waits for an IDCMD (Intuition Direct Communication Message Port) event, which will signify that the user has selected a gadget. If the window close gadget was selected, CloseOut() is called to close up libraries and the open window, then call exit() to fix up the stack and return to CLI. If another gadget was selected, the gadget’s ID is returned to the caller of GetChoice(). The IDs of the structure member gadgets are their ordinal values — 1 for the first and so on. The “Previous Level” gadget has an ID of 0, and the “(MORE)” gadget’s ID is the constant MOREGADG, defined as 25 in “sb.h”.

To SB 2.0... And Beyond!

As you’ve read, SB can be expected to expand well beyond its current capabilities. If anyone adds new structures, flags, or features to SB, we would be happy to incorporate the changes in the latest version for wide public-domain distribution. We will play ‘keeper of the source’ and attempt to coordinate all improvements to maintain the latest, greatest SB that will be uploaded to ComputerServe and other services, and included on our public-domain Amiga disks (probably available next issue).

Meanwhile, have fun picking Intuition’s brains with SB 1.0, and we’ll see you next issue!

************** File "header/sb.h" **************
#include <intuition/intuition.h>
define SZ(x) sizeof(x) / sizeof(char)  
define BYTESIZE sizeof (APTR)  
define INTSIZE sizeof (int)  
define PTRSIZE sizeof (APTR)  
define MAXGADG 16  
define MOREGADG 25 /* ID of "more" gadget */
struct StructData (char *membername;  
    char *membertype;  
    int printtype;  
    int datatime;  
    )

************** File "sb.c"

/*  
   The Transactor Structure Browser (SB) V1.0  
   From Transactor Magazine, Volume 7, Issue 06  
   By Nick Sullivan and Chris Zamora (ANAI) (c) 1986  
   SB displays system structures via pointers found  
   in other structures. You start from intuitionbase.  
   structures implemented in V1.0:  
      intuitionbase, Window, Screen, RastPort, Bitmap, Gadget  
      Usage is through intuition, clicking on structure member  
      names to display info or a new structure.  
   ***** THIS PROGRAM MAY BE FREELY DISTRIBUTED ***** 
*/

#include "header/sb.h"  
define WIN(x,y) ((x)+(y)+(y))  
define BACKGROUND 4  
#define external struct intuitionbase *intuitionbase;  
#define external struct intuitiontext *intuitiontext;  
define APORT OpenLibrary ();  
#define extern struct intuitiontext *intuitiontext;  
#define extern char textline[MAXGADG + 1] [80];  

The Transactor 66 May 1987: Volume 7, Issue 06
main ()
{
    int choice = -1;
    SetDefault(); /* open int LIBRARY & window */
    while (choices) {
        puttext("Choose a Library structure, NULL;"
            struct Int Lib Text = (BYTE *) Intuition"
            Struct Library;
            BackText.Text = (BYTE *)"Quit Program;"
            switch (choice = GetChoice()) {
                case 1: PrintIntBase ("The IntuitionBase structure, IntuitionBase;"
                    print a byte */
                    printf(buf, "$\times x 10^d", num, num);
                    break;
                case 13: setColor (option), buf;
                    ChoiceTextLine.Text = (BYTE *) textoLines(option);"
                    FlagPrint (string, names, flags)
                    char *string, **names; ULONG flags;"
                    int i, line, fields = FLAGFIELDS;
                    char buf[32];
                    SetbackText(); /* prev level */
                    for (i = 0; i < 8; i++) {
                        strcpy(textLines[i], "-");
                        ChoiceTextLine.Text = (BYTE *) textoLines(i);"
                    putheader (string, NULL,"
                        for (i = line = 0; i < 16; i++) {
                            if ((flags & (1L << i)) && names[i]) {
                                sprintf(buf, "$\times x 10^d", names[i]);
                                strctextLines(line, buf);
                                if (line ++)
                                    ChoiceTextLine.Text = (BYTE *) textoLines(line);
                                line++;
                                fields = FLAGFIELDS;
                            }
                        }
                        if (fields < FLAGFIELDS)
                            ChoiceTextLine.Text = (BYTE *) textoLines(line);"
                        while (GetChoice() = line + 1)"
                    );
        HexDumps (string, address, unit, size)
        char *address, *string, int unit; long size;
        int line = 0, c, char buf[32];
        BackText.Text = (BYTE *)"End Hex Dump ";
        if (size < -1)
            size = Ox ???
        do {
            sprintf(buf, "$\times x 10^d$","
                string, address, address);
            putheader (buf, NULL);
            if (line = MAXDAG)
                line = 0;
            while (line < MAXDAG && size > 0) {
                HexLine (address, unit, line++, size);
                size = 16;
                address = address ++;
                c = GetChoice (size > 0 ? MAXDAG + 1 : line);
                while (size > 0 && c == MOREDAG)
                HexLine (address, unit, line, size)
                UBYTE *address; int unit; long size;
                char buf[32];
                static char headdigit[] = "0123456789ABCDUEF"
                sprintf(textLines[line],"
                    \%S\%d, address);";
f (i = 0; i < 16; i++) {
                switch (unit)
                    case BYTESIZE:"
                        j = *address + 1;
                        sprintf(buf, "$\times x 10^d$", headdigit(); i 16), headdigit(j) X 16);
                        break;
                    case INTSIZE:"
                        sprintf(buf, "$\times x 10^d$", (int *) (address + 1));
                        break;
                    case PRINTSIZE:"
                        sprintf(buf, "$\times x 10^d$", (long *) (address + 1));
                        break;
                    break;
                strctextLines(line, buf);
                ChoiceTextLine.Text = (BYTE *) textoLines(line);
            }
        SetOptionText (hdrtext, data, object, size, offset)
        char *hdrtext;
        struct StructureData *data;
        APTR object;
        int size, offset;
        int i, sum;
        SetbackText (offset ? 1 : 0);
        putheader (hdrtext, object,"
            for (i = sum = 0; i < size; i++) {
                put(i, &data[i], object, sum + offset);
                sum += data[i].data[i];
            return (sum + offset);
        })
PrString (heading, string) char *heading, *string;
{
  char *newstr, *malloc();
  putheader(heading, NULL);
  newstr = malloc(strlen(string) + 1);
  *newstr = 't';
  strcpy(newstr + 1, string);
  ChoiceText[0].setFont = (UBYTE *)newstr;
  GetChoice();
  free(newstr);
}

**************************************************************************************

/* Include not needed for Aztec C using provided makefile */
#include "header.h"
#define CHOICEWIDTH 280
#define CHOICEHEIGHT 8
#define SPACING 9
#define TOPSADG 30
#define PUTTEXT(x, y) { Move(rp, (long)x, (long)y); 
 struct IntuitionBase *intuitionBase = NULL;
  struct Xframe *xframe = NULL;
  struct Window *window = NULL;
  struct Rect RP = NULL;
  extern int level;
  Aptr OpenLibrary();
  struct Intuition Text ChoiceText[MAXSADG + 1];
  struct Gadget ChoiceGadgets[MAXGADGET + 1];
  struct Intuition BackText = { 0, 2, AMX, 2, NULL, NULL, NULL,
  struct Intuition MoreText = { 2, 2, AMX, 2, NULL, NULL, NULL, CUBYTE "(MORE)", NULL,
  struct Gadget BackGadgets = { NULL, 10, -12, 10, GDADCOR | GRELBOX, RELERVED, BOOLGADGET, NULL, NULL, &BackText, NULL, NULL, NULL, 0, 0, NULL /* gadget 10 is zero */;
  struct Gadget MoreGadgets = { NULL, 50, 12, 10, GDADCOR | GRELBOX, RELERVED, BOOLGADGET, NULL, NULL, NULL, NULL, NULL, NULL, NULL, MoreText, NULL, NULL, NULL, MOREGADGET, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NUL
else
    AddGadget(MainWindow, &ChoiceGad[0], 11);
}

if (MakeFlag)
    AddGadget(MainWindow, &MakedGad, -11);
/* display gadget library (the text) */
RefreshGadgets(MainWindow, MainWindow, NULL);

SetBackText(wflag int wflag;

BackText().Text = CBYTE (wflag ? "Previous Page" : "Next Page ");

"Previous Page" : "Next Page ");

67

************** File "abscreen.c" **************

/* include not needed for Aztec C using provided makefile */
#include "header/shb.h"
extern int level;

PrScreen(string, screen) char *string; struct Screen *screen;
{
static struct StructData structData = {,
    (KXScreen, *struct Screen *, 0),
    (FirstWindow, *struct Window *, 0),
    (LeftEdge, *SHORT *, 0),
    (TopEdge, *SHORT *, 0),
    (Width, *SHORT *, 0),
    (Height, *SHORT *, 0),
    (Mouse, *SHORT *, 0),
    (Flags, *BYTE *, 0),
    (Title, *BYTE *, 0),
    (DefaultTitle, *BYTE *, 0),
    (BackgroundColor, *BYTE *, 0),
    (BarBorder, *BYTE *, 0),
    (BarBorder, *BYTE *, 0),
    (MenuBorder, *BYTE *, 0),
    (MenuBorder, *BYTE *, 0),
};

static char *flagsNames[32] = {
    "BRIGHTSCreen", "CUSTOMSCREEN", NULL, NULL,
    "BEEPER", "CUSTOMBITMAP", NULL,
};

int i, sum, choice = -1;
ULONG bits;
level = 0;
while (choice = SetOptionText(string, structData, (APTR)Screen, DATASIZE, 0))
    switch (choice = GetChoice(DATASIZE)) {
    case 1:
        PrScreen("The next screen in Intuition's list", screen = NextScreen);
        break;
    case 9:
        PrScreen("The screen's first window", screen = FirstWindow);
        break;
    case 10:
        PrScreen("The screen's first window", screen = NextWindow);
        break;
    case 11:
        PrScreen("The screen's first window", screen = DefaultTitle);
        break;
    case 12:
        PrScreen("Screen members (page 2)", screen, sum = 2);
        break;
    }

level = 0;
PrScreen2(string, screen, offset) char *string; struct Screen *screen; int offset;
{
static struct StructData structData = {,
    (ScreenTop, *BYTE *, 0),
    (ScreenLeft, *BYTE *, 0),
    (ScreenRight, *BYTE *, 0),
    (ScreenBottom, *BYTE *, 0),
    (Front, *struct StructData *, 0),
    (ViewPort, *struct ViewPort *, 0),
    (RectPort, *struct RectPort *, 0),
    (Bimap, *BYTE *, 0),
    (LayerInfo, *struct LayerInfo *, 0),
    (FirstGadget, *struct Gadget *, 0),
    (DetailPanel, *BYTE *, 0),
    (BlockPen, *BYTE *, 0),
    (SaveColor, *BYTE *, 0),
    (BarLayer, *struct Layer *, 0),
    (UserData, *BYTE *, 0),
};

int i, sum, choice = -1;
level = 0;
while (choice = SetOptionText(string, structData, (APTR)Screen, DATASIZE, offset))
    switch (choice = GetChoice(DATASIZE)) {
    case 7:
        PrRectPort("The screen's RectPort", screen = RectPort);
        break;
    case 8:
        PrBimap("The screen's Bimap", screen = Bimap);
        break;
    case 10:
        PrGadget("The screen's first gadget", screen = FirstGadget);
        break;
    }
level = 0;
PrScreen2(string, screen, offset) char *string; struct Screen *screen; int offset;
{
static struct StructData structData = {,
    (ScreenTop, *BYTE *, 0),
    (ScreenLeft, *BYTE *, 0),
    (ScreenRight, *BYTE *, 0),
    (ScreenBottom, *BYTE *, 0),
    (Front, *struct StructData *, 0),
    (ViewPort, *struct ViewPort *, 0),
    (RectPort, *struct RectPort *, 0),
    (Bimap, *BYTE *, 0),
    (LayerInfo, *struct LayerInfo *, 0),
    (FirstGadget, *struct Gadget *, 0),
    (DetailPanel, *BYTE *, 0),
    (BlockPen, *BYTE *, 0),
    (SaveColor, *BYTE *, 0),
    (BarLayer, *struct Layer *, 0),
    (UserData, *BYTE *, 0),
};

int i, sum, choice = -1;
level = 0;
while (choice = SetOptionText(string, structData, (APTR)Screen, DATASIZE, offset))
    switch (choice = GetChoice(DATASIZE)) {
    case 7:
        PrRectPort("The screen's RectPort", screen = RectPort);
        break;
    case 8:
        PrBimap("The screen's Bimap", screen = Bimap);
        break;
    case 10:
        PrGadget("The screen's first gadget", screen = FirstGadget);
        break;
    }
level = 0;
PrScreen2(string, screen, offset) char *string; struct Screen *screen; int offset;
{
static struct StructData structData = {,
    (ScreenTop, *BYTE *, 0),
    (ScreenLeft, *BYTE *, 0),
    (ScreenRight, *BYTE *, 0),
    (ScreenBottom, *BYTE *, 0),
    (Front, *struct StructData *, 0),
    (ViewPort, *struct ViewPort *, 0),
    (RectPort, *struct RectPort *, 0),
    (Bimap, *BYTE *, 0),
    (LayerInfo, *struct LayerInfo *, 0),
    (FirstGadget, *struct Gadget *, 0),
    (DetailPanel, *BYTE *, 0),
    (BlockPen, *BYTE *, 0),
    (SaveColor, *BYTE *, 0),
    (BarLayer, *struct Layer *, 0),
    (UserData, *BYTE *, 0),
};

int i, sum, choice = -1;
level = 0;
while (choice = SetOptionText(string, structData, (APTR)Screen, DATASIZE, offset))
    switch (choice = GetChoice(DATASIZE)) {
    case 7:
        PrRectPort("The screen's RectPort", screen = RectPort);
        break;
    case 8:
        PrBimap("The screen's Bimap", screen = Bimap);
        break;
    case 10:
        PrGadget("The screen's first gadget", screen = FirstGadget);
        break;
    }
level = 0;
PrScreen2(string, screen, offset) char *string; struct Screen *screen; int offset;
{
static struct StructData structData = {,
    (ScreenTop, *BYTE *, 0),
    (ScreenLeft, *BYTE *, 0),
    (ScreenRight, *BYTE *, 0),
    (ScreenBottom, *BYTE *, 0),
    (Front, *struct StructData *, 0),
    (ViewPort, *struct ViewPort *, 0),
    (RectPort, *struct RectPort *, 0),
    (Bimap, *BYTE *, 0),
    (LayerInfo, *struct LayerInfo *, 0),
    (FirstGadget, *struct Gadget *, 0),
    (DetailPanel, *BYTE *, 0),
    (BlockPen, *BYTE *, 0),
    (SaveColor, *BYTE *, 0),
    (BarLayer, *struct Layer *, 0),
    (UserData, *BYTE *, 0),
};

int i, sum, choice = -1;
level = 0;
while (choice = SetOptionText(string, structData, (APTR)Screen, DATASIZE, offset))
    switch (choice = GetChoice(DATASIZE)) {
    case 7:
        PrRectPort("The screen's RectPort", screen = RectPort);
        break;
    case 8:
        PrBimap("The screen's Bimap", screen = Bimap);
        break;
    case 10:
        PrGadget("The screen's first gadget", screen = FirstGadget);
        break;
    }
level = 0;
PrScreen2(string, screen, offset) char *string; struct Screen *screen; int offset;
{
```c
bits = (bits & 0xff00) | (1 << ((bits & 0xff) >> 4) + 1));
FlagFlips = gadget type flags set for this gadget,
systypeS.names, (ULTGARD)bits);
} else {
bits = (bits & 0xff00) | (1 << ((bits & 0xff) + 1));
FlagFlips = gadget type flags set for this gadget,
applyS.names, (ULTGARD)bits);
} break;
} level--;

*************** file "abfx.a" ***************

"/ include not needed for Aztec C using provided makefile "/
#include "header/sh.b"

extern int level;

PrBitmap(string, bitmap) char *string; struct Bitmap *bitmap;
{
  static struct StructData structdata[] = {
    {"BytesPerRow", "MABOR", 12, INTSIZE },
    {"Rows", "MABOR", 12, INTSIZE },
    {"Flags", "BYTE", 15, BYTESIZE },
    {"Depth", "BYTE", 15, BYTESIZE },
    {"Pad", "MABOR", 12, INTSIZE },
    {"Planes[E]", "PLANEPIR", 5, PTRSIZE * 8 }
  };

  int i, sum, choice = 1;
  level++;
  while (choice) {
    sum = SetOptionText(string, structdata,
                         (APTR)bitmap, DATASIZE, 0);
    if (choice = GetChoice(DATASIZE)) == 6)
      PrPlanes("bitPlanes belong to the bitmap", bitmap->Planes,
               bitmap->Rows, bitmap->BytesPerRow);
    level++;
  }

  PrPlanes(string, planes, rows, bytes) char *string; PLANEPTR planes[]; UMOR rows, bytes;
  {
    static struct StructData structdata[] = {
      {"BitPlane[0]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[1]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[2]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[3]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[4]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[5]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[6]", "PLANEPIR", 5, PTRSIZE },
      {"BitPlane[7]", "PLANEPIR", 5, PTRSIZE }
    };

    int i, sum, choice = 1;
    level++;
    while (choice) {
      sum = SetOptionText(string, structdata,
                           (APTR)planes, DATASIZE, 0);
      choice = GetChoice(DATASIZE);
      if (choice = GetChoice(DATASIZE)) == 12)
        PrDump("structdata[choice] = ",
                (APTR)planes, DATASIZE, 0);
      level++;
    }

  PrReadPort(string, restport) char *string; struct ReadPort *restport;
  {
    static struct StructData structdata[] = {
      {"Layer", "struct Layer *"}, 5, PTRSIZE },
      {"Bitmap", "struct Bitmap *"}, 5, PTRSIZE },
      {"AreaPt", "struct AreaPt *"}, 5, PTRSIZE },
      {"TopRas", "struct TopRas *"}, 5, PTRSIZE },
      {"AreaInfo", "struct AreaInfo *"}, 5, PTRSIZE },
      {"GelInfo", "struct GelInfo *"}, 5, PTRSIZE },
      {"Mask", "BYTE", 13, BYTESIZE },
      {"Flip", "BYTE", 3, BYTESIZE },
      {"RGBenC", "BYTE", 3, BYTESIZE },
      {"RasPt2", "BYTE", 3, BYTESIZE },
      {"lmpPt", "BYTE", 3, BYTESIZE },
      {"dummy", "BYTE", 3, BYTESIZE },
      {"Flags", "USHORT", 12, INTSIZE },
      {"LinePt", "USHORT", 12, INTSIZE }
    };

    int i, sum, choice = 1;
    level++;
    while (choice) {
      sum = SetOptionText(string, structdata,
                           (APTR)restport, DATASIZE, 0);
      switch (choice = GetChoice(DATASIZE)) {
        case 2: if (restport->Bitmap)
                     PrBitmap("The Bitmap for the ReadPort", restport->Bitmap);
                   break;
        case MOREGAD:
```

Amiga File Structure –
A Second Look

Most readers of Transactor are familiar with the layout of the diskettes formatted on all of the Commodore drives prior to the Amiga. We have studied the books by Dick Imms and Gerry Neufeld, read the articles by Mike Todd of IPCUG, and we have learned about header gaps, tail gaps, and GCR. We know that the directory is on track 18 (or 39 on the 8050/8250), that the number of sectors increases as the head moves out to the edge of the disk, and that the 4040, 2031, 1541, and 8050 drives write on the bottom of the disk only. Most of us are adept at using track and sector editors on those drives. This knowledge can help us on the Amiga, but these diskettes are organized in a very different way.

PHYSICAL LAYOUT

The Amiga diskettes are laid out in tracks and sectors, but each side of the disk has tracks numbered from zero through seventy-nine. Each of these tracks (160 in all) has eleven sectors. Two similarly numbered tracks from opposite sides of the disk make a cylinder with a total of twenty-two sectors. These sectors are numbered sequentially from zero through 1759, and the sector number is stored within the sector. The cylinders are numbered from the outside in. That is, cylinder zero is the outermost, and cylinder 79 is nearest the center. The sectors begin at cylinder 0, sector 0, surface 0 (the top side).

The drives have two heads that move in unison, but they can be addressed individually. Head zero is on the upper side of the disk, and head number one is below. The heads are moved to the desired track with a SEEK command, and the entire track is read in with a CMD_READ or and ETD_READ command. These, and other commands, are used by the track-disk device to control the drives. There are programs available to control the drives for C programmers and for assembly language programmers, but I have not found a way to make the necessary .bmap files to control them from BASIC.

The data is written in MFM (Modified Frequency Modulation) format, and is encoded or decoded in the blitter chip. It is possible to set the drive to write in GCR, but this mode reads or writes at half the speed of MFM. Someone will probably find a way to use this mode so that we can read the normal Commodore disks with the Amiga, and we are not likely to complain about speed if this ability becomes possible.

LOGICAL LAYOUT

Each of the eleven sectors on a track has a sector-header of 16 bytes, plus 16 bytes that are reserved for future use, and 512 bytes of 'usable data', so our 880K diskettes also contain 55K of label data, making a total of at least 935K bytes per disk (there is a hint of 28K more). There are no header gaps or tail gaps, but each track has one gap made up of nulls. Since each track has the same number of sectors, one would assume that the gap of nulls grows longer toward the outer edge of the disk.

The header or label area has the same pattern for all sectors, but the 512 bytes of data are organized differently according to the type of block. In a data file, there are 24 bytes of identification at the beginning of the sector, followed by 488 bytes of actual data. A directory block or a file header block has 24 bytes (six long-words) for sector number, file type, checksum, etc., at the beginning of the block, 288 bytes of hash-table or data, and fifty long-words for the file name, comment, date, forward and backward pointers, etc., at the end of the block.

On an Amiga disk, some information is coded in 8-bit bytes, some in 16-bit words, and some in 32-bit long-words. A 'byte' still means 8 bits, but we must distinguish between 'words' and 'long-words.' In the diagram below, all of the 'words' are actually 32-bit long-words.

DIAGRAM OF BLOCK LAYOUT

For any type of sector:

Label area: 2 bytes of 00
2 bytes of A1 (a sync byte in MFM)
1 byte of format type (FF on 1.0)
1 byte of track number
1 byte of sector number
1 offset byte - MORE ABOUT THIS BELOW!
16 bytes of operating system recovery info (not currently used)
4 bytes of header checksum
4 bytes of data-area checksum

For a data block, such as a sequential file:

Data Area: 1 word of file type
1 word for header key (sector where file begins)
1 word for the sequence number of this block
1 word for number of bytes of data in the block
1 word for the number of the next block in file
1 word for the checksum
up to 488 bytes of data

Or for a root directory block:

Data Area: 1 word of file type
1 word of sector number
1 word of file length
1 word of hash-table size
1 word not used
72 words of hash-table
1 word of bit map flag
25 words of bitmap pages (instead of BAM)
3 words of date and time last altered
13 words of disk name
3 words of date and time of disk creation
3 words of forward and backward pointers
1 word of secondary file type
The software is written to allow for sectors of a different length in the future versions. Instead of pointing to a particular word, the identifying information in the last fifty words are accessed with 'Block-size - n', where 'n' is the number of longwords before the end of the sector.

**READING THE DISK**

When the SEEK command is given, the heads move in unison to the designated track. Upon receiving the READ command, the heads read continuously for two full rotations of the disk, bringing in the contents of an entire cylinder. Eleven sectors from one side of the disk are read into one buffer, and the eleven from the other side go into another. The heads do not wait for a sync mark, and make no effort to find a particular sector. They do not even check for the beginning of a sector! They just read in the data from whatever part of the track is under the head when the READ begins. The data is decoded by the blitter and the decoded bytes are placed in the track buffer as they came from the disk. It is this manner of reading that makes the offset byte of the label area so important.

**THE OFFSET NUMBER**

The very first time a track is written to the disk, it will be written with a gap of null bytes, and then the eleven sectors in numerical order. In the track buffer, this first write would have this form:

```
Sector No. 0 1 2 3 4 5 6 7 8 9 10 nulls
Offset No. 11 10 9 8 7 6 5 4 3 2 1
```

and it would be encoded for the disk in the exact order it has in the buffer. The offset number tells the data pointer how many more sectors must be read or written before it reaches the gap of nulls.

When this sector is read back, however, the heads are not likely to begin reading at the gap, nor at sector 0. Suppose that sector 7 were under the head at the beginning of the READ. Then the track buffer would be like this:

```
Part of sector 7 (junk) and then:
Sector No. 8 9 10 nulls 0 1 2 3 4 5 6 7 (more junk)
Offset No. 3 2 1 11 10 9 8 7 6 5 4
```

Once in the track buffer, the pointer will find the first sync mark and block move the data in sectors 8, 9, and 10 so that sector eight is aligned with the beginning of the track buffer. Then it will move past the nulls to find the next sync mark and move the remaining sectors up to join the first ones, leaving the nulls at the end of the buffer. When a sector is needed for use, the software sorts through the sectors in the buffer, and brings in the ones that are needed. The disk need not be read again until there is need to read a different track. If the data has been changed, the current track will be written back to the disk before the next track is read in.

When our hypothetical track is written back to the disk, the sectors retain their original numbers, but not the original position. So now the sector offsets will be changed to agree with the new order of the sectors:

```
Sector No. 8 9 10 0 1 2 3 4 5 6 7 nulls
Offset No. 11 10 9 8 7 6 5 4 3 2 1
```

Each time the track is re-written, the sector offsets will be changed, and the computer uses the offset numbers to find the correct sector. This method of writing to the track, and the practice of reading an entire track each time, removes the need for interleaving and header- or tail-gaps, allowing about twenty percent more information to be put on the disk. By reading an entire track sequentially, disk access is much faster and more efficient.

**MFM ENCODING**

When data is encoded in MFM, an extra digit is placed in front of each bit to ensure that there will never be two '1's in a row, nor more than four zeros in a row. In MFM, a '1' bit becomes '01'. A '0' bit will be encoded as '00' if it follows a '1' bit, but as '10' if it follows a '0' bit. Thus, for every bit of data to be recorded, two bits are actually written. The Amiga software, using the blitter chip for encoding and decoding, separates the odd bits from the even bits. First the odd bits are encoded and written out; then the even bits are shifted left one position, encoded, and written behind the odd bits on the disk. When the data is read back, the blitter chip removes the extra leading zeros and ones that were added for encoding, leaving 'holes' between the odd bits for the even bits to fall into.

The encoding process for the sector labels is interesting. The first four bytes of the label are encoded as separate bytes—first the odd bits, then the even bits of each byte. Then the next four (the format, track number, sector number, and offset) are encoded as one long-word. The 16 bytes of operating system recovery information are encoded as a block of 16 bytes. The header checksum and the data area checksum are each encoded as one long-word. The 512 bytes of the data block are encoded as a single block of data—the odd bits of all 512 bytes, and then the even bits.

**A BIT OF MATHEMATICS**

Disks are timed to rotate at 300 revolutions per minute. For GCR encoding, the Amiga writes at a rate of four microseconds per bit. In MFM, it requires only two microseconds per bit. At 300 revolutions per minute, there would be five revolutions per second, or two hundred milliseconds per revolution. Two microseconds per bit permits the writing of 100,000 bits per revolution. It takes two MFM bits for each actual data bit, though, allowing a maximum of 50,000 data bits per track. We have 160 tracks, making 8,000,000 possible data bits, or 1,000,000 bytes per disk. We have accounted for 957,440 bytes (935 x 1024), excluding the gaps of nulls. Isn't this remarkable efficiency?

**AND A MYSTERY**

There is a mystery about the sector labels. The ROM KERNEL MANUAL says that there is a 16-byte section of descriptive data for each sector, a total of 27.5K bytes. The drive, it says, does not interpret these sections unless the programmer has instructed it to do so. This cannot be the label area described above, because the information in that area must be interpreted for normal drive operations. Does this refer to the 16 bytes the RKMS says are currently unused, but are to be used for operating system recovery information? Or is there another descriptive label area? Have YOU found the answer to this mystery?
Amiga Dispatches
by Tim Grantham, Toronto, Ontario

"And on the third quarter, Commodore arose."

I don’t know about you but I got a big kick out of the fourth annual World of Commodore show, held at the International Centre in Toronto during the first week of December. After a year of being kept on the edge of our seats, it was very satisfying to actually see the Sidecar, the Genlock and DOS 1.2 working and up for sale. True, Sidecar cost about $400 too much (I mean, really, you can get a Commodore PC 10 II for $1000 and that comes with a keyboard, and the Genlock wasn’t quite ready for sale, but these were minor flies in the soup. With plenty of exhibitors, vendors and visitors, the whole show positively radiated optimism and good times returned for Commodore.

It was also a chance for me to meet the faces attached to the handles – if you spend any time on the information networks you know what I mean by that. ‘Hazy’ Dave Haynie (author of Disksalv), John Foust (editor of Amazing Computing), Larry Phillips (sysop of the Amiga forum on The Source) from Vancouver, and Brian Niessen and Wayne Schmidt (assistant sysops on the Commodore Forums on CompuServe), Dave Paul... it was a pleasure to actually meet them all face to face and exchange news and views. The press was there in force – the Transactor, of course, represented by the usual gang of idiots; Run and AmigaWorld, fronted by publisher Steven Twombly and editor Guy Wright; Henry Shilling and Linda Byrket Shilling of Ami Project; and Chris Willey of AmiTalk.

Other personages present included Jim Butterfield, Paul Higginbottom (author of LPD Writer), Vladimir Schneider (author of Professional Text Engine) and Bob Hoover of Mimetics (author of Soundscape).

All the major Amiga software companies were there, with the exception of the game companies. (With the money Electronic Arts is making on the Amiga I’m not surprised they didn’t feel the need to show up, though they have been there in the past.) The number of major vendors made for some healthy competition: by show’s end, a 512K Amiga with 1080 colour monitor could be had for $1650 (Cdn.). And who ever thought Computerland would actually exhibit at a lowly Commodore show? (I shouldn’t sneer. Computerland has done fairly well with the Amiga, and even better with the PC 10 II. They’ll be especially happy now that Commodore is coming out with an AT compatible.)

What follows is not in any particular order; nor is it necessarily a balanced view of the Amiga products exhibited – one of the nice things about the show was that there was so much to see, I’m not sure I saw it all. My apologies to anyone I’ve left out.

- Very Vivid – Without a doubt, the hit of the show. This product received its premiere at the World of Commodore show, as part of the "Cricketts" multimedia performance and blew everyone away. It’s not easy to describe but try to imagine the following: a dancer stands in front of a video camera, dressed in white. On the Amiga’s screen, and on the other monitors around the stage, we see his image in silhouette, moving on a background graphic generated by the Amiga, a version of Leonardo da Vinci’s famous sketch of a multi-angled man standing inside a circle. The performer reaches out and he digitized image on the Amiga screen reaches out and his hand touches a small circle. Simultaneously, a synthesizer connected via MIDI to the Amiga plays the first note of the opening fanfare from Also Sprach Zarathustra. The performer, or rather his image, touches other small circles above his head on the screen. Each time, the circles pulse and produce the notes of the fanfare, a different note, and sometimes a different timbre, being triggered on the synthesizer by each circle. As the fanfare ends, he reaches up and touches another circle and the background switches to a strange landscape. The two musicians accompanying him start playing.

If we were to look at the performer in front of the camera, we would see a human against a plain backdrop making odd gesticulations into thin air. But our eyes are glued to the monitors, as we watch him jam with the musicians, quickly touching this Amiga object for this sound, and another object for that one. We watch as his image grasps one coloured circle and paints the screen with it; then another, and another, until he stands amid coils of colours. In one quietly amazing sequence, two coloured spheres attach themselves to his arms, and stick there, lobbing back and forth as he waves his arms. Then, he snaps his arms, and the spheres detach, change into birds, and fly off in opposite directions.

And so it went, effect after dazzling effect, beautifully conceived and professionally performed, for a solid 20 minutes. The applause was tumultuous and there were shouts of ‘Bravo!’ I saw it three times.

The potential for this kind of product is mind boggling – not just for performance applications but for production houses, educational programs, therapeutic programs and – let’s get serious, here – games! For the first time, you, or a very reasonable facsimile thereof, will be able to actually enter a game. And you thought Tron was idle fantasy.

- Soundscape, by Mimetica. This marvelous MIDI sequencer was used to control not only the synthesizers in the “Cricketts” performance but the lights and video sequences as well. Electronic Arts has apparently written a module for Soundscape that will let their Deluxe Music Construction Set print Soundscape files. Soundscape has all the signs of becoming the MIDI music software for the Amiga.

- Draw Plus and Sonix – Aegis Development were running their CAD software on a packaged system they are co-marketing with Roland and Commodore: a 512K Amiga, 1080 colour monitor, and a DXY 880 or 980 plotter. I’ve since had a closer look at Draw Plus: it’s impressive software that comes with good documentation and plenty of example drawings and parts libraries. However, I am no design engineer and really have nothing to compare it to. Sonix is the reincarnation of Musicraft with MIDI capability and IFF compatibility added.

- PageSetter and Publisher – Commodore is now touting desktop publishing programs like these two, published by Gold Disk and Brown-Wagh, respectively, as the application that will make the Amiga take off. While these programs certainly go well beyond Print Shop, they must be
improved before they can compete with the likes of PageMaker. Gold Disk does not appear to have kerning, for example (a process of adjusting the space between letters depending on their shape), and neither has PostScript capability, though both claim that such will soon be available. The Macintosh will always have a lead in this field: as Henry Shilling of AmiProject magazine reminded us in a TPUG panel discussion on the future of the Amiga, the pixel size on the Mac's screen is exactly 1/72", or one 'point' in typesetting jargon — in other words, it was designed from the desktop up for publishing applications.

• System Monitor — This nifty utility is put out by a small American firm called Zen Software. It may just be the product that will let Jim Butterfield really find out what the hell is going on in the Amiga Kernel. Among other things, it lets you assign a monitor window to any task and track its current status, resource allocation, and CPU usage.

• Professional Text Engine — from Zirkonics of Montreal. If you ever wished your favourite text editor had commands to, say, reverse the order of adjacent characters or assign custom format commands to 'hot' keys, this programmable text editor is the answer. Not only is the keyboard fully programmable, using a simple macro language, even the drop-down menus and the action of the mouse buttons can be customized. The number of files that can be open at once is limited only by the memory available. You can cut and paste freely between any file. Configuration files to allow PTE to act like other editors, such as Wordstar, are available. But the fun is programming it for all the features you couldn't have before. Pretty good, for $150 (Cdn.).

• DOS–2–DOS — This is a utility that provides translation between AmigaDOS and MS–DOS, stripping high bits if necessary, and taking care of the linefeed/carriage return combinations. The software can also format both kinds of disks, and search through sub-directories. When I pointed out to the saleswoman that such utilities are provided in AmigaDOS 1.2, she started to squirm, but pointed out that DOS–2–DOS permits use of wildcards that can be used to automate the process to some extent.

• Key to C — another product worthy of note (grin). This is a library of functions written in C for C programmers that closely approximate BASIC statements and functions.

Other software at the show: Pro Video CGI–1, from JDK Images, a video character generator; B.E.S.T. business management and accounting program, from Business Electronics Software & Technology; MiAmiga File II file management software, MiAmiga Word word processor, MiAmiga Ledger general ledger, all from SoftWood Company; JetSet, utilities and fonts for the Hewlett-Packard LaserJet + laser printer, from C Ltd.; Order desktop organizer from Northeast Software Group; Datamat programmable database software, from Transtime Technologies Corp, being marketed in Canada by Creative Systems; LPD Planner and LPD Writer, spreadsheet and word processing software from Digital Solutions; Zing!, a mouse-driven CLI, from Meridian Software; ProWrite multiple-font word processor, from New Horizons Software; a version of Superkit for the Amiga from Prism Software; and Superbase and Logistix programmable database and integrated spreadsheet, from Progressive Peripherals.

What was particularly encouraging about the products shown at the show was not so much what they did, but what they made possible. Soundscape, Very Vivid, PTE and others are marvelous because they have been designed in the spirit of the machine — open-ended, smoothly multitasking, providing the user with the most flexible tools possible for her/his endeavours.

And the hardware...

• Amiga Sidecar — Complete and for sale. Everything works as promised, including the ability to stick 2 megs of RAM in the Sidecar for use by the Amiga.

• Genlock — Ditto, almost. It should be for sale by the time you read this. (I know, I know, I've said that before.)

• DigiView — Tim Jenison, the developer, was at the show and demonstrated the improvements: cleaner images, motorized filter wheel, and editing functions — impressive product.

• Both Xetec and C Ltd. were selling SCSI interfaces and 20 Meg hard–drives for the Amiga for $1000 (US). Both permitted daisy-chaining of further SCSI devices and both came with Amiga driver software. As with the Microforge and Tecmar units, the Xetec controller suspends operation of all other tasks on the Amiga during reads and writes; I assume this is also true of the C Ltd. product.

• C Ltd. was also selling their aMEGA 1 megabyte memory expansion units for the Amiga. They were joined by Comspec's AX2000 2 megabyte board, and RS Data Systems' Pow'r-card, which can attach to the full 8 megabytes of RAM. This last is an imposing unit which offers a lot of memory at very reasonable cost. However, it is not auto-configuring: it has to be added by calling 'addemn' in the startup–sequence.

A few of these memory expansion units (such as the ASDG Inc. RAM board, Side Effects's Side-Store, and the Microbots unit) come with virtual disk capability. This works just like the RAM disk but it can be written protected. This means that should your Amiga crash while testing your latest tic–tac–toe game, say, you can reboot and the files in the virtual disk will still be intact — your source and include files wouldn't need any reloading... a tremendous convenience.

• Perfect Sound, from SunRize Industries, is a stereo sound digitizer for the Amiga that comes with very good sound sample editing software. It is currently the only stereo digitizer and sells for the extremely reasonable price of $79.95 (US).

I'll finish this installment of A–D with some news about expansion hardware that I was unable to put into the last column. I was hoping to see this stuff at the WOC but little of it was shown. Nevertheless, it is very heartening to see the broad variety of products available. There are expansion boxes like the PAL unit from Byte–by–Byte, the Turbo Amiga from CSA, and the Side–Arm from Side–Effects — they come in all shapes and sizes but most offer at least two slots for 100-pin Zorro standard option boards. You can fill them with RAM boards, SCSI interfaces and hard drives, DMA hard drives, coprocessor boards, tape backup drives — the list goes on and on.

One company in particular drew my attention with their Ethernet and ARCNET interfaces. Ameristar Technologies also supplies a version of Sun Microsystems's NFS with the Ethernet interface that enables the Amiga to act as a graphics workstation on a network with Sun workstations and other computers using an implementation of NFS. Each interface is available in either a bus or backplane version (86-pin or 100-pin). These capabilities, combined with the coprocessor boards from CSA and others, are making the Amiga a serious engineering workstation computer.

The essential features to look for in any expansion hardware are threelfold: if it is a bus attachment, does it have a pass–through? Does it auto–configure? And is it a no wait–state device? If the manufacturer answers yes to all of these, then you are safe to pass on to more mundane matters, such as price, quality and availability.

Finally, my apologies to any of you who have sent me mail on ComputeServe or PeopleLink. I was without a modem for over a month and was unable to respond. After suffering acute withdrawal symptoms, I have managed to obtain another, so if you have any questions or comments, you can reach me at CIS 71426,1646 or PeopleLink AMTAG. Until next time, may your mouse never squeak.
Compu-toons

"Using electronic mail has saved us $500,000! But, we lost $750,000 for not being able to use the excuse 'the check is in the mail!"

I'm sorry to bother you on your holidays, Ms. Thomas, but I can't find 'W'!

The perils of hunt and peck

"Apparently Bob's suggestion to turn the office paperless wasn't received well!"
News BRK

Submitting NEWS BRK Press Releases

If you have a press release you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Transactor Currency Standard Decreed

After a long day sweating in the Transactor Abbreviations Laboratory (TAL), our scientists have come up with a new convention that will be Official Transactor Policy (OTP) from now on. The rest of the world is invited to follow suit. Without getting too technical, the idea is this: prices for our mail order products (MOPs) and for products listed here in NEWS BRK, will henceforth be given in one of three forms: $x (Cdn) for Canadian prices, $x (US) for US prices, and $xx (US/C) when the specified amount is to be paid in Canadian dollars by Canadians, and US dollars by Americans. For an example of the new US/C abbreviation in action, please refer to the final paragraph of the following announcement (FPctFA).

Subscription Intersection Set

On closer inspection we have found the overlap of TPUG members and Transactor subscribers to be about 1000, not 400 as previously reported. During our first comparison, many went undetected due to differences between the two mail list databases. It seems that a computer just can’t tell if two names are the same when a middle initial is included in one but not the other. To compensate, a program was written that bordered on artificial intelligence. However, it’s still possible that some matches were missed, especially if the postal/zip code was different in the two databases. So, if you’re a TPUG member AND a Transactor subscriber, and you’re still getting two issues, please let us know and the two will be combined.

Combining subscriptions has also proved to be an exercise in mind bending. We had hoped to have updated the expiry date for all the combined subscriptions by this issue. However, only those TPUG members whose memberships would have expired before receiving this issue have had their expiry dates extended. The remaining TPUG/Transactor combinations will have their expiry dates updated by next issue.

Check your last magazine. If the expiry date shows “Dec 86” or “Jan 87”, then your expiry date should now be extended by “the number of issues remaining in your Transactor subscription, times two months”. When this date actually arrives, you should have received the same number of magazines shown in the table published in News BRK last issue.

At this point you’ll receive a renewal notice. The notice will come from TPUG because the combination is done by merging the two into the TPUG database. Naturally you should renew to one OR the other. Renewing to both will mean you’ll start getting two magazines all over again, and after we get this situation sorted out, we’d like it to be eliminated for good. The fact is, we could make a career outta this. So, after all the existing overlapping memberships/subscriptions have been combined and extended, any and all new ones will get two magazines; one with an insert from TPUG, and another without an insert from us. This applies mainly to those who are currently Transactor subscribers and are not TPUG members, but considering becoming one.

If you’re renewing a subscription, or subscribing for the first time, you’ll have the choice of getting the regular magazine (no TPUG insert) or becoming/remaining a TPUG member, and getting the 8-page TPUG insert as part of your Transactor. If you want just the Transactor, use the insert card in the magazine to subscribe/renew. If you want the insert as well, send your TPUG renewal form to TPUG (they’ll send it to you before your current membership expires) or, if you’re not currently a member, contact them about becoming one. The current cost of a TPUG ‘associate’ membership is $25.00 (US/C). For that you get the insert plus access to TPUG’s large public domain disk library, neither of which comes with the $15.00 (US/C) subscription to the Transactor.

Disk Subscription Notes

Many of those who fall into the intersection set described above also have Transactor Disk subscriptions. These will still be handled by us, and when they expire, a notice will be sent. Some have pointed out that, “we would find it much easier to renew to both if they were to both expire at the same time. Now that our memberships/subscriptions have been combined, the magazines end at one issue and the disks at another”. Good point. What we suggest is this: when you renew your disk subscription, add $7.50 (US/C) for every disk necessary to make your disk and magazine subscriptions concurrent.

Toronto CompuServe Node

The direct dial port to CompuServe in Toronto is 752-4150. This number has been in effect since September ’86. However, the CompuServe Intro-Paks that were bound into the Gizmos and Gadgets issue (released Oct. 1) still show the old number. If you haven’t used your Intro-Pak yet, please make a note in the list at the back of the booklet.

Free Transactor T’s with Mag + Disk Subscription

Subscribe or renew to a combination magazine and disk subscription, and we’ll send you a free Transactor T-Shirt! You save 29% off the magazines, 16% off the disks, and get a Transactor T worth $13.95 ($17.95 if you order the jumbo size)! The T-Shirts come in 5 sizes (red only), with a 3-color screen featuring Duke, our mascot, dressed in a snappy white tux, standing behind the Transactor logo done in yellow with black “3-D” borders. The screen was done using a special “super-opaquing” process that cost us quite a bit more than those decals that crack and fade. Mine has been through the wash at least 25 times now, and it still shows virtually no sign of wear due to what’s called “washing machine punishment”.

Subscriber Mail Orders

If you’re a Transactor subscriber, and you’re using the postage paid order card to purchase items other than a subscription, please write your subscriber number on the card. This way your order is recorded along with your subscription information in our database.

Customs/Duty on Hardware Products

Shipping hardware to the US from Canada often incurs customs and/or duty charges at the destination. Some of our suppliers are in the US and for US orders processed by us, we have the items shipped direct without bringing them into Canada. However, other hardware items manufactured in Canada and sent to US destinations may arrive with a surcharge payable. The Transactor cannot be responsible for these charges. This may also add to delivery delays. If you’ve placed an order for a hardware item and it seems to be taking a rather long time to arrive, it may be sitting at your local customs office, in which case a notice is probably on its way to you to come pick it up.
Sold Out!

The Toolbox from Pro-Line, PAL and POWER for the 64, is now sold out. Refunds will be sent for any orders beyond what could be filled.

Volume 4, Issue 03 and Volume 5, Issue 02 (The Transition to Machine Language issue) are now only available on microfiche.

Transactor Mail Order

The following details are for products listed on the mail order card. If you have a particular question about an item that isn’t answered here, please write or call. We’ll get back to you and most likely incorporate the answer into future editions of these descriptions so that others might benefit from your enquiry.

■ Moving Pictures - the C-64 Animation System, $29.95 (US/C)
   This package is a fast, smooth, full-screen animator for the Commodore 64, written by AHA! (Acme Heuristic Applications!). Moving Pictures gives you the most accurate creation of a frame, giving you the ability to move, animate, blend, or even animate objects in a frame with a single command. The 'transcendental' version can be used in the Moving Pictures command to display a complete control of animated creations. BASIC is still available for editing scripts or executing programs even while a movie is being displayed. Animation sequences can easily be added to BASIC programs. Moving Pictures features include: split screen operation – part graphics, part text – even while a movie is running; repeat, stop at any frame, change position and colors, vary display speed, etc.; and several movies in memory and switch instantly from one movie to another; instant, on-line help available at the touch of a key; no copy protection used on disk.

■ Volksmodem 12, w/cable, and CIS Intro-Pack, $329.00 (Cdn), $199 (US)
   Not only do you get the Volksmodem 12 (DOC approved), but you get the cable at no extra charge (the C64 cable goes directly onto the User Port, and the RS232 cable for any standard RS232 DB-25 female connector) Plus you’ll receive a free Compuserve Intro-Pak which contains a User ID, a Password, and $15.00 of connect time! The Volksmodem 12 will work at 300 or 1200 baud, and is “Hayes compatible” so it will work with virtually any terminal software because the commands are controlled by you from the keyboard – just type “AT” (for ATention) and follow with any of several easy-to-remember commands – no special POKing or elaborate dialing routines necessary! I’ve been using a Hayes for almost 3 years, and my Volks for over a year – I love them both! - KJH It comes with (get this) a 5 year manufacturer’s warranty on parts and labour! The modem is shipped insured via UPS at no extra charge.

■ Intelligent I/O Interface Cards
   BH110 I/O Interface Card w/documentation $129 (US), $199 (Cdn)
   BH100-AD8 8-Channel A to D Conversion Module $45 (US), $69 (Cdn)
   BH110 Beginners Course $159 (US), $239 (Cdn)
   BH100-S Security System $25 (US), $39 (Cdn)
   These products from Intelligent I/O will make great Christmas gifts! And if you’ve been wondering what to do with that VIC 20 that doesn’t get much attention anymore, they’re perfect! If you’ve ever wanted to start doing some real world interfacing, real easy, and inexpensive, then these items are ideal. The boards they send us for evaluation are currently watching for floods in my basement. Too bad I didn’t think of it before the flood – it only took about an hour using spare parts I had laying around – no resistors, no capacitors, just two strips of metal, a piece of styrofoam, a brick, and about 20 feet of wire that was also collecting dust. Once I get time, I intend to make it do some more surveillance since only one channel is currently in use. And the program to do it? A quick and messy 5 lines! Since the boards are memory mapped through the cartridge port, a PEEK is all you need! The 22 page manual is clear and concise. All products come with a 90 day manufacturer’s warranty. Shipped insured via UPS at no extra charge.

■ Transactor T-Shirts, $13.95 and $17.95 (US/C)
   As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. They’re 13.95 each, $17.95 for the Jumbo. The Jumbo makes a good nightshirt/beach-top – it’s BIG. I’m 6 foot tall, and weigh in at a slim 150 pounds – the Small fits me tight, but that’s how I like them. If you don’t, we suggest you order them 1 size over what you usually buy. The design is screened using a “super-opaque” process so they wear much longer than your ordinary screens and iron-ons.

■ The Transactor Book of Bits and Pieces #1, $14.95 (US/C)
   Not counting the Table of Contents, the Index, and title pages, it’s 246 pages of Bits and Pieces from issues of The Transactor, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference – no more flipping through magazines for that one bit that you just know is somewhere. Also, each item is forward/reverse referenced. Occasionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Transactor 1541 ROM Upgrades, $59.95 (US/C)
   You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. There are 2 ROMs per set, and they fix not only the Save@ bug, but a number of other bugs too (as described in P.A. Slaymaker’s article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn’t been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

■ The Micro Sleuth: C64/1541 Test Cartridge, $89.95 (US), $129.95 (Cdn)
   This cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:
   1) Check drive speed
   2) Check drive alignment
   3) 1541 Serial test
   4) C64 serial test
   5) Joystick port 1 test
   6) Joystick port 2 test
   7) Cassette port test
   8) User port test
   A second board, that plugs onto the User Port, contains 8 LEDs that lets you zero in on the faulty chip. Complete with manual.

■ Inner Space Anthology $14.95 (US/C)
   This is our ever popular Complete Commodore Inner Space Anthology. Even after a year and a half, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no “reading” material (ie. “paragraphs”). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MML commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ AX1000 Amiga 1 MEG RAM Box $729.00 (+ $100 S&H) (LS), $1035.00 (+ $25 S&H) (Cdn)
   ■ AX1000 Amiga 2 MEG RAM Box $899.00 (+ $100 S&H) (LS), $1276.00 (+ $25 S&H) (Cdn)
   The AX2000 adds 2 Megabytes of "fast" RAM to the Amiga, allowing more tasks to run in the system at once, or for use as a fast RAM-drive. The unit plugs into the expansion connector on the side of the Amiga and duplicates the connector for other devices to plug into. Up to two RAM boards may be plugged in together (limited by the Amiga’s power supply), adding 4 Megabytes. The box has "auto-config", so with Kickstart 1.2 the RAM will automatically be added to the system when it is booted. If you are using Kickstart 1.0 or 1.1 (no auto-config), you can use the program included with the AX2000 to add the memory to the system, and change your startup-sequence to automatically add the memory on power-up. Standard expansion bus architecture was used in the design of the AX2000, ensuring compatibility with all peripherals and operating system releases. The
unobtrusive steel box is the same height and colour as the Amiga, and snug up to the side without taking up much extra space. The unit is built tough and comes with a 1 year manufacturer warranty.

This seems to be the most highly-recommended Amiga RAM board, and the first one to actually be available, so we're selling it here at The Transactor. You can order the AX2000 or the 1-Meg AX1000 from the subscription form in this issue. Shipping and Handling to the USA, is via courier and includes all customs clearance, or you can opt to clear shipments yourself and have it shipped "collect".

- **Superpak 1.0** C64 $49.95 (US), $59.95 (Cdn)
- **Pocket Writer** C64 $29.95 (US), $39.95 (Cdn)
- **Pocket Planner** C64 $29.95 (US), $39.95 (Cdn)
- **Pocket Filer** C64 $29.95 (US), $39.95 (Cdn)
- **Superpaks** C128 $59.95 (US), $69.95 (Cdn)
- **Pocket Writer** C128 $39.95 (US), $49.95 (Cdn)
- **Pocket Planner** C128 $39.95 (US), $49.95 (Cdn)
- **Pocket Filer** C128 $39.95 (US), $49.95 (Cdn)
- **Pocket Dictionary** $14.95 (US), $19.95 (Cdn)

*Version 2.0 of the software trio from Digital Solutions is now in production. The new packages include both the 64 and 128 versions on the same disk. Each 2.0 Pocket package will sell for $59.95 (US) or $64.95 (Cdn). A Superpaks will include all three for $99.95 (US) or $139.95 (Cdn). The Pocket Dictionary is still $14.95 (US), $19.95 (Cdn). However, they won't be available from us until next issue.*

*Version 1.0 is still available, and at terrific prices! The 64 and 128 versions still come in separate packages, but the real deal is the special price for all three. The C64 Superpak is $49.95 (US) or $59.95 (Cdn). C128 Superpaks are $59.95 (US) or $69.95 (Cdn). To top it off, we'll throw in the Pocket Dictionary program for free! If you average the price of all four, it comes to less than the price of two!*

- **The TransBASIC Disk** $9.95 (US/C)

This is the complete collection of every TransBASIC module ever published up to Volume 7, Issue 01. There are over 120 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

- **Super Kit 1541** $29.95 (US), $39.95 (Cdn)

Super Kit is, quite simply, the best disk file utility there is. No more losing those valuable copy-protected originals (like what's happened to me twice too many times). So far we've shipped over 600 Super Kits and orders continue to pour in.

- **Gnome Speed Compiler** $59.95 (US), $69.95 (Cdn)

This compiler is for BASIC 7.0 on the Commodore 128.

- **Gnome Kit Utility** $39.95 (US), $49.95 (Cdn)

Gnome Kit is a Commodore 128 utility with enhancements for the BASIC editor (like Trace, Find, Renumber, Delete, Auto, etc.) as well as enhanced monitor commands, and floppy disk monitor functions.

**Transactor Disks, Transactor Back Issues, and Microfiche**

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issues are ONLY available on microfiche - these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, with one exception. A single back issue will be $4.50 (US/C) and subscriptions are $15.00 (US/C). The exception? A complete set of 18 (Volumes 4, 5, and 6) will cost just $39.95 (US/C)!

This list also shows the "themes" of each issue. "Theme issues" didn't start until Volume 5, Issue 01. The Transactor Disk #1 contains all program from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1-3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

- **Vol. 4, Issue 04** (Disk 1)  
- **Vol. 4, Issue 05** (Disk 1)
- **Vol. 5, Issue 01** (Disk 1)  
- **Vol. 5, Issue 02** (Disk 1)
- **Vol. 5, Issue 03** (Disk 1)  
- **Vol. 5, Issue 04** (Disk 1)
- **Vol. 5, Issue 05** (Disk 1)
- **Vol. 5, Issue 06** (Disk 1)
- **Vol. 5, Issue 07** (Disk 1)
- **Vol. 5, Issue 08** (Disk 1)
- **Vol. 5, Issue 09** (Disk 1)
- **Vol. 5, Issue 10** (Disk 1)
- **Vol. 5, Issue 11** (Disk 1)
- **Vol. 5, Issue 12** (Disk 1)
- **Vol. 5, Issue 13** (Disk 1)
- **Vol. 5, Issue 14** (Disk 1)
- **Vol. 5, Issue 15** (Disk 1)
- **Vol. 5, Issue 16** (Disk 1)
- **Vol. 6, Issue 01** (Disk 1)  
- **Vol. 6, Issue 02** (Disk 1)
- **Vol. 6, Issue 03** (Disk 1)  
- **Vol. 6, Issue 04** (Disk 1)
- **Vol. 6, Issue 05** (Disk 1)  
- **Vol. 6, Issue 06** (Disk 1)
- **Vol. 6, Issue 07** (Disk 1)
- **Vol. 6, Issue 08** (Disk 1)
- **Vol. 6, Issue 09** (Disk 1)
- **Vol. 6, Issue 10** (Disk 1)
- **Vol. 6, Issue 11** (Disk 1)
- **Vol. 6, Issue 12** (Disk 1)
- **Vol. 6, Issue 13** (Disk 1)
- **Vol. 6, Issue 14** (Disk 1)
- **Vol. 6, Issue 15** (Disk 1)
- **Vol. 6, Issue 16** (Disk 1)
- **Vol. 7, Issue 01** (Disk 1)  
- **Vol. 7, Issue 02** (Disk 1)
- **Vol. 7, Issue 03** (Disk 1)  
- **Vol. 7, Issue 04** (Disk 1)
- **Vol. 7, Issue 05** (Disk 1)
- **Vol. 7, Issue 06** (Disk 1)
- **Vol. 7, Issue 07** (Disk 1)
- **Vol. 7, Issue 08** (Disk 1)
- **Vol. 7, Issue 09** (Disk 1)
- **Vol. 7, Issue 10** (Disk 1)
- **Vol. 7, Issue 11** (Disk 1)
- **Vol. 7, Issue 12** (Disk 1)
- **Vol. 7, Issue 13** (Disk 1)
- **Vol. 7, Issue 14** (Disk 1)
- **Vol. 7, Issue 15** (Disk 1)
- **Vol. 7, Issue 16** (Disk 1)

**Industry News**

The following items, compiled by Astrid Kumas, are based on press releases recently received from the manufacturers. Please note that product descriptions are not the result of evaluation by The Transactor.

**New Books from Abacus**

Abacus Software has published another volume in their reference series of books for the Commodore 128 -- the C-128 BASIC Training Guide. The book is aimed at the user who wants to learn the Commodore 128's built-in BASIC programming language. The book aims to be a thorough introduction with numerous examples to lead the reader from simple to more advanced programming techniques. The suggested retail price is $16.95 (US).

The next book in the C-128 series -- BASIC 7.0 Internals -- was scheduled for shipping in late December '86. The suggested retail price is $24.95 (US).

Another recent publication from Abacus Software is GEOS -- Inside and Out, written by Manfred Tornsdorf and R. Kerkloh. It includes introductory material about GEOS, the Desktop, GEOSWRITE and GEOSPAINT, a large collection of tips for every GEOS user, as well as a description of GEOS internals. The suggested retail price is $19.95 (US). More information is available from:

```
Abacus Software
2201 Kalamazoo S.E.
P.O. Box 7211
Grand Rapids MI
49510 (616)241-5510
```

**Eye-Scan for C-64/128**

Digital Engineering has announced the release of its first product -- Eye-Scan, a video digitizer for the C-64, C-128 and SX64 computers.

Eye-Scan's hardware cartridge plugs into the computer's "user-port" making graphics input simple: composite video in via an RCA jack. Conversion time is approximately 6 seconds per grey level.
Eye-Scan disk software utilizes pull-down windows to accomplish black and white imaging, up to 8 grey levels, image inversion, disk and 1525 printer support. Also included is a programmer’s utility package that allows users to utilize the image capturing algorithms in their own programs. Eye-Scan is compatible with Koala, Doodle! and Blazing Paddles graphic programs.

Possible applications include animation, security, automated process control, pattern analysis, robot vision and text recognition.

Eye-Scan can be ordered for $89.95 (US) from:

Digital Engineering and Design
2718 S.W. Kelly, Suite C165
Portland, Oregon 97201
(503) 245-1503

Spartan now with Apple II compatible disk drive

Effective November 1, 1986 Mimic systems has repackaged the Spartan, the Apple II emulator for the C-64, to include an Apple compatible disk drive. Spartan says the decision to discontinue the DOS card and include an Apple II compatible disk drive was prompted by numerous requests from customers, and the amount of technical assistance required for the installation of the DOS card in the 1541 disk drive.

Cost of the Spartan including Apple compatible disk drive is $329.95 (Cdn). For further information, call 1-800-663-8527 or contact:

Mimic Systems
c/o EDP Industries
*205-1401 West 8th Avenue
Vancouver, B.C. V6H 1C9

Peek A Byte 128

Quantum Software’s Peek A Byte 64, a disk and memory utility for the C-64 programmer, has been upgraded for use on Commodore 128. The new version, Peek A Byte 128, will be available in February 1987.

In addition to all the features of the original product, Peek A Byte 128 presents several other options, such as 80-column display, reading or writing to a 1571 double-sided disk, and converting 1541 single-sided to 1571 double-sided format without harming data already on front side.

An enhanced version of the original product, Peek A Byte 64 V2.0 together with the Disk Mechanic and new manual, will also be available at the same time. Amongst features, added or improved, the manufacturer lists the following:

- read or write up to track 40 even with DOS header errors
- edit sector GCR data
- read “raw” track GCR data
- fast format single or multiple tracks up to 40
- do half-tracks up to track 40
- analyze disk errors

Users who own the original Peek A Byte 64 can order the Peek A Byte 128 upgrade package (including Peek A Byte 64 V2.0 and new manual) for the price of $20.00 (US). For more information, contact:

Quantum Software
P.O. Box 12716
Lake Park
FL 33403

---

**The Commodore Show**

**Fri., Feb. 20, 10:00-6:00**
**Sat., Feb. 21, 10:00-6:00**
**Sun., Feb. 22, Noon-5:00**

**Brooks Hall, Civic Center**
**San Francisco**

- EXHIBITS, EVENTS AND DRAW PRIZES
- NATIONAL COMMODORE SPEAKERS
- SHOW SPECIALS AND DISCOUNTS
- SEE THE LATEST INNOVATIONS IN HARDWARE/SOFTWARE TECHNOLOGY

The Commodore Show is the only West Coast exhibition and conference focusing exclusively on the AMIGA, Commodore 128 PC and C-64 marketplace.

REGISTRATION FEES:
One Day Only—$10
Three Day Pass—$15

For More Information Or To Reserve Exhibit Space Contact

**COMPUTER SWAP, INC.**
PO Box 18906, San Jose, CA 95158
(408) 978-8SWAP • 800-722-SWAP • IN CA 800-252-SWAP

---

Lincoln College
Commodore Computer Camp
with
JIM BUTTERFIELD
and other experts

July 19-25, 1987

Topics include:
- Amiga
- C-128
- Robotics
- Telecomputing
- Additional selected topics

For further information, contact:
Office of Continuing Education
Lincoln College
300 Keokuk
Lincoln, IL 62656
217/732-3155
When you have a job to do, get it done right...
THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $8.95 Per Issue
6 Disk Subscription (one year)
Just $45.00
(see order form at center fold)

Also check out the TransBASIC Disk
Complete with 24 page manual, just $9.95!
CompuServe.
You Don’t Have To Know How It Works To Appreciate All It Can Do.

CompuServe is a computer information service. You subscribe to it. In return, you have access to an incredible amount of information, entertainment, communications and services. Here are a few of the hundreds of amazing things you can do.

COMMUNICATE
CB Simulator features 72 channels for "talking" with other subscribers. National Bulletin Boards let you post messages where thousands will see them. Friends, relatives and business associates can stay in touch through EasyPlex Electronic Mail. More than 100 CompuServe Forums welcome participation in discussions on all sorts of topics. Software Forums help with online solutions to software problems. Hardware Support Forums cater to specific computers. There’s even free software, and online editions of computer periodicals.

SAVE ON TRIPS
With CompuServe’s travel services you can scan flight availabilities, find airfare bargains and even book your own flights online. Plus, there are complete listings of over 28,000 hotels worldwide.

BE INFORMED
CompuServe puts all of the latest news at your fingertips, including the AP news wire, the Washington Post, the St. Louis Post-Dispatch, specialized business and trade publications and more. Our executive news service will electronically find, "clip" and file news for you ... to read whenever you’d like.

INVEST WISELY
Get complete statistics on over 10,000 NYSE, AMEX and OTC securities. Historic trading statistics on over 90,000 stocks, bonds, funds, issues and options. Five years of daily commodity quotes. Updates on hundreds of companies worldwide. Standard & Poor’s Value Line. Over a dozen investment tools.

HAVE FUN
Play all sorts of sports and entertainment trivia games, brain-teasing educational games and the only online TV-style game show with real prizes. Or, for the ultimate in excitement, get into an interactive space adventure.

SHOP
THE ELECTRONIC MALL™ takes you on a coast-to-coast shopping spree of nationally known merchants, without ever leaving home.

So easy the whole family can go online.
CompuServe is "menu-driven," so beginners can simply read the menus (lists of options) that appear on their screens, then type in their selections. If you ever get lost or confused, type H for help. Remember, you can always ask questions online through our feedback service or phone our Customer Service Department.

Before you can access CompuServe, you need a computer, a modem (to connect your computer to your phone) and, in some cases, some simple communications software. Now you’re ready to order. For your low, one-time subscription fee, you’ll receive:
• a complete, easy-to-understand, 170-page spiral-bound Users Guide
• your exclusive preliminary password
• a subscription to CompuServe’s monthly magazine, Online Today
• a $25.00 U.S. usage credit!

To buy a CompuServe Subscription Kit, see your nearest computer dealer. To receive our informative brochure or to order direct, write or call 614-457-0802.
CompuServe. You don’t have to know how it works to appreciate all it can do — for you.

CompuServe®
Information Services, P.O. Box 20212, 5000 Arlington Centre Blvd., Columbus, Ohio 43220 U.S.A.
An IBM/Black Company
EasyPlex and ELECTRONIC MALL are trademarks of CompuServe Incorporated.