CHICAGO B128 USER'S GROUP -- INTERNATIONAL (CBUG, Inc.)
4102 N. Odell -- Norridge, Il. 60634 USA

Bulk Rate
U.S. Postage
PAID
Desplaines, Il
60018
Permit # 296

FORM 3547
REQUESTED

DATED MATERIAL
DO NOT DELAY!!

- -

Return the ENTIRE Page -- including the mailing label. Make corrections as necessary
Please help us out. Print very legibly or type

- -

This issue is scheduled for mailing the end of the week of Feb. 2, 1988. If you don't get it in a reasonable period of time CBUG would be real interested in what excuse your local post-master gives for the delay.

* *

!!! WHEW !!!

THE LAST OF 1987 !

NAME ______________________________

ADDRESS ______________________________

CITY ______________ STATE ____ ZIP ______

THIS IS YOUR SHIPPING LABEL -- PRINT OR TYPE NEATLY

ORDERS OVER 4 DISKS OR PHYSICAL EXAM SHIPPED BY BY UPS WITHIN CONTINENTAL U.S. YOU MUST USE STREET ADDRESSES FOR SUCH ORDERS -- UPS CAN NOT DELIVER TO POST OFFICE BOXES!

Your phone # ______________________

Remit to: CBUG, Inc., 4102 N. Odell, Norridge, Il. 60634 U.S.A. All payments must be in US funds.
IMPORTANT: If your shipping address is different than your membership address, please give your membership address!!

My Membership Address (if different) is: ______ street ______ city ____ state ____ zip

R ck mo ca pr S

Description	Quantity	Stock #	Price	Extension
COMMODORE IEEE to Centronics Adaptor Board (6400 type)		11221	35.00	
CBUG connector type changer for above (changes output connector to standard Centronics "blue ribbon" D shell type		11236	15.00	
!!See pages A19 & A20 for disks & data cases!! MERCHANDISE SUB TOTAL				
Illinois residents add 7% Sales tax to above items only				
CBUG #72 Goceliak's Third Mine (Gold that is)		11982	14.00	
CBUG #73 THE ESCAPE Index (Superbase application) v.Fall 87		11997	14.00	
CBUG #74 SuperOffice Scrubber		12007	15.00	
CBUG #75 Machine Language Index (Superbase application)		12012	9.00	
CBUG #76 The King James Bible, Complete. set of nine disks		12026	40.00	
CBUG #77 Super Teacher - High Capacity. 4 disk set + blanks (Upgrade for $18 -- send the label from your prior version)		12031	20.00	
CBUG #78 Gold Coast Instructional		12045	9.00	
CBUG #79 Fall 1987 Issue Escape Print Files		12168	9.00	
CBUG #M80 CBUG MISC 12-87		12064	9.00	
CBUG #81 David Green's Updage		12079	9.00	
Pre Release #11 CP/M 86 Info & Pgms #4		12083	9.00	
Pre Release #12 CP/M 86 Info & Pgms #5		12098	9.00	
Pre Release #13 CP/M 86 Info & Pgms #6		12101	9.00	
Pre Release #14 CP/M 86 Info & Pgms #7		12116	9.00	
Pre Release #15 Dave Wack's Assortment		12050	9.00	
KNIGHT's 8050 (DOS 2.7) COPY UTILITY		12204	20.00	
PHYSICAL EXAM for the 8050 Disk Drive		12219	35.00	
PHYSICAL EXAM for the 8250 and SFD-1001 Disk Drives		12192	35.00	
BEELINE v2.1 Telecommunications Program		12280	40.00	
Recopy fee remittance		12401	5.00	
BALANCE FORWARD FROM PAGE 2 RIGHT HAND SIDE OF FORM -------->				
BALANCE FWD FROM DISK & DATA CASE ORDER FORMS, page A20 --------> <North America addresses only!!!>				
MERCHANDISE SHIPPING AND HANDLING CHARGE --- ALWAYS INCLUDED, NO EXCEPTIONS				$ 2 . 0 0
SUBTOTAL			$	
Free Will Contribution to CBUG.		12416		
Extra Copy of this issue,		11555	6.00	

BY CHECK ___ BY MONEY ORDER ___ TOTAL REMITTED $ ____________

NOTE: We endeavor to ship against money order within 3 business days of receipt. Others delayed 14 to 18 days.

THE CBUG ESCAPE

EIGHTH ISSUE

Fall 1987

* *

THE CBUG ESCAPE is published 4 times a year, more or less, by the Chicago B128 User's Group - International (CBUG, Inc.), an international membership organization in support of applications and usage of the Commodore B128 Computer. Note that some issues are combined in one publication.

CHECK YOUR ADDRESS LABEL FOR EXPIRATION DATE. The expiration date is to the right of the postal code in the form of YYMM, eg. 8712 indicates an expiration at the end of December 1987. CBUG will be unable to mail publications without renewal. PLEASE KEEP YOUR MAILING ADDRESS CURRENT. Renewal invoices will be mailed early Feb. 1988.

Publishing address: CBUG, Inc. c/o Norman Deltzke, 4102 N. Odell, Norridge, Il. 60634 USA

Cover price this issue: $6.00. 1988 subscription rate: $14.00 (bulk rate, U.S. & possessions ONLY); $20.00 (first class, U.S. & possessions, Canada & Mexico); $21.00 (surface <boat> first class, all others); $35.00 (small packet rate air mail, any country).

* *

TABLE OF CONTENTS

CBUG SCRATCH PAD

By: Norman Deltzke

Jan. 17, 1988

Well we are catching up a little -- this issue will mail only 5 weeks late! A serious effort is going to be made to get 1988 issues out such that each quarter's issue will mail by the end of the month after the quarter's close: i.e. April 30, July 31, Oct. 31 and Jan. 31, 1989. Our ability to do this is dependant on many things. One of the most critical is receiving articles and library disks well in advance of publication. It is easiest if they flow continuously but when we get about 6 weeks from publication, it is critical that the great majority of that issue be on hand.

This issue has gone together rather smoothly. Most of the materials were in by mid December and with the holidays there were a few extra days I could devote to putting this issue together. We all owe a great deal of thanks to our authors and library contributors. Tony Goceliak, John Berezinski, Clyde Northrup, Marilyn Gardner & Ed Rhyner to point out a few who obviously donated hundreds upon hundreds of hours to our collective efforts. No matter what your level of expertise is, it is hard to immagine anyone who can't find something useful to write about or lend a helping hand to. If you can't think of anything to add, atleast help our regular contributors by suggesting topics.

As usual the "in box" is now empty. What's in the next issue is up to each of you. How about trying to out do this one.

re SEI: While communications remain operable between this office and Mr. Cotter, definitive information was not being received. A number of members have received their 8050 drives back from SEI/Cotter et al. To date they have performed as advertised. The only exceptions were a failing not related to the SEI work (which was not repeatable) and a problem believed to be minor shipping damage. Mr. Cotter claims 16 machines out of 32 have been shipped. Today he provided a list which is not yet fully verifable. Members who receive drives back should contact CBUG so the list we are maintaining remains accurate. If you don't have your unit back yet, you have to push and push hard.

Coprocessor: Work in that area is progressing on several fronts. Not as fast as we would prefer, but then it has to be done evenings and weekends. Many family members express concern as widows and orphans to an orphaned computer.....

A compliment: Some weeks ago Gerd Schumacher, the head of our sister B128 group in Cologne West Germany called asking if I could secure a letter of permission from CBM to allow their group to purchase a quantity of the mechanical guts of the SFD drive so that they could build 8250's for use in the group. Those guts are now surpluss but can't be sold due to CBM proprietory design elements. The answer that came back from CBM legal was a truly backhanded compliment. CBM would release a modest quantity but no more BECAUSE: CBUG thru a substantial number of our members is believed by CBM engineering to have the ability to take those drives and build a quality consumer market drive and thus go in competition directly with CBM. Yes, surely we do have that type of engineer prowess in the group though doing something of that overall magnitude would not happen.

EDITORIAL POLICY QUIRY

Lastly, we have some far reaching decisions to make. Last issue the concept of Goldware was introduced and has met with mixed reactions. Unfortunately members usually do not communicate, but we now need as much input as possible from both our reading/library subscribing members AND most critically from our past, present and future contributors. NOTHING IS CAST IN STONE, and nothing permanent has been established. Don't mistake what is being committed nor "done" at this time. The time is for nothing more than trial balloons, experimental efforts, and careful deliberation.

It is well known that the majority of our members have neither time nor skill to go digging thru materials, trying to decipher Library listings, etc. This has been one of the largest sources of complaint to CBUG from day one. The efforts now being structured are to help those in need rather than create an authoritarian monster. To date CBUG has run without any material editoral interferance. This in great part because I don't have the wisdom of Solomon to know how to cut up each author's baby. Now I'm faced with a discussion which could threaten "puree of CBUG."

As a direct result of those comments received, Goldware henceforth is to be known as EASYWARE, which far better describes the true intent of the project -- that members can be informed which disks/programs may be applicable to various levels of convenience of use. However, questions have been raised whether subjecting incoming or issued materials to "book reviews" might offend or irritate authors such as to impare the flow of important information into the library. Setting forth rules and standards seems to be nearly impossible under a broad based concensus. It is appearing that the appropriate course is going to be one of voluntary softly stated guidelines and cooperative effort between consenting authors and CBUG volunteers.

When we get into the issue of repairing, improving, organizing, the issue becomes hotter. The sensitivity evolves not only of pride of authorship and/or not feeling comfortable with criticism, but continues into the risk that alterations may create unknown bugs and maybe even legal question of consequential damages. While the type of work envisioned should not do anything offensive, such opinions are subjective.

Whatever may be done, the authors rights are foremost. Nothing will be done without permission and if possible participation of the CBUG author. While I maintain absolute control over what eminates from CBUG, members rights of free speach and privledge of authorship are foremost. Rarely have articles or disks had any editing or expungement applied. Today I've decided to withhold from publication atleast 6 pages of articles and comments on this subject. I appologize to all those who rushed these materials in.

Everyone is requested to contribute to the body of opinion to be considered in trying to serve all interests involved. Anyone with opinions and suggestions is strongly requested to write them out. Concisely, neatly, and very legible (preferably typed/word processsed). They are to be sent to CBUG c/o Marilyn Gardner, 1630 Madison St., Evanston, Il. 60202. Absolutely no phone calls on this subject. Any authors who believe they may have a vested interest in the deliberations should let us know. The quantity of inquiries demonstrating the need of review and repair functions suggest we try dilligently to make such an effort operate -- but we have to satisfy all seemingly conflicting concerns.

ECONOMIC MATTERS, ETC

Atleast 300 members have made financial donations to CBUG in the last year -- most were a few to ten dollars. A few have been as high as $50 and $75. This generosity has kept CBUG from digging into my pockets again to maintain our level of service. Thank you all for the financial support.

We have not increased subscription dues for 1988 though they did not cover the costs of operation in 1987. To which end we need to ask for contributions from those members who are able and have the opportunity to send one in. Come 1989 renewals the rate increase will likely be $5.00 plus allowance for expected postal rate changes.

So many members renewed last quarter an increase would have been a nightmare to administer. Come ~~next~~ next fall, please wait for the renewal invoice as this method makes our paper flow less of a tribulation. Invoices will be mailed first class about Feb. 8. Several dozen members renewed but sent less than the proper fee. We would appreciate the balance due.

The US Post Office is threatening to further curtail bulk mail service by lengthing delivery time. Already they are unable to come anywhere near their aspoused 2 week time, and appear to loose an inordinate number of copies. You may wish to renew using first class mail.

◆◆◆◆◆◆◆◆

NOTES FROM THE UNDERGROUND

by: Angel M. Matos

Copyright (c) 1988 Angel M. Matos

Hello again, and the Happiest of New Years to one and all. Well folks it been a while since I put together some ramblings for The ESCape, and there are some reasons behind that.

<*> All of my columns have ended with a request for 'feedback' regarding the content and 'understand-ability' of my ramblings. Occasionally, I'll get a note (even some Christmas cards, Thanks!!!!!) stating that the writer appreciates my attempts to inform. However, no clues as to the needs of the masses. Those in desperation consult via MaBell, the confused and curious shout on Delphi, but still, no word from the masses.

<*> Many of The ESCape's authors have received request (for personal response or inclusion in future articles) for 'advanced' information, as well as request for elaborations on 'undefined' computing 'basics'.

<*> Norman is always saying, "they want to know about 'the basics'." He's mentioned that he has received letters from members asking that The ESCape 'authors' address 'the basics' of computing. The letter writers, unfortunately, never specify what they mean by 'the basics'.

<*> In the Summer 1987 issue of The ESCape (page 21) my friend across the Hudson River, Anthony J. Goceliak addressed these issues, and Norman added his comments (page 22) regarding this situation.

The simple reality is that we, and our merry batch of helpers, will gladly handle your request, and tackle the problems you pose, all in due time of course, BUT ---- BUT you have to help!!!!! You have to be specific as to what it is you want to know. What it is you want explained. What is it YOU mean by 'the basics'?! You have to take the responsibility of being specific with your request, or suggestions. You are the first step in the formula.

YOU have to keep in mind that we all do this because we enjoy it, not because we are on Commodore's payroll, or because we are making our first million off of our efforts. Quite to the contrary, for most of us, our time devoted to this effort, means money lost.

In my particular case, I need you to tell me what it is you want me to write about. As I have mentioned on prior occasions, I am a Computer (and a Consumer Electronics) consultant. I am therefore best suited to address issues relating to:

*software choices and implementation,
*'Power-usage' of software,
*support systems (modems, printers, power filters, conditioners & 'UPS', monitors, cables, etc.,
*computer to computer file transfers,
*computers in general,
*etc., etc., etc.

NOTE: I am not a programmer, as noted in The ESCape's 'Yell for Help' listings.

Please understand that unlike most of the membership, I am not a "one-computer" man. As a computerist I personally use(d) twelve systems, using ten different operating systems. At work (yes, I also have a regular, non-computer related j-o-b), I use three systems and two more, distinct, operating systems. Therefore, as someone who has dealt with 'the basics' of a wide range of computers and operating systems, the phrase "tell me about 'the basics'", doesn't tell me much, and ask a whole hell of alot. I need you to be specific!!!!!

Anyway, after all of this, the bottom line is that "you" have to communicate with "us". So please send us your comments, suggestions, requests, questions, and complaints. Whatever, just communicate with us!!!!!

As always, I can be reached via various routes:

-The Delphi Network, via EMail, or on CBUG OnLine.
 Username: AMATOS

-On CompuServe via EMail. ID #75146,2224

-Courtesy of the US Snail at:
 Mr. Angel M. Matos
 3176 Decatur Ave. #6E
 Bronx, New York 10467
 Attn: CBUG

-Via MaBell on (212) 231-6028, usually available:
 *Evenings: 7:30pm to 10:00pm, eastern.
 *Weekends: 12noon to 7:00pm, eastern.
 *Other time, talk to my answering machine.

If you request that I return your call, specify that you WILL accept a collect call, and note the best time(s) for me to call back.

REMEMBER: Join us on CBUG OnLine

Well 'till next time,
Keep On Truckin' /// Angel
/30/

A footnote: To join us on Delphi, first call Delphi/American Videotext at 800 544 4005. Obtain the network access numbers for your area -- these are the special numbers which you call using your modem to access Delphi with only a local phone bill. If you have no local number, get several surrounding out of state numbers as interstate rates are usually lower than intra-state. As Delphi to explain to you how to sign on to each network up to the point where you specify "DELPHI". At that follow these directions: When asked for "carrier?" respond DELPHI, when asked for "user name" type "JOINCBUG", when asked for "ID number or password" respond "NEWMEM". Of course do not type the quote signs. You may type everything in lower case as well.

The system will then ask a series of questions, as you to choose a password, take your credit card number for billing purposes or offer alternative means of payment, etc. The next business day they will phone you to confirm your subscription. This procedure saves you over $50.00 in subscription fees, gives you a couple of hours of free time, and a deal (worth taking) on a user's manual.

...Norman

Another hint for Delphi Users. When you sign-on to the Delphi system, you receive a number of menus in succession asking you what you want to do. To get to the CBUG area, you can respond with the answers to three menus at once. The first menu response would be "GR", the second can be either "CO" for Commodore or "CB" for CBUG. And the third is "CB" again for CBUG. Conveniently you can type all three at the first menu and save a bit of time "GR CB CB".

If you don't plan on regularly using any other part of the Delphi Net, you can make this automatic. From main menu got to "SET", then "DEFAULT" and enter the the above code "GR CB CB". If on occasion you need to get back to the main or other menus, simply type " 'up arrow symbol' Z" or "EX" for exit.

To sign off, go to any of the principle menus, not necessarily the main DELPHI menu, and type "BYE". Also remember when in the DELPHI-MAIL area to DELete each piece of mail after received or by the end of each month to avoid storage charges.

And if you think the above is advertising, well my handle on Delphi is CBUG!

...Norman

◆◆◆◆◆◆◆◆◆

UPLOADING TO DELPHI HINTS

From: BOS1B::ORPHAN

<<This is text transmitted via Delphi. ORPHAN is infact Col. John Wright. On Delphi everyone is known by their "handle".>>

One of the problems with using a commercial bulletin board service is that it costs money. Delphi is one of the least expensive around at about $6.00 per hour of on line time.

Most B users I assume use the E-Mail facility on the boards they access. One way to reduce the cost of this process is to compose your messages using Superscript, while off line, then use a feature of Beeline to download directly into the text of the message you want to send. <<There is also a small word processor in BeeLine, & Liz Deal's memo maker.>>

Here's how you would do just that on Delphi.
1. Use Superscript to compose (spell check if you are like me and can't spell) and save the file.

2. Next load and run Beeline.

3. While at the Number 1 menu (that's the one that appears before you as soon as Beeline comes up) select Option number 6; "TXMIT Disk File (XON/XOFF)" (NOTE: DO THIS BEFORE YOU LOG ON TO DELPHI OR ANY OTHER SERVICE, I TAKES A FEW SECONDS TO SET UP THE FILE FOR TRANSMISSION, NO SENSE IN PAYING FOR THAT TIME)

4. You will be presented with another menu. Select option 1. to select the file for transfer. You will be asked what disk the file is on and whether or not you want to see that disk's directory. Follow the prompts and provide the file name that you want to transmit.

5. When you have finished identifying the file, you will be back at the menu. (The one that asks: 1: Specify Filename 2: Transmit a file) Press the return key and you will return to main menu number 1.

6. You are now ready to establish a connection with your computer service. I'll use Delphi as an example. Once connected select the E-Mail option. (On Delphi, enter "Delphi mail". At the next prompt enter "Mail" You will be left at the "MAIL>" prompt. <<If entering from inside the CBUG area, go directly to "MA" for mail. DELPHI MAIL will also allow access to other carriers, telex, etc.>>

7. However you get there, you need to be at the Mail prompt to continue. Do whatever is needed to provide the address or addresses of the person or persons you want to send the letter to. (On delphi use "SEND" at the "Mail>" prompt. Delphi will ask for an address <<the handle of the user to whom you are writing>>, enter it here.
Next will appear a "subject" line, just type the subject of your message. Next a short help note will be displayed and instructions on how to end your message. This is where you need to be to download the file you selected previously.

8. Now escape back to Beeline by pressing the shift and British Pound key at the same time. Select 6 again, and at the next menu select 2. You will be asked three questions. I press the return key to the first two and the space bar to the last one. (These are the default options.)

9. Your file will transfer just as if you were typing it on Delphi yourself, but very much faster. When the transfer is complete, you will be left in the text portion of the message. Type the control and z keys at the same time and presto your message is sent.

10. You will then be back at the "Mail>" prompt. Type "exit", you will exit to the "Delphi Mail>" prompt, and type "bye" and you are logged off Delphi.

1. I believe this procedure is referred to as "Direct Upload" and is used by some computers like the Atari for file transfers. It is very handy for uploading ASCII files to any on line message composer or text editor. It is also quite fast. I will send this to Norm via E-Mail and it should only take about 3 minutes vs the 30 it took to compose. And I'm not under any pressure to do it fast because it is not costing me anything to use Superscript.

12. There are a few words of caution. Delphi cannot accept any more than 255 characters before it needs a carriage return. So I insert one every other line. Delphi also doesn't like some characters like the shifted 2 key (the "at" sign), so try not to use anything but letters in your messages.

13. That's about it. It's fast and easy, and best of all, it cuts down the Delphi monthly bill.

Good luck and good computering.

◆◆◆◆◆◆◆◆◆

GET in SUPERBASE

by: Clyde Northrup

While working on the ESCAPE INDEXER program, I discovered something that many of you already knew. In using the GET command in the applications program, I use a numeric string 'n'. Later, while testing the program for bugs, (there were many but I hope I got most of them critters) I entered a non-numeric character 'v' and was thrown out of the program with an error. I fixed the problem by GET-ting a text string 'n$' and then converting it to a numeric with 'n = val(n$)' and all was well. I had the numeric I needed and if you accidentally enter a text string instead of the desired numeric then you will not loose control of the program. I was able to trap out the erronious entry and give you another chance.

Clyde Northrop is a U.S. Army Chaplain from the Southern Baptist Denomination. He is 51 years old, married, with 3 children and 2 grandchildren (& one on the way.) He has been in the army for 21 years as a chaplain. His family has four horses and they live south of Atlanta in Fayetteville, GA. He now works at Fort McPherson as the Theater Army Chaplain for Third U.S. Army.

ESCAPE INDEX PROGRAM

I have long felt a need to be able to go to previous issues of ESCAPE and look up information to help me with a particular problem. When I had only one issue there was no problem. When I had three issues I began to find that I was spending a lot of look-up time and often got nowhere in my hurry and frustration. Just when I needed a bit of information, I could not find it. I wanted to get a thorough index to ESCAPE but I knew it was a large task, so I kept putting it off.

Recently I received help in the form of encouragement from the DELPHI crowd. I, therefore, got off my duff and got to work, a lot of work, and got the job done almost to my satisfaction. However, I do plan to continue to update the index and the application programs on the INDEX disk.

The INDEX application programs are almost as simple as I know how to make them and still have them do the job I think each of you desire. If you have ideas for improvement, please write to the address in the help files on the disk and I will try to incorporate your suggestions in future editions. I would like to put out a new edition each year, if the Spirit moves me.

I always believe that anything I do is not quite good enough. Therefore, I want to hold on to this helpful disk until I have tested it over and over again. However, it would never get published, if I wait for my complete satisfaction. So out it goes to CBUG in the present condition, with whatever warts there happen to be. Let me hear from you if you have any real problems.

HOW TO USE ESCAPE INDEX DISK

NEWCOMERS: These application programs require that you first load SUPERBASE master program into your computer. If you are new at this, then just place the SUPERBASE master in you drive 0 and close the door. Hold down SHIFT and press RUN. Wait for the program to load. (While it is loading you might begin to review your SUPERBASE Manual). When the drive stop turning you will see a prompt on the screen to remove the master disk and either create a data disk or insert an already created data disk into drive 0. DO NOT ELECT TO CREATE A DATA DISK! You might erase your applications disk.

EVERYBODY: Insert your applications disk into drive 0 and press RETURN. This will run the start program on the disk and reconfigure the F KEYS and load the applications menu program from which you can elect to enter information from recent ESCAPE issues or from other magazines you wish to index.

NOTE: The applications disk already has an index which is current as of November 1987. I suggest that you make a two copies of the applications disk. One should be a backup copy and the other can be used to input information from other sources. I would delete the ESCAPE information from this disk and use it for an "all others" file. Each year when the new ESCAPE INDEX is released, you will be able to use the new version without compromising your personal file input. If you wish to continue to input your own ESCAPE index each issue, then you will not need to heed the above.

SEARCHES: Searches may be performed with ESCAPE INDEX either from program control to make it easy for newcomers or they may be done with the full range of the powerful Superbase FIND & SEARCH commands. The programs are MENU and PROMPT driven. I hope even the newcomers will have no problems, but one can never tell. Let me know if you do.

I tried to keep a consistent manner of entering information, but did not entirely succeed. When I entered keywords on which program searches are done, I used B128 instead of B-128 or B 128. I used C64 instead of C-64. I used RS232 instead of RS-232C, etc. In doing a search, I suggest that you key in on, for instance, 128 as a keyword. It will then give you all instances of B128, C128 or just 128. You can then select the articles you desire to read. In doing a search, I suggest that you enter as a keyword just enough of a phrase to get you the desired result. In looking for Superbase, I just type in 'base' and the results are fine. In some instances, where room was limited for keywords I might have use sb for superbase or ss for superscript. If you are not getting all the info you desire from 'script' or 'base' try a search on 'sb' or 'ss' and see what comes up. Searches take a while because there are many records. I did not enter a record for each keyword, because the searches would be so long that they would defeat the purpose of the INDEX.

You need not re-search each time you look up a keyword. For instance, many searches are already done. A list of them may be obtained by pressing F Key 'F19'. You need not re-do these searches until you add more information to the index which concerns that keyword.

ARRANGEMENT OF FILES: The files are located in the directory in a certain order so that you can browse them from the SELECT menu without doing searches. They are arranged, however, only on the dominant information in an article. Therefore, you will have to do a search if you want to see casual mentions (sometimes very important) of a keyword.

You can browse the files from Superbase MENU 1 by using SELECT. Select the first file by pressing 'f' within the SELECT menu. You will see that the 'key' starts with an A. The first-letter code is given on the file so that you can select information at will. To select, for example, TELECOMMUNICATIONS, just SELECT KEY and then type in the letter 't'. When you press RETURN the drive will fetch you the first file on telecommunications. You can then browse the file by pressing 'n' for NEXT until you have seen the whole file on telecommunications. If this sounds too complicated for the newcommer (I hope not) then you can stay with the prompt driven program and find your information by following the prompts.

The second character of the KEY field is a number which tells the type of article or information by using a 1-9 designator. The 1 would stand for an article or review. Each number is explained on the file format, so you should have no trouble using these numbers to create specific searches if you are familiar with Superbase. If you see a need, I could create applications programs to do the job for you.

I hope that the ESCAPE INDEX applications disk will be one of the most helpful tools in your library. You not only can use it, but since I did not protect the programs (it can be done in Superbase) you can see just how they are written and use them as an example to create your own programs. I even left one of my own self-help programs on the disk to show you another example of what can be done with Superbase. It is a COMPUTER SHOPPER price comparison program designed to help me pick out the system which would best meet my needs for the lowest cost. (No, I do not plan to leave the B128 but my employer requires that I use MS DOS at the office).

Great Computing, Merry Christmas and a Happy New Year!

Clyde Northrop is a U.S. Army Chaplain from the Southern Baptist Denomination. He is 51 years old, married, with 3 children and 2 grandchildren (& one on the way.) He has been in the army for 21 years as a chaplain. His family has four horses and they live south of Atlanta in Fayetteville GA. He now works at Fort McPherson as the Theater Army Chaplain for Third U.S. Army.

```
0 "sb7 data 1         "    Name of Disk
7    "loader"              Allows you to read files
1    "COMPUTER"            Superbase Database
8    "start.p"             Configures SB for you
6    "prices"              FILE for COMPUTER SHOPPER
5    "p.r.125column.p"     report program for CS
6    "escape"              file for ESCAPE INDEX
5    "h.script"            search file on "script"
7    "hprice"              HELP file for CS program
14   "p.e.escape.p"        ENTER program for ESCAPE
2    "p.r.price.p"         REPORT program for CS
5    "menu.p"              MENU  for applications
3    "dir.p"               a Directory program
10   "hsearch"             HELP file for EI SEARCH
6    "p.r.search.p"        REPORT program for EIndex
3    "instructions"        Application Instructions
8    "hhelp"               HELP DIRECTORY
8    "henter"              HELP file for EI ENTER
8    "hindex"              HELP file for EI INDEX
8    "h125"                HELP 125 column report
8    "hdir"                HELPS DIRECTORY
1    "p.p.sort.p"          CS price Sort Program
4    "h.8050"              seacrh file on "8050"
22   "hsearch&report"      HELP on Search & Report
7    "hfuture"             help screen announcement
8    "hkey"                HELP - F Key Directory
3    "h.modem"             search file on "modem"
6    "h.disk"              search file on "disk"
3    "h.rom"               search file on "rom"
4    "h.base"              search file on Superbase
3    "h.cbug"              search file on CBUG
30   "ESCAPE read file"    Article on this program
  1158 blocks free.
```

```
LOAD & RUN PROGRAMS with F11, F12, F13, F14        Get Directories & Other Helps with F16 thru F20
INDEXER   INDEXER   SHOPPER   SHOPPER              SORT    D I R E C T O R Y       D I R E C T O R Y
 ENTER     SEARCH   125 COL   80  COL              LISTS   DRIVE 1    DRIVE 2      HELPS     PROGRAM

      F KEY template pattern for use with SUPERBASE APPLICATION PROGRAM

  F1        F2        F3        F4   ESCAPE INDEXER PROGRAM       F7        F8        F9        F10
                                                    Cut out center at lines to fit over F KEYS

 ENTER     SEARCH    125.COL   80 COL              SHOPPER   F KEY     DIR        INDEXER   SUPERBASE
 HELPS     HELPS     HELPS     HELPS                HELPS    DIR       DRIVES     HELPS     HELPS
```

KEYTRIX BUG

By: Norm Deltzke
for: Liz Deal

In the course of preparing to release a new version of copy-all which would operate on a factory B-256 (with the CBM 256 ROMS), it was discovered that there was a bug in the last two releases of Keytrix -- which are very considerably upgraded from the earlier versions. The bug is a mere typo, but causes the program to crash in line 180. Here is what you do to fix it:

THIS INFORMATION APPLIES ONLY TO
LIZ DEAL's UTILITIES CBUG #17a and
LIZ DEAL's B-KIT CBUG #60

Make a copy of the disk before starting and then make changes on the copy rather than the original.

```
First load file "tsk"
type       list180 RETURN
move the cursor towards the right hand end of line 180 and
change "myb" to "tsk" and type RETURN
type       scratch "tsk" RETURN (wait for disk drive
type       dsave "tsk" RETURN

   Now let's try the program
type         run RETURN
at menu      1   RETURN (that's numeric one)
                 RETURN
```

Try using the F9 and F10 keys for starters.

To make absolutly sure we did everything right, turn your B128 off and on again to cold start it. Now load "tsk" and RUN it. If it performs as above, all is well.

NOW, an important point COPY-ALL B-128 was also upgraded and the version found on #17a and #60 do not work in many machines. If you load and run it you get an error in line 96. Until the bug can be found, load the fixed Keytrix first then run Copy-All. It will work fine in the Keytrix environment as Keytrix has taken over and reset all the pointers in the B to its own standards.

Liz and CBUG apologize for any inconvenience these errors may have caused you.

LITSEARCH FOR LAWYERS

by: Harlan A. Schmidt, esq.

CBUG attorneys will be pleased to hear that Bing Hart has adapted his LitSearch software to accommodate the LEXIS legal research available through Mead Data Central. Now lawyers using the B128 may use LEXIS through its regular licensing arrangement for the regular $125.00 monthly subscription fee.

The ABA has just announced that it will make LEXIS available to subscribers of ABA/net. The membership group arrangement allows small firms to use the LEXIS service at rates that significantly reduce their overhead costs.

Alternatively, lawyers may join a group agreement. According to Mead Data Control, more than 1600 small- and medium-sized firms are now using LEXIS through various bar associations and law libraries in 25 states. I belong to a group agreement sponsored through the University of South Dakota where each member is charged an additional surcharge per search of $5.00 up to a maximum of $25.00 per month as a membership fee.

In addition to the membership fee surcharge, the member is charged for the actual cost of each search. The cost of each search varies according to the library searched, but my searches generally cost $22.00 each. In addition, the connection time is charged at $20.00 per hour and WATTS line time for the telephone charges is charged at $17.00 per hour.

Included with the membership is access to Autocite and to Shepard's services and NEXIS, a service providing access to full text of newspaper, magazine, and wire service publications.

A Hayes compatible modem is necessary to access Mead Data. Mead Data's technical group has a list of modems which they have found to be compatible. Those which will not work are Leading Edge, most Ventel and some Multitech modems. The software supports 2400 baud. Unless you use the WATTS line you may find that your telecommunication net may not support 2400 baud. LEXIS advertises that it's available 24 hours a day but I was unable to access an early Sunday morning search.

Mead Data Central will train you in its own training center for a $75.00 charge. In our area, Minneapolis, Omaha and Denver are training areas. The $75.00 training charge includes one free hour of search time. That free hour of search time can be used on any library so I would suggest that you jealously protect your search time. During my training, the instructor chatted on about personal matters while connected, allowing the free practice time to click away.

If you are interested in the service, you may acquire the software entitled LitSearch directly from Bing Hart. Please call him to purchase the software, which is priced at $175.00:

Bing Hart, Ph.D.
1408 East 19th
Lawrence, Kansas 66044
Telephone: 913-843-7668

With no prior experience with data banks, I was able to use the LitSearch program successfully on my first search without reading Bing's manual. (It is on the disk available for your own printing.) I did encounter a subsequent problem which Bing eagerly and promptly cured. His interest and support have been outstanding.

LitSearch was not designed for attorneys, but its regular features work well with LEXIS. It has an unlimited file length. Your search is prepared offline through Superscript. It has more options than one is apt to use. Its cursor is online. It automatically dials, logs on and logs off. It hardcopies the search. It's easily reprogrammed. It tracks expenses and has easy to follow prompts.

If you have no group to join, you may join the South Dakota group. To join, contact:

Assistant Professor Mary Jensen
Assistant Law Librarian
USD School of Law
414 East Clark Street
Vermillion, South Dakota 57069
Telephone: 605-677-5259

I understand that the South Dakota law school offers membership to lawyers situated outside this sovereign state! I am available to answer queries which I would ask that you direct to my residence where I keep my LEXIS material. Telephone 605-642-3122 between 6-9 o'clock p.m. mountain time or any time Saturday or Sunday. I encourage lawyers to add their name to my attorney user's group.

LitSearch is not an elementary program. It's a sophisticated tool. I recommend the LitSearch software to all B128 data center users, whether for professional or personal use.

Harlan A. Schmidt, Esquire
P.O. Box 1048
Spearfish, South Dakota 57783
Phone: 605-642-3122

◆◆◆◆◆◆◆◆◆

BASIC 4.0+ TUTORIAL (ABRIDGED)
by Warren D. Swan

3 BASIC FUNCTIONS

3.1 Print Fields and Comma [Basic Output 3]

Earlier we learned how to print out numbers and messages directly, and then how to print them out from variables. We also learned how to use semicolon between the items in a print list so as to cause the items to be printed "right next to each other" or in juxtaposition. We also learned that a semicolon at the end of a print statement tells BASIC not to return the "carriage" to the beginning of the next line.

One thing that was not mentioned is that BASIC really doesn't need the semicolon in all cases when it is used to separate print items. For example:

```
  print "The answer is: "a
```

is perfectly alright. BASIC "knows" that "The answer is: " is one print item, a string message, and that the a is another print item, a real variable. In some cases we must use the semicolon. For example, if we want to print the value contained in the variables j and q$ we could NOT use:

```
  print jq$: rem wrong
```

because BASIC would think we meant to print the value of the variable jq$. Even putting a space in does not help, since BASIC ignores spaces (outside of strings):

```
  print j q$: rem still wrong
```

The only way to do it right here is to use j;q$. However, if we wanted to switch the order in which j and q$ are printed we COULD use:

```
  print q$j: rem ok
```

since now BASIC can "see" that the $ ends the q$ variable, and that the j must be another variable to print.

In all the cases where you leave out the semicolon, BASIC still assumes that that's what you meant. Of course, you must still use the semicolon if you want to cause BASIC to leave the cursor at the end of the printed items:

```
  print "What is your age" :rem basic does not assume an
ending ; here.
  print "What is your age";:rem you must supply it.
```

BASIC also has other ways to make sure that you can get your output to look the way you want it. If you want to print out some simple tables, BASIC has something just right for it. But to understand this next BASIC feature, we will have to be introduced to a new concept - that of the print field.

BASIC divides up the screen output line into 8 ten-character print fields, like this:

```
field 1---field 2---field 3---field 4---field 5---field 6

....!....1....!....2....!....3....!....4....!....5....!..

                           <<right end truncated for ESCAPE>>
```

When you use a comma (,) in a print statement instead of a semicolon, BASIC will skip to the next print field, also called a print zone. Let's use a for/next loop to print a table of the integers -4 to 4 and their squares and cubes:

```
 50 print "Number:","Square:","Cube:"
 100 for num = -4 to 4
 110 print num, num*num, num*num*num
 120 next
run
Number:    Square:   Cube:
-4          16        -64
-3          9         -27
-2          4         -8
-1          1         -1
 0          0          0
 1          1          1
 2          4          8
 3          9          27
 4          16         64
```

Since we used commas in the print statement, BASIC skipped over to the next print field each time to print the items after the commas. We used a space after each comma in the program just for readability. Notice that BASIC still prints numbers the same way, regardless of whether we use semicolon or comma. It still prints a minus sign or space, the digits of the number, and a cursor right.

The comma has one property similar to semicolon. If it appears at the end of a print statement, BASIC will move the cursor to the next field and will leave it there, rather than finishing the line with a carriage return like it usually does. We will see an example of this shortly.

Use of the comma and the print field go hand in hand. If you don't use the comma in your print statement, BASIC will simply forget about print fields. Whenever you do use a comma, BASIC will simply skip to the next print field. It can easily find the next print field by looking at what column it's currently in and finding the next column whose number is divisible by 10. This is a simplification but it works.

3.2 TAB(X) and SPC(X) [Basic Output 4]

The print field is a very handy thing and allows us to construct some quick tables. In many instances though, we want to construct tables that don't fall into the print fields exactly. So BASIC gives us a couple of functions to allow us to pretty much do what we want.

As this is an abridged version of the tutorial, we will not discuss these functions here. The tutorial disk fully explains what tab(x) and spc(x) do, when to use them, and what to watch out for. It also provides examples of their use, including a quicky program to draw a diamond.

3.3 Functions (Built-In) [Handling Data 6]

BASIC 4.0+ offers many functions that are very useful. The more difficult the function, the more we will describe it here. Many functions need very little explanation. When in doubt, try a function until you understand it.

All functions can take expressions as their arguments. That is they are not limited to just constants (-13 or "message") or variables (ac or st$). The order in which the arguments are placed in the parenthesis is significant. In the following descriptions, those arguments that must be string values will contain a string variable in the corresponding position, and the arguments that must contain numeric values will contain a numeric variable in the corresponding position. Integer values may be used where a numeric value is required. BASIC will convert such integer values to their corresponding real value first.

3.3.1 Numeric Functions

For the sake of describing the trigonometric functions, assume that the variable pi is set to the value 3.14159265357989... In fact the PI key (next to the return key) can do this for us. Unhappily, the Commodore printers cannot print the PI symbol when they are in the lower case mode.

All numeric functions result in real values. Of course, BASIC will convert the result to integer, if possible, if assigned to an integer variable.

ABS(X)

Finds the absolute value of the argument expression. abs(-4) is 4. abs(13) is 13.

ASC(X$)

Finds the ASCII value of the first character of the argument expression, which must be a string expression. If a$ = "-12", then asc(a$) is 45, the ASCII code for "-". If a$ is undefined or set to "", then asc(a$) will cause an illegal quantity error.

ATN(X)

Finds the arctangent of the argument expression. atn(1) is pi/4.

COS(X)

Finds the cosine of the argument expression. a = cos(pi/2) sets a to 0.

EXP(X)

Finds the natural antilog of the argument expression. That is, it raises e (approximately 2.718281828) to the power of the argument expression. So, e = exp(1). EXP is the mathematical inverse of LOG.

FRE(X)

Reports the number of bytes not being used in the bank whose number is the argument. The tutorial disk explains this function more fully.

INSTR(X1$,X2$) or INSTR(X1$,X2$,X)

The first 2 arguments to this function must be string expressions. The optional third argument must be numeric. This function searches through string X1$ to find string X2$. If it does not find it, the result will be 0. If it does find it, the result will be the position in X1$ where X2$ starts. For example, if q$ = "cab", then:

```
    print instr("abccbabaccab",q$)
     10
```

The string "cab" occurs in the first string starting at the 10th character, in fact it is the last 3 characters. Unlike TAB(x), which starts counting column positions at 0, INSTR starts counting characters at 1.
If the optional third argument is used, it must be a character position to start searching from; for example, if q$ = "ab":

```
    print instr("abccbabaccab",q$,3)
     6
```

The string "ab" occurs 3 times in the first string; at positions 1, 6, and 11. Since the third argument is 3, it will skip over the "ab" in position 1 and will find the next occurrence in position 6.

INT(X)

Finds the greatest integer not greater than the argument. If a is 4.2, then int(a) will be 4, since 4 is the greatest integer not greater than 4.2. If a is -10.83, int(a) will be -11, since -11 is the greatest integer not greater than -10.83.

LEN(X$)

Finds the length of the string argument expression. If a$ = "olive", then len(a$) is 5.

LOG(X)

Obtains the natural logarithm of the argument expression. The argument must be greater than 0. if e = exp(1), then log(e) is 1.

PEEK(X)

Gives the value stored in the location specified by the argument expression in the currently selected bank. A later chapter will discuss PEEK more.

PI

This function does not need an argument like all the other functions. It is simply the PI key next to the return key. Since the 6400 printer cannot print this symbol, we will just use the 2 letters PI to represent the graphic symbol PI that is on that key. We can set the variable pi to the value of pi (3.14159265357989...) by:

```
20 pi = PI: rem key next to return key
```

POS(X)

Gives the column number of the cursor position. The argument expression must be specified, but it is ignored. POS(X) will always give a value from 0 to 79, with 0 being the leftmost column and 79 being the rightmost.

RND(X)

Picks a random real number from a sequence of random numbers. The random number will always be between 0 and 1. If the argument expression is 0 or positive, RND will choose the next number in the sequence of random numbers. If the argument is negative, RND will use the argument expression value to pick a starting point in the sequence. The tutorial disk further explains this useful function and clarifies it with examples.

SGN(X)

This function finds the sign of the argument expression. It's value will be -1 if the argument was negative, 1 if the argument was positive, and 0 if tAe argument was 0. In math, this is called the signum function.

SIN(X)

Finds the sine of the argument expression. a = sin(pi/2) sets a to 1.

SQR(X)

Finds the square root of the argument expression, which must be non-negative. If a is 10, sqr(a) will be approximately 3.16227766.

TAN(X)

Finds the tangent of the argument expression. TAN(X) is the same as SIN(X)/COS(X). tan(pi/2) is infinitely large. If a is pi/4, then tan(a) is 1.

USR(X)

Is covered in a later chapter.

VAL(X$)

Finds the numeric quantity represented by the string argument expression. For example, if a$ = "-22300e-3", then val(a$) is -22.3. If the string contains non-numeric characters in it, VAL will convert the portion before these characters and will ignore the non-numerics and everything after them. For example, if a$ = "12% for 1 year", then val(a$) is 12. If the first character is non-numeric the result is 0; so val("#35") is 0. VAL is the inverse of the STR$ function.

3.3.2 String Functions

All string functions result in a string value.

CHR$(X)

Finds the single character whose ASCII code is the argument expression. The argument may be from 0 to 255. For example:

```
20 cr$ = chr$(13): es$ = chr$(27)
```

The string variable cr$ will contain a carriage return, and the string variable es$ will contain an ESCape character. The manual(s) that came with your B contain(s) tables of ASCII code to character conversions in appendices.

ERR$(X)

Finds the error message associated with the error whose number is the argument expression. This function will be covered in a later section about TRAP and RESUME.

LEFT$(X$,X)

Finds the left X characters of the string expression X$. The first argument expression must be a string, and the second must be a numeric expression. For example:

```
   who$ = left$("IDIOT",1)+" am"
```

assigns the string value "I am" to the string variable who$. If the numeric expression is 0, an empty string ("") is the result. If the numeric expression value is larger than the length of the string, the result is the entire string.

MID$(X$,X) or MID$(X$,X1,X2)

The first argument expression must be a string. The second, and the optional third argument must be a numeric expression. This finds a portion of the string starting with the character whose position is given by the second argument. If the third argument is used, the resulting portion will be that many characters long. For example:

```
   name$ = "variables": first = 5: size = 4: w$ =
mid$(name$,first,size)
```

will set w$ to the string value "able", the 4 characters starting from the 5th character of "variables". If the third argument is not used, the resulting portion will be the rest of the string from the starting position. For example:

```
   mr$ = "ur85-12302": num$ = mid$(mr$,6)
```

will set num$ to "12302", the last characters of "ur85-12302" starting from the 6th character.

The second argument must be from 1 to 255, while the third argument must be from 0 to 255. If the third argument is 0, or if the second argument is greater than the length of the string, the result is an empty string (""). As with LEFT$, if the third argument would cause the MID$ portion to extend beyond the end of the string, the result is the rest of the string (as if the third argument hadn't been given). Thus, mid$(a$,3) is the same as mid$(a$,3,255).

RIGHT$(X$,X)

This is similar to LEFT$ except that it finds the right-hand portion of the string argument that is as long as the value of the numeric argument. If the numeric argument is 0, the empty string ("") is the result. If the numeric argument is greater than or equal to the length of the string argument, the result is the entire string argument [right$("abc",6) is "abc"].

STR$(X)

Finds the string that represents the printed form of the value of the numeric argument. The leading space or minus sign is included, but not the trailing cursor right. Thus str$(-cos(0)) is the string value "-1" and str$(3/5) is the string value " .6" (including the space).

3.4 More Decision Making (ON/GOTO) [Program Flow 5]

Earlier we learned that the IF/GOTO can be used to direct the flow of the program execution into one of two directions. Also, IF/THEN/ELSE could also split the flow into one of two relatively small program sections (the statements of the THEN and ELSE parts). Often your program will have to make decisions that might take it into one of many different directions. The ON/GOTO statement accomplishes this fairly simply.

The syntax of the ON/GOTO statement (that is, the way it should "look") is as follows:

```
   ON expression GOTO line#1, line#2, line#3, ...
```

The "expression" represents any numeric expression, and the line#Ns represent a list of 1 or more line numbers. When BASIC encounters this statement it will go to a line based on the value of the expression. If the expression has the value 1, it will go to line#1; if it has the value 2, it will go to line#2; and so on.

The odd situations here are: (1) if the expression has a negative value, the result will be a BASIC error message and the program will stop; (2) if the expression is either 0, or greater than the number of line#s provided, BASIC will simply ignore the whole statement and will continue on to the next statement - possibly even immediately following the ON/GOTO on the same line.

Here is a graphical example:

```
   100 on len(a$) goto 200, 300, 400: go to 50
   200 rem a$ had 1 character
     ...
   300 rem a$ had 2 characters
     ...
   400 rem a$ had 3 characters
     ...
```

If a$ has 0 characters or more than 3 characters, BASIC will execute the "go to 50" statement. In this case it is impossible to get an error message, since the length of a string variable can never be less than 0.

(Fortran fans will find that the ON/GOTO is a "computed goto", and the tutorial disk shows how to use the SGN (signum) function and ON/GOTO to simulate the "arithmetic goto".)

Let's pause here to implement a small program based on some of the things we have learned so far. We will write a small section of a program that shoots one round of craps. The rules state that a 7 or 11 on the first roll is a win; a 2, 3 or 12 on the first roll is a lose. Rolling any other number (4, 5, 6, 8, 9, or 10) causes this number to be called your "point". You then roll until you get your "point" again, in which case you win; or until you roll a 7, in which case you lose.

```
 100 d = rnd(-val(ti$)): rem randomize the random number
generator
 1000 rem "roll" dice:
 1010 d1 = int(rnd(1)*6+1): d2 = int(rnd(1)*6+1): d = d1+d2
 1020 rem d will be 2 to 12.  for on/goto we will subtract 1
to get 1 to 11
 1030 rem rolled:  2     3     4     5     6     7     8     9
10   11   12
 1040               on               d-1                 goto
1100,1100,1200,1200,1200,1150,1200,1200,1200,1150,1100
 1050 stop: rem if it ever gets here, we really messed up
somewhere
 1100 print d,"You lost on the first roll!"
 1110 go to 1300
 1150 print d,"You won on the first roll!"
 1160 go to 1300
 1200 print "Your point is" d
 1210 p = d: rem save point for later.
 1220 d1 = int(rnd(1)*6+1): d2 = int(rnd(1)*6+1): d = d1+d2
 1230 print d,: if (d<>p and d<>7) then print"Not yet.": go
to 1220
 1240 if (d=p) then print"You won!": else print"You lost!"
 1300 end
```

The longer description of the RND function in section 3.3 on the tutorial disk makes it clear what is being done in lines 100, 1010, and 1220.

Line 1230 is an example of using a comma at the end of a print statement. The appropriate message ("Not yet.", "You won!", or "You lost!") will appear in the second print field.

This is a good start to a real game. It isn't a real finished product, but it works. We might give the player a chance to try again:

```
 1300 print"Try again (y/n)?"
 1310 get c$: if (c$="") goto 1310
 1320 if (c$="y") goto 1000
 1330 if (c$<>"n") goto 1310
```

We could add betting, then graphics, and so on. BASIC is great for allowing you to develop an idea and then expand on it as you wish.

3.5 Saving and Loading Programs (DSAVE/DLOAD) [Disk Commands 1]

Now that we've put in our first significant program, we will probably want to save it on a disk. First we must pick a name for the program. How about just "craps"? Usually we use only lower case letters in file names so that we can read the names even when the computer is in graphics mode. Remember that upper case letters get changed to graphics characters in that mode. To save our program with the name we chose we would type:

```
    dsave"craps
```

The dsave command requires a string expression which will be the name of the program on the disk. Here we used the string constant "craps" but blatantly left off the trailing quote, which is perfectly OK in Commodore BASIC.

Since we did not tell it otherwise, BASIC will assume that we wanted to save the program on the disk in drive 0. The dsave command allows us to specify the drive that we want to save the program on, assuming drive 0 if we don't tell it otherwise. To save the program to a disk in drive 1, use:

```
    dsave"craps",d1
```

The d1 stands for drive 1. BASIC doesn't care in which order you give it disk command information, so we could have used:

```
    dsave d1,"craps
```

The space after dsave was not mandatory.

Whenever you change a program, you are only changing it in the computer's memory - until you use the dsave command to save it back to the disk. If the program was already there and you try to dsave it again, you will get an error (the disk drive error light will come on). The disk unit will not assume that you want to update an existing program, it will assume that you forgot that you already had a program on the disk with that name, or that you mistyped the name. In order to tell the disk unit to REPLACE the existing file with the changed program, you must place an at-sign (@) in front of the program name:

```
    dsave "@craps
```

The @ is a special character in this case. There are other special characters that will be explained later, but for now we will list them. These special characters must not be used for the first character of a file name:

```
    @ # $
```

These special characters should not be used anywhere in a file name:

```
    : = ? * ,
```

Any other character, even non-printing characters and control characters, may be used in a file's name. However, it is best to stick to just the printable characters and avoid the upper case letters or the graphics that they get replaced with - just so you can see what the file name means.

To load in a program from a disk, use the dload command. It uses the same parameters as the dsave command, including d# and u#:

```
    dload "craps
    dload d1,"craps
```

The dload command will wipe out any program already in memory (as if you used NEW first) before loading the program from disk. It will also clear all the variables. Once the program is DLOADed, you can RUN it, change it, or whatever you want to do.

The tutorial disk explains how to dsave to / dload from a second disk unit, if you are fortunate enough to own one.

3.6 Checking Statuses - DS, DS$, ST [Disk Commands 2]

There are several BASIC variables that have pre-defined meanings. That is, they cannot be used as general purpose variables because BASIC always associates certain values with them. Three of these variables can be used to check the status of an operation that was just performed with a disk. DS and DS$ report errors specifically on the disk unit that was used in the last disk command. ST reports error conditions on any Input/Output device, such as the printer, the RS-232 port or the disk. None of these variables may have their values changed via an assignment statement or anything else. Thus all three of the following statements will cause an error:

```
   100 ds = 0: input ds$: read st: rem read instruction
covered later
```

After a disk command is issued, you can check for errors by checking the DS$ variable:

```
    ?ds$
```

Remember that ? is an abbreviation for PRINT. This will interrogate the disk unit just used for the previous disk operation to see what its status is. If nothing went wrong, the result will be one of:

```
   00,ok,00,00
or 00,ok,00,00,0
or 00,ok,00,00,1
```

The first line is given with DOS versions prior to version 2.7. Either of the latter 2 is possible depending on which drive (0 or 1) was referenced in the command. DOS 2.7 reports which drive had an error (or in this case a non-error) as the last number.

If there was an error, some other message will be displayed such as:

```
   23,read error,14,3,0
```

Again the final ,0 won't be present in DOS versions prior to version 2.7. The number 23 is the error number. This number may be used to look up the error in the disk manual for further explanation. Error numbers less than 20 are not really errors - they are just information that the disk unit wants to report to you. The "ok" message above is an example of such an error. Another is:

```
   01,files scratched,02,00,1
```

The 01 indicates this is not an error. This message is just used to indicate that some files were successfully removed from the disk.

The second item in these messages is the actual error message itself. "ok" meant everything was "oll korect" as Andrew Jackson used to say when he coined the term OK. The "read error" message means the disk unit could not read some information from the disk. Again, the error number will help you to find a more substantial explanation for the error in the disk manual.

The next 2 numeric items are usually the track & sector, respectively, of the disk where the error occurred. For many errors the track and sector number are 00,00 simply because the error was not caused by some specific track and sector of the disk. For example, if no disk was present when you tried to dsave a program on drive 1, you might get this error:

```
74,drive not ready,00,00,1
```

Since the drive was not even ready (no disk even being present), no specific track and sector was at fault, so it just reports 00,00.

In one peculiar case, that of "error" 01, (files scratched), the number that is usually the track is actually a count of how many files were scratched. This is the information that the disk unit simply wanted to tell you, which was not an error at all.

For drives with DOS 2.7 or later, the final 1-digit number is either 0 or 1, indicating which drive was at fault. Again, some errors are not caused by a specific drive, so this number will simply be the last drive that was referenced.

The DS variable, being able to only store a single numeric quantity, always reports just the error number part of the error message from the disk - that is, the first number in the DS$ string. For example, if everything is OK, DS will be 0. If the "drive not ready" error occurs, DS will be 74. Sometimes it is easier for your programs to check DS before deciding whether to continue or report an error:

```
1070 if ds>19 then print ds$:stop
```

Remember that ds values less than 20 are not really errors.

The ST variable is not just linked to disk operations. It is actually modified whenever the IEEE bus is used. This variable is described further in the tutorial disk.

The next issue will continue with Chapter 4.

Any questions arising from this tutorial should be sent directly to the author, whose address is given below. Also, you may obtain disks containing the entire tutorial directly from the author. It comes in either a dot-matrix (4023, 4022, 2022, etc.) version, or a letter quality version (6400, etc.). Each version (1 disk) costs $11, or you can obtain both versions for $18 (both disks). Handling is included. Write to:

Warren D. Swan
1 N 114 Woods Avenue
Wheaton, IL 60188

◆◆◆◆◆◆◆◆◆

USEFUL TIPS + A THANK YOU

by: Mary K. Hoyt

Dear Norman,

This started out as a simple letter, but then I decided to go ahead and put it on a disk, just in case you wanted to reproduce part of it, without retyping. Please forgive the plain old PET-ASCII format, but I just don't have access to the B128 or SuperScript right now, and if I waited until I did, and had time to write again, this letter would be delayed another 6 months, I'm sure. (To give you an idea of how long it takes me to do things, the other excuse is that I have been having so much fun with the utility programs I bought from CBUG, I haven't yet learned SuperScript. I am not a writer).

I found a bug in the FINANCIAL UTIL OCT87 CBUG #5 disk I bought from you a while back. It is in program ANNU1, a nifty little program that computes various annuity values. Being an actuary, I like to play with such programs, and have written several myself, for the home and the office. When I got some strange results for one value, I listed line 410 of the program. There are several valid formulas that could have been used here, but I will propose the following correct one:

```
410 IF FV = 9999 THEN FV = PMT * (((1+I)@(N+1)-1)/I - 1)
  <<The symbol for UP ARROW has been substituted as @
```

The second thing I thought I might share with some other members is my Dad's experience with OUT OF STACK errors in his BASIC program. We read as much as we could in the manual, and in a couple old magazine articles I knew of, but they didn't really help him, because he said he had very few nested loops and GOSUBs. I "yelled for help", and found that few other BASIC programmers had ever had that error message. But a couple of "helpers", including Warren Swan, told me that it had to be a problem with FOR/NEXT loops or GOSUBs, and suggested I use Liz's Keytrix program to find all his subroutines and loops, etc. It was such a large program, I couldn't bear the thought of tackling it without Keytrix, but using Keytrix, I located all the appropriate subroutines and branches, and, by golly, found one subroutine that contained a branch back to the main program, where it could possibly encounter the GOSUB again (depending on the data in that particular run),and branch back without RETURNing again, etc. We fixed it to branch to the RETURN statement (after setting a couple flags) instead of to the main program, and we haven't seen that nasty error message since. Thank you, all of you generous members who have listed themselves in the Yell for Help directory!

Mary K. Hoyt, 154 Robmore
Houston, TX 77076

◆◆◆◆◆◆◆◆◆

THE NEW JERSEY GOLD MINE

Following is a series of articles from one of our most prolific members -- author, programmer, scientist, and engineer extraordinaire. Single handedly Tony has created miracle upon magic upon sheer genius. We all owe him a great thank you.

IEEE UNIT TO UNIT BACKUPS

by: Anthony Goceliak

These two programs will allow owners of b-128 / b-256 computers to make backups of disks from one disk drive unit to another. What that means is for instance backing up from one sfd-1001 to another sfd-1001, or from a 2031 to an 8050. They are being released in this issue of THE CBUG ESCAPE as CBUG #72 -- see the library section.

Owners of the sfd-1001 single drive units have heretofore been unable to reliably backup disks, especially if one of the drives is even slightly mis-aligned and has difficulty in reading disks written by another unit. The only method of backing up disks previously available has been the copy-all type approach, which is both agonizingly slow and does not handle the rnd type of data storage used by Superbase for instance.

These two programs however, handle the transfer of data in the same manner as the dual drive backup d0 to d1 command, transferring each sector of each track exactly as it is encountered, thereby preserving any type of data storage possible with the drives. Even a 2031 disk which had a program which stored and retrieved data by direct access u1:##,0,(track#),(sector#) instead of using a named file will be preserved intact on the 8050 destination disk, and should function identically. The only exception is for the very rare case of--ampersand files, or block execute commands, which run not in your computer, but in the drive itself. Such a file must be written for the specific type of drive which it is to be run on, and although an ampersand file written for the 2031 will be faithfully transcribed to an 8050 disk, it will doubtlessly fail to function.

The programs WILL NOT backup copy-protected disks, which are disks with either intentional or un-intentional dos errors on them. Unclosed files, those marked by dos with an asterisk, can be backed up by these programs, although they will not be repaired.

The requirements of the programs are as follows:
1. A b-series computer. Either 128K or 256K are acceptable. Neither plug-in cartridges nor memory expansion is needed, but if already installed these do not have to be removed.

2. Two IEEE drive units. 2031, 4040, 8050, 8250, sfd-1001. (two of the same type except 2031 to 2031, or two different units)

Selection of which program to use is simple. If you do NOT intend to backup from a 2031, select the first program by typing shift/run. If you intend to backup from a 2031 to an 8050, 8250, or sfd-1001 select the third program by typing dload"2031-8050 backup" (and return), followed by run (and return). The second file is a suite of machine language programs automatically loaded and disbursed to the source and destination drive units.

If your disk units are all left as they were addressed at the factory, you will have to switch IEEE plugs around as the units are software configured to unique device numbers. Non 2031 drives may be switched off while the remaining unit has it's device number changed. For everyday use of multiple drive units, I reccommend performing the simple operation of permanently re-addressing one of the units as described in the Commodore Disk System User Reference Guide Appendix C., which will allow you to bypass this section of the program.

Follow the instructions and answer the questions presented to you on the screen. When backing up data disks in a repetitive manner, to a disk which has already been formatted and not bulk erased or subjected to a dos error, it is sometimes not necessary to re-header a disk. This can save several minutes, and when the program asks, you may elect to eliminate this step or not, as you choose. In general, if the disk which will receive the data was NOT formatted originally by the drive which is to receive the data, it is best to re-header.

The non-2031 program, due to the relatively larger memory available inside the drives executes three carefully written, general purpose programs from within the drives, one for each of the interface processors, and one for the source floppy disk controller, each executing in concert with the master machine language supervisory program executed by the b- computer. The programs can identify and skip over all-zero data blocks, thereby significantly speeding up backup of a disk which is not 100% full. Representative times for backing up disks will vary depending on how full a given source disk is, whether you skip headering, and how many sectors require a second, third, or up to a fifth re-try before being read or written correctly. This disk was backed up between an 8050 and an sfd-1001 in 1 minute 55 seconds excluding the time for headering. A worst case of an entirely full sfd-1001 to sfd-1001 should require less than 18 minutes unless re-tries are needed.

As the data is transferred from source to destination, it is stored briefly on the screen of the b-computer, allowing you to monitor the progress of the operation if you desire. No action is required from you, and the monitor screen may be turned off or left unattended while the program does its work.

Once the headering or backup operations have begun, the rest is automatic, and the program will proceed either to completion, or indicate an error. All reads and writes are re-attempted a minimum of five times before the program will give up. If an error is indicated there are six possibilities.

1. The Source disk (the one which was to be backed up) was copy protected. These programs will NOT backup disks with intentional dos errors.

2. The Source disk has an un-intentional error. This is generally caused by flipping open the drive door during a write operation (save, dsave, header, collect, close, dclose, open, u2: [or any command which writes to the disk]). Recovery of disks with these types of errors is very much a case of art instead of science, and is not covered by the scope of these programs.

3. The disk is 'out of alignment' RELATIVE TO THE DRIVE. Unless you skipped headering this should only affect the source disk/drive combination. Either the disk or the drive or both may be wrong. Try running the program again, but change the disk to your other unit and backup again, but pay attention and do not mistakenly designate the drive with the disk to be backed up as the destination!

4. You have attempted to transfer more tracks than the source or destination disk or drive is capable of. The programs will indicate automatically the correct maximum number of tracks to transfer based on the disk drive hardware connected, but a request to transfer more than 77 tracks from an 8050 source disk will apparently fail even when two sfd-1001's are used.

5. You have attempted to bypass the headering of a disk which has either never been headered, which was bulk erased or which was otherwise damaged.

6. You are attempting to use a computer which is not a b-series computer. A pet 2001 or C-64, even when equipped to handle an IEEE connection will not work.

When running this program to backup from an 8250 or sfd-1001 disk using an 8050 drive, be aware that the 8050 cannot read or write the top side of the disk, so even though the directory may indicate that a file has been backed up, if the sfd or 8250 disk originally showed less than 2100 blocks free, it may not have been transferred. When backing up between the same type drives this will not occur.

The 2031 to 8050 program, after finishing the backup function, will have a bit more work to do, transferring the directory and bam blocks to the appropriate tracks. This is also entirely automatic.

One last thing, you may not backup from an 8050, 8250, or sfd-1001 to a 2031 or 4040, because the small drives will not be able to accommodate a directory, bam or for that matter the first-used tracks of the big drives.

IF YOUR BELT'S TOO LOOSE, YOUR PANTS FALL DOWN

by: Anthony Goceliak

Anybody who has read even part of one of my articles knows that the above title refers not to sartorial spendor and potential embarrassment, but instead to certain portions of your 'hardware', namely your disk drive.

More than one CBUG member has complained of loss of data, write errors, 'intermittant' errors or inability to format on one drive and while many other failure modes are possible besides this one, here is a relatively rare opportunity to take a deserved potshot at Commodore for a failure to adequately test for belt slippage from their own diagnostic programs.

Yes indeed, the diagnostics as they stand do test for belt slippage, and the test itself is even a valid method of observing for belt defects, so what else is left? The PLACE the test is performed!

Commodore, probably in order to leave the poor 'formatted test disk' in some semblance of order upon completion of testing chose track #78 as the track to use for speed and belt tests. The normal 8050 has tracks from 1 to 77 and so no normal data is overwritten. This would allow a 'CBM Authorized Service Center' [B-128? Surely you mean 'c'-128!] to test your drive with the very disk which was giving you trouble. Highly commendable, but it leaves a hole big enough to drop a disk in. I know, since I just spent several (!) minutes recovering from the big C's decision to be kind.

Taking a few steps back, having finished writing and debugging a program I was routinely dsaving it on drive #0 when I noticed considerable blinking of the error led. After what seemed to be an eternity, the drive stopped with a green led. I promptly saved the program again on another disk from drive #1, and that operation went smoothly. Verifying time, and a repetition of the above. OK, time for the good old diagnostics. Everything looked fine!

Slightly suspicious, but thinking that I had probably run across a defective disk (never have before, but why not?), nothing was done. The next day, I again wanted to add a program to my disks, but now a symphony of bumps and the drive is apparently embarrased, since the error led is red.

Right here let me tell all of you something. When the error led is red, ALWAYS ask for ds$! When you ask for it, (by typing from basic mode ?ds$ or from SS II/SB I the disk status information), WRITE IT DOWN. If you ever hope to be able to do something about it, you will need to know each and every digit.

Ds$ was 25,write error,04,00,0 which means really that the automatic read after a write (that's why writing takes so much longer to do than reading) failed to compare with the buffer of information which was destined to go to track #4, sector #0 on drive #0.

Not to panic, I have utilities to handle this. Out comes my advanced disk doctor, and blotto, ten direct appeals to the fdc job queue result in a byte decoding error message, not only on that particular block, but also on track 3, 2, and 1, and I haven't even tried to write to them lately!

AHA! The trouble's not on the disk, except for the single sector #0 from track #4, but in the drive. Moving the disk to drive #1 and disk doc's getting much better results except for the single bad sector. Obviously the diagnostics come out again, and they pronounce everything ok once again. I now had the drive open, having looked at the head to see if it was dirty (it wasn't), and spotted the flaw in the diagnostic program right away. Track #78, where the belt test is performed is very near the center of the disk, while tracks 1 to 4 which are giving me trouble are as far from the center as possible. Think way back to your high school physics class, or if you slept through that, consider whether it is easier to make Frank Sinatra sound like a basso profundo by pressing your finger down near the edge of a phonograph record or by pressing near the hub.

The answer, of course is near the edge, and that is where the head was when it couldn't write or read properly. There is a certain amount of drag which the belt drive of your floppy disk drive must overcome to spin the disk. On a given disk, some of it is constant (the disk to the liner, the friction in the drive spindle, etc) and some is variable (the head and pressure pad grabbing the disk) depending on the track which is currently accessed. The drag was exceeding the amount tolerable to the belt mechanism only near the outside edge of the disk.

Temporarily reworking the diagnostic program in b-128 memory to use track #1 as the speed and belt track and re-running with a freshly formatted but expendable(remember the speed/belt track will be destroyed, not merely overwritten!) disk yielded a much sorrier tale for the belt.

Removing the drive unit, cleaning and reversing the belt and re-running the diagnostics got me up and running, and if you have the same type of difficulty, may I suggest the same remedy for you. A bit of time, some distilled spirits, a small cloth and you may no longer need a trip to the service center with your spinning but no longer dizzy disk drive.

THE SMART PERIPHERAL
or
What Computer? I'm running my 8050

by: Anthony Goceliak

Every now and then a bizarre thought can bear fruit. For instance, in a recent discussion with a friend, who maintains anyone working with an orphan computer must be similarly situated regards a brain, reliability was the topic. When (note I said when, not if) his 'current technology' (read as EXPENSIVE!) beast goes to the shop, they will rent him a replacement. The last time he rented it for over six weeks (I wonder how much extra speed the service station uses to repair a computer when it knows that as soon as the repair is done, they lose rental revenue?).

I of course pointed out the advantages of having separate, intelligent peripherals, and he countered with a comment to the effect of 'so what? without the computer, they can't do anything!', but guess what, he was WRONG!

The good ol' 8050, sfd, and 8250 each have a test performed at power-up which allows a properly constructed disk utility to be in essence 'shift/run' without intervention by a computer at all.

A specially constructed IEEE plug, when placed on the drive's connector before powering up, sets the stage. As soon as the plug is removed, or in my case, a switch wired into the plug is thrown, the drive will spin d0 and attempt to load and run the first file on that disk. It is a simple matter to write such a utility to backup disks (on the dual drive units), to enable another cbug member to print out your files in a dire emergency without having to expose the original disks to the vague and sometimes destructive whims of the postal service. Now that is reliability in distributed intelligence!

REVERSE VIDEO SCREEN IN SSII
by: Anthony Goceliak

Did you know that you can enter the Superscript II program while your screen is set up as reverse video, and the entire program will respect your choice?

Just type the following BEFORE SHIFT/RUNning SS II:

```
print chr$(27)+"r" [and return]
```

No furthur actions are necessary, the program will load and run as before, but your video display screen will show everything reversed from normal operation. Some people (traditionalists?) apparently read black text on a light background better than light text on black background, and this gives you the opportunity to decide for yourself.

The modification causes no ill effects when used with any combination of no bump loader and / or pre-superscript (Ms. Deal's), and may be incorporated in either of the above basic programs to make reversed-screen format automatic when bringing up SS II.

One word of caution, do not leave your display showing the same screen of text for extended periods or you run the risk of fatiguing the crt display phosphors. In other words, you can 'etch' a ghost image of that reversed screen permanently on the crt. In practice, however, if your screen has shown no ill effects from the top line of reversed video which is normally displayed by SS II, then you have little to worry about. In order to cut down on crt fatigue, at least on the b-128, try shutting off only your monitor if you must leave the system unattended for hours. My personal preference however, is to shut down the entire system (after saving text!) and reload Superscript when it is needed.

FUNCTION KEY DEFAULTS

by: Anthony Goceliak

FUNCTION KEY DEFAULTS

Since I wrote a program which changed the default screen initialization, I just explained how to restore the screen to its power-up state. But how about the function keys? Well, I don't use the standard function keys, having written a new set of definitions for them, so this piece of b-magic is not used by me routinely, but for those of you who have played with only the first ten keys!!! this will reset them to the power-up definitions. If you have defined any of the keys 11 through 20, the definition of the original ten will be fine, but 11 through 20 will have mutated into something relatively unpredictable, and not likely readable. To initialize to the exact power up state of function keys should you have messed with keys 11-20, a significant number of pokes are required, so it is easier to code a loop in basic which says:

```
1000 for x=11 to 20:key(x),"":next x
```

This lets the operating system do all the poking for you, and sets up for the real trick which is here:

```
1010 bank 15:sys 57487:end
```

This is a kernal call, so both the b/128 (which I have and have tested this on), and the b/256 which I don't have, but which possesses the identical kernal, should execute properly. As always, there is no guarantee for the 256 since the code has not been tested there, but a simple test described near the end of this article will enable you to determine if we have a 'go' on the 256 or not.

The line numbers of 1000 and 1010 are for illustration only and may of course be altered to suit your own programs' needs, and the two lines may even be combined into one.

Following execution of line 1010, the function keys from 1 through 10 will be found to have their original definitions. The reason for keys 11 through 20 mutating is that the keys had no pre-determined definitions to be copied from the b/ROM but when powering up, the static RAM test convieniently left the locations describing the lengths of the key definitions all equal to zero. We do not want to invoke this RAM test, or the entire b will have to be reset, having lost track of its entire operating system! As a consequence, we should first re-define these 'extra' keys as nothing at all, which will prevent the display of garbage when keys 11 and up are accidentally pressed. The code will proceed onward and set your screen to the default mode also, (no windows, text mode, normal screen, scrolling enabled, etc.) so although the code could be placed in the middle of a basic program, I do not reccommend it. This is a 'return to default settings' procedure. For those wishing to temporarily store non-standard key definitions, allowing re-definition for the duration of a program and subsequent restoration at it's end, read my other function key article for a 'lazy man' approach using no disk writing and no m/l or any non-standard memory.

The test to determine the suitability of the code is quite simple, and may be typed in and run as follows;

```
10 key:rem this will list out the current definitions of the f-keys

20 key 1,"fazz-bazz"
30 key 2,"goorp"
40 key 3,"irnbltz"

50 rem just go on and 'invent' new definitions for keys 4 through 10 (or 20)

120 input "Press return to proceed";x$

130 rem just a pause while you study the screen

140 key:rem this will now show how the keys have been re-defined

150 rem by the way, unless you did something useful, don't try to
160 rem execute these nonsense keys!

170 input "Press return to proceed";x$

180 rem another pause because the next instructions will clear the screen

190 for x=11 to 20:key(x),"":next x

200 rem line 190 not really needed here but included for completeness

210 bank15:sys 57487:end
```

Now type key (and return) to show that the list of function keys has been restored.

By the way, why am I telling you about this? Well the usual way to restore the keys runs to several lines of basic and quite a bit of typing, actually typing each definition for each key. Including this sys call at the end of a program saves considerable effort, and should help promote the free-er use of those convienient programmable functuion keys to eliminate repetitive typing or user input.

P. S. Remember this is a return to power-up default keys, so if you are a regular user of say Ms. Deal's Keytrix or my own f-key set up, the non-standard definitions will have to be re-loaded.

THE 'BEST' ROUTE TO DEFAULT SETTINGS

by: Anthony Goceliak

Speaking of default settings on the b, there is a relatively quick, absolutely foolproof way to ensure that a program cannot leave your computer in a state which could cause anything unexpected. It is called the power switch, and is located (from the operator's viewpoint) at the rear-left of the computer. Operation of this switch first off and then after perhaps 10 seconds back on will completely reset your machine as though the former program had never existed. Screen set-up, function key definition, program erasure, pointer initializations, even memory test after a fashion are all accomplished entirely automatically!

Seriously,. if a basic program includes the possibility of a 'usr' or 'sys', or even a poke, the computer may or may not be left in its original state when the program ends. Only a full cold-start reset will guarantee this, and even the cold-start can be fooled into skipping parts of the initialization by finding certain values deposited in memory. However, nothing fools the power switch method as long as the computer has remained off long enough to ensure that all RAM has 'forgotten' what was there before.

SAVING NON-STANDARD FUNCTION KEYS FOR RESTORATION

THE NO-DISK OPTION

by: Anthony Goceliak

Programmable function keys are great, but most people are afraid to use them. It is possible to save selected areas of the b's memory to disk, preserving the key lengths and definitions, allowing free modification of the keys and subsequent retrieval of their original definitions. Many times however, it is inconvienient or inappropriate to require disk writing in order to save the original set-up. You would be required to leave off any write protect tabs, and if the directory or disk was full, the file won't fit anyway, subsequent runs of the program after the first must support scratching the old file, and finally you must make sure that the original disk is back in the drive when it is time to end the program.

Alternately, it is possible to transfer the function key lengths and definitions to some area of the b's memory under control of a short m/l routine, [and this route will proceed with lightning speed] but this article will demonstrate how to accomplish this from basic, without worries over exactly where to store anything.

The b has a most ample supply of memory, and this basic method allows literally any function key set-up, standard or not, to be temporarily stored as basic variables until the end of a program, when all f-keys are automatically replaced. This allows free re-definition and use of all function keys, giving you the opportunity to cut down on your typing, or allowing very sophisticated programming.

Basically, what you will do is merge your program with mine and observe three simple rules:

1. Avoid line numbers 0, 1, 2, 3, 63950 and up.

2. Avoid use of the String array variable fk$( ). The program uses fk$(0) through fk$(20), and although you could alter the dim statement and use fk$(21) or higher, why bother? Doesn't the b support enough other variables?

3. Avoid inclusion of spaces within the original function key definitions. None of the default function keys include spaces, but for instance my old set-up included key 17 as "backup d0 to d1"+chr$(13)+"y". The spaces will cause the key definition to be truncated. Just squash everything together, as in "backupd0tod1"+chr$(13)+"y" and the program will handle the definition properly.

My code uses variable x as a counter as well, but restores it to zero before leaving line #3, so x is left as if you had typed 'run'. Later on when your program would ordinarily end, replace the 'end' statement with 'goto 63950' instead, and the function keys will be restored to whatever they were before the program ran.

If your program halted on an error, so that the normal exit through my code at line 63950- was not reached, the original function keys may sometimes be restored (depending on what the error was), by typing 'goto 63950', and of course, return.

To use this set-up, first dload your program. If you are intending to write a basic program from scratch, obviously skip this step. Type the following two commands to make sure that there will be no duplicate line numbers.

```
list -3
list 63950-
```

If you see any lines of program listed, you must renumber the program first, using whatever utility or method you desire. This is crucial.

Assuming you can type the two list commands and see no program lines, there is now only one thing left to check, and that is the use of the string array variable fk$(..). Note that variables fk, fk%, and fk$ are ok, only references to fk$(0) and so on through fk$(20) are forbidden. Duplicate use of this string array, or efforts to re-dimension it by your basic program will result in an error message, or incorrect restoration of function keys. Assuming that there are no references to fk$(..) in the program, type 'new'.

Now dload my program 'f-key restorer'. Type 'list', then 'new', and finally dload your own program again. If you are writing your own program from scratch, do not type 'new'.

Last, cursor up to my line #1, and press 'return'. Continue pressing return over each line of my program. When you finish, type list, and you should see that both programs are properly merged in line number order. Save the new program to disk or tape, and that is that. I highly reccommend changing the name of the program instead of 'dsaving @'.

GET THE RAID! I FOUND A BUG!

by: Anthony Goceliak

<<NOTE the symbol for up arrow has been substituted with "Q" as our typesetting printer does not have the up arrow.>>

Recently while working with my b-128 I had occasion to write a program which uses a 'good guess' approach to figuring out something. Essentially this method is for those of us who know how to test to see if a guess is too high , too low or right on, but who are either unable or too lazy to go and find the right formula to determine the answer properly.

'Good guess' is a royal pain to do by hand, but it is the kind of thing that a computer just eats for breakfast. The exact problem is not really germane to this discussion, only the reason why the 'ggm' did not work.

Try this out. First power down your b, disconnect (if you wish) all the extras which you may have hooked up to the b except of course your monitor. [it doesn't really matter, the b will bomb anyway, but this step is important for one of us to have taken in order to prove that it wasn't some "unauthorized something or other" hooked into the rs-232 port that caused the difficulty.]

Now, whether you have disconnected anything or not, power up again after perhaps 30 seconds. The computer is now in it's pristine power-up state, so we can't blame some previous program for what will happen.

Next, after seeing the power-up logo, type in the following basic program:

```
10 for x=800 to 900
20 a=.9Qx
```

Just in case the up arrow is mis-translated by the type-setter, line 20 reads ::==> a equals nine-tenths raised to the power of x
(Raised to the power of key is shifted six ON THE MAIN KEYBOARD!)

```
30 print x,a
40 next x
```

After entering those four lines type run (and return!), and watch. Lines of arcane numbers come pouring out until we get to 835 and then what? If you are up to it, try the stop key, but it won't work if you let the program reach the 'x=836' attempt. Reset regains control, but not without penalty, try some of your "new" function key definitions. Power down/up and re-type the program using values of x beginning with 836 through 842. Hmm, I remember wishing that I knew the 'right way' to enter the monitor, but I'll guarantee that this ain't it! Should you try with values of x=843 or higher, (or line 10 re-written as for x=900 to 800 step-1), the variable a gets set to zero, just like the PRG says.

Essentially, the b badly mishandles numbers between approximately 6.1e-39 and 2.9e-39 (between there and zero, the numbers are also technically mishandled, by being set to zero, but at least they don't crash the machine!)

For the curious, the foulup occurs not because of the values .9 or 836 (try a=.91Qx or a=.899Qx), it is the handling of the result in the range of 6.1e-39 and 2.9e-39 that gets us in big trouble, precisely why I do not yet know, but I have a tentative hypothesis. However, a bug bad enough to make the machine break out of basic and fall into the monitor or alternatively completely wipe the memory of the entire machine warrents immediate advertisement. These numbers are pretty small indeed, but even a repetitive addition and subtraction of relatively normal sized numbers can yield results in this range due to internal rounding errors. If anyone has a pure basic program that either locks up the b or crashes it due to arithmetic computation I would be interested in hearing about it.

If you doubt my word about the evaluation routine versus specific values, try this one from direct mode! Power down / up, and then type :
?.11Q40 (and return)

This example is from direct mode, not program mode, does not use a for-next loop, avoids any mention of any variables, and only evaluates (and should print) a value of roughly 4.5e-39! For the REAL die-hards, printing is not at fault, try this which should do something after evaluating the expression:

if .11Q40 then dload"b-128 bug demo":else directoryd0 (and return!)

After you power down and up again, doubting Thomas's may test the syntax by requesting ?.11Q39 [which works as intended].

The bug only strikes in the exponentiation routines, as may be demonstrated by the two simple lines following:

```
x=.11Q39:y=.11*x
?x,y
```

Y is now assigned the "fatal" value of (.11Q40), but the b will now calculate, display, and properly handle the previously forbidden value! Once more, if x=.11Q20 and y=x*x guess what? Can you predict what will occur when y=xQ2? (Hint: don't wait for your b to answer this one).

P.S. What my 'ggm' program was supposed to do was work the error downward until it became "zero", and then tell me the "good guess" (with an error of zero a pretty good guess indeed!), relying on the information in the Protecto Programmer's Reference Guide Page 14 concerning numbers. Gee, maybe it did get to zero, but basic never found out about it!

TRIVIAL EXAMPLE OF A POWERFUL TECHNIQUE

by: Anthony Goceliak

One of the CBUG'gers who is trying to learn m/l [alright assembly language] programming is doing so 'to escape the confines of basic'. In most respects he is correct, there are indeed many things which cannot be done either fast enough or easily enough to make them worthwhile from basic. But that is not to say that they cannot be done. The 650X chip which is the heart of our b has one particularly great opcode which is the indirect jump, especially when taken through RAM. Although the basic interpreter does not use this code as often as I might like, the kernal does, and this can often allow a 'way out' of a trap which a rigidly written basic system tries to impose on us.

To simplify that first paragraph, many times when basic says 'you can't do that!', it is armed with a pea-shooter instead of a bazooka.

A case in point is the automatic close all whenever a basic line is edited. In the general case this is an excellent precaution, but it does create a seemingly insurmountable road block to many things. The example given here is a sequential file to basic program converter, a task which is admittedly best left to m/l routines, but will serve to illustrate our dilemma.

A seq file listing with no basic tokens may be created in the following manner, and is also quite often the result of a 'download' of a generic basic program from some data service not particularly supporting commodore machines.

Presuming there is a basic program in your b's memory and a formatted disk in d0 of unit 8.

```
dopen#1,"listing file",w:cmd1:list [and of course return]
```

and when the disk activity has ceased;

```
dclose#1 [return]
```

Such a file may be loaded and read legibly by SS II. Unfortunately, the program cannot be dloaded or run in this form.

From the b, perhaps we can use some trick like a seq file reading program to print the lines on screen and then press return over them? Sounds good, but enter that close-all call. As soon as you try to press return over one of your screen lines, the screen editor decides to tokenize it and add it to the basic program in memory. It also closes all files! The 'listing file' will no longer yield more characters, and the process grinds to a halt after adding only one line to the program.

Now back to those indirect jumps. On page 3 of bank 15 there are a number of addresses of the individual pieces of code which actually do the work assigned. By re-directing those vectors, we can cause more, less, different, or even no work to be done by a call to a particular vector. AHA!

By ordering the close-all jump to do nothing, when return is pressed over a particular line of code, the line is added to the basic program in memory and best of all the file is left open and ready to read in the next line.

Included on this disk is a simple basic program which automates this entire operation, performing pokes to the keyboard queue instead of requiring you to balance a brick on the return key, and 'storing' information on screen to be picked up by these pokes. The program also deals with the somewhat less than perfect listing produced by the dopen#, etc. method described above. The leading return only lines are ignored, and the trailing 'ready.' is suppressed so the program can eliminate the syntax error message that would otherwise end the runs. Finally the program restores the close-all vector, proceeds to close the file, and even deletes itself from the end of the now entirely functional program in memory. Didn't think you could use 'delete' inside an executing program did you?

There is only one thing to worry about (but not much) with this particular code. If the incoming basic listing file has line numbers exceeding 63996, they will replace vital lines of this program. In this case you may have to manually type the equivalent of line 63996 yourself from direct mode! It is possible to carry out the procedure by also listing the line with a conditional test of st on screen, but this slows the conversion process from 9 lines at a run to only 8, and I did say this was a trivial demonstration didn't I?

```
bank15:poke 792,127:dclose#1 [return]
```

BASIC AUTO-VERIFY UNNEW and PGM APPEND VIA FUNCTION KEYS

by: Anthony Goceliak

I have recently done some rummaging through the b-128 instead of the 8050 and wish to pass along a few new tricks that the b can perform under the guiding influence of your function keys. To save typing just dload and run the program entitled 'function keys(three up arrows)'. <<Up arrows in the library pages will be shown as the ² (exponent of 2) symbol. Sorry for the oversight folks.>> The majority of the keys except for those which I will describe are documented at the summary listing at the end of this article, and are short enough to be essentially self-documenting.

For anyone curious as to why the program first re-defines most of the keys as "", [nothing], it is to prevent a memory crunch. The b operating system assigns 512 bytes to function key definitions, and although it is possible to extend this space, the final definitions do fit where they belong, and so there is no permanent need for increased memory allocation, and consequently no need to make the function key area non-standard. However, try this one on for size. Type (from basic) key1,followed by a qoute and 80 aaaaaa's. Hit return, cusor up to the line again and change the '1' to '2', thereby re-assigning key 2. So far, so good. Continue, using 3,4,etc. Right now I'll bet you didn't make it to 20. 'Out of memory'? 80 characters per key times 7 keys and you already exceed 512 bytes of storage. The original key definitions are excess baggage except of course for the few that this program keeps, and therefore to avoid the crunch, they are first wiped clean.

Automatic Verify

Pressing key F7 will yield a couple of lines of goop on screen which the b makes perfect sense of provided that you have lately loaded or saved a file. The proper drive on the proper unit will be addressed using the proper filename. Should you have connected my tape code to your b, even a tape verify will be correctly spelled out to device #1. The verify will proceed apace, and the only misadventure possible is the depression of key F7 either before or well after the file was loaded or saved (if the operating system has already altered the memory area which contained the file specs.), or if you have performed some other IEEE or tape operation which would leave wrong filename, device or drive# information. The use of such 'wrong' information will merely cause the drive or cassette to search for and not find a spectacularly wierd filename, or if one is found, to undoubtedly report a verify error.

Verifying a program is vital to ensuring correct operation, but most people don't like to type 'verify"0 [or 1]:etc etc [or whatever the filename might be up to 16 characters]",8 [or whatever device you have chosen]' each time a program has been dloaded or dsaved. A minimum of fourteen keystrokes required, and up to twenty-nine for a full length filename! This key cuts your own involvement to a single keypress, and lets the computer do the rest of the required typing (without mis-spelling the filename or sending the wrong device number either!).

Unnew

Have you ever, in basic, accidentally typed "new" when you shouldn't have? Most times it is just a nuisance, and you will merely have to re-load the program which you wiped clean. Sometimes it is a Major pain in the Neck, as when you have Finally got that stupid program to work after umpteen alterations, and you find yourself staring at the screen saying OH NO! after typing New instead of first saving the program.

The traditional way to 'unnew' a basic program is to call a machine language program which does the deed. I recently saw just such a clever routine for the C-128 published in Transactor magazine. It got me thinking about how handy it would be to have an equivalent routine for the b, but our convieniently usable memory for machine code routines is pretty well filled up by some of our Ace M/L wizzards. The problem, it seemed was to find unused bank 15 ram and write similar code. Then it hit me, why not use the function keys? All that is needed is to type a couple of pokes to appropriate locations and sys the appropriate routine in the b's own rom. This fools the computer into believing that we have added a line while simultaneously concealing the fact that the old program was 'newed' thus causing the b itself to restore all the vital internal pointers to their proper values. If you dload and run the program mentioned above, F11 does the deed for you. As a matter of fact, after you have run the program, you may test key 11 for yourself immediately, since the last line executed tells the b to new the program. Press F2, or type list and you'll see nothing. Press F11 (Shift and F1 together), and you will see my incantation of poke and sys and all of a sudden, after the 'ready', when you again type 'list', you will see, (and can run or edit) the program!

There is, of course one limitation. Variables were discarded as part of the 'new', and are not recovered by this little trick. What this means is that you may 'run' an unnewed program, and all will be just fine, but you may not 'goto 10' or 'goto whatever' to start a program, unless you realize that all variables have been cleared. If you didn't understand the significance of the two methods of starting a basic program, goto does not execute an automatic clear variables before beginning execution. If you had typed previously something like x$="abcdefg" and the first line of your program says print x$, using run to start would print nothing (x$ was cleared), but goto 10 would print abcdefg except when you have newed and then unnewed before restarting the program, when it would (just like run) print nothing.

Program Append

Perhaps you aren't impressed, but I was, and so I started thinking about other ways to trick the b. Many times it would have been nice to have two program spaces available (perhaps you have already solved a programming problem in one of your previous programs, but can't remember quite how it went, or perhaps you want to eliminate a long involved calculation and just put in the results and already have the lines typed in another program, or perhaps you would just like to 'tack on' those 400 lines of data statements which took hours to type in the last time. This kind of thing sounded great to me, so once again function keys to the rescue!

First type in, or dload your initial program. Run it if you wish. Everything is normal so far. Now press F15 another magic line gets typed for you, and again try listing or running. Don't expect much, 'cause we just fooled your b into thinking the program is gone (groan! - but not forgotten!)

Dload, or type in new lines and list, and you will only see your second program. Run, and the b will execute only the second program unless it contains peeks and pokes into bank 1, which is very rare. Ordinarily peeks and pokes are used in bank 15, which is fine. When you are done inspecting or running or whatever you wished to do with the second program you have two ways to go. Either type 'new' and then press F16, which will yield Only the initial program left in the b, or don't type 'new', but press F16. This will paste the second program onto the end of the first. At this time, may I caution you about line numbers. The second program will reside after the first, no matter if it's line numbers are smaller than the first, or even if the second program's line numbers are identical to the first!

Extreme caution is urged when proceeding in this mode, because the lines will be executed in basic's own logical order. When line #100 has finished excecution, for instance, basic will move onward in memory to check for a following line, and if it is line #5, basic doesn't care. (Ordinarily this could never happen as lines are automatically sorted numerically as you enter them, so basic doesn't perform a needless check here).

Further, if your chimera has two line #200's (which again can never happen via normal program entry), which one is executed depends on fairly complicated rules. The simple way to go about program append is to make sure that the lowest number line of your second program is higher than the highest numbered line of your initial program, in which case you have a program merge instead of a program append, and the whole thing has become one program.

I have included a simple demonstration program entitled 'whoopie!', to show one of these pitfalls. Users are invited to dload and after listing, run it to see how the duplicate line 120's are handled. After testing, try editing the Second line 120 and list again to see another problem. As I said, it is absolutely great for pasting on those myriad 'data' lines, and things like that, or for temporarily holding a second program separate from the first.

Should you forget to 'new' a second program before restoring the first, it is possible to shed the second with only a bit of work. List the combined programs to screen. Stop the scrolling with the 'chicken' key from time to time until you are sure that you have found the last line of the original program. Make sure it is comfortably on screen and press 'stop' unless the cursor is blinking already. Now type the following:
delete [insert the last line number of the original program here] followed by a minus sign and return. Next position the cursor over the original last line of your program and press return. This adds the line back again. Your original basic program is now restored without the excess baggage.

One last thing, you may wonder about the leading 'if' in key 16's definition. It allows you to press key 16 without disaster striking if you have never pressed key15 to set up the append mode.

Summary

F1 = print#15, (the ? means print anyway so why waste an f-key?)

F2 through F4 remain the same, since they are the three most useful of the pre-defined keys.

F5 = dopen# (you must specify a number anyway so why have to type the octothorpe [#]?)

F6 remains the same, since sometimes you wish to dclose, and othertimes dclose#[whatever]

F7 = Automatic Verify to correct device [and drive if appropriate] using correct filename

F8 = directory d0 with an automatic backspace. Output to screen. Press return for drive 0 directory, or "1" plus return for drive 1.

F9 = scratch" The leading quotes have been added, since people who are using syntax like scratch (f$) are probably not using function keys anyway.

F10 = chr$( ) plus backspaces to allow typing the number inside.

F11 = Basic Program Un-new Key. Restores program if you accidentally type 'new'.

F12 = list a basic program to printer#4. Printer set to upper/lower case, file opened and closed automatically.

F13 = dload d1," Works exactly like F3 but for drive#1

F14 = dsave d1," Works exactly like F4 but again for drive#1

F15 = Set up 'program append' mode. A second basic program may now be loaded and run without losing the original program even though it is now invisible. Typing 'new' will erase the second program but not the first in this mode.

F16 = Restore to pre-append mode. Original program is once more accessable, either with or without a second program pasted on.

F17 = Backup d0 to d1, carriage return, and the expected "y" reply to the b's arcane 'are you sure?'. The final c-return, however is NOT included as a safeguard to your accidentally pressing this key.

F18 = Print out directory on printer#4. Key is set to leave you the option of changing the drive number, so if d1 press return to activate, and if d0, type "0" and return.

F19 = Close the printer file left open by key F18.

F20 = print#15,"m-w"chr$(Used to send m/l code or data directly to disk memory. I use this one a lot, but you may not.

Your b is getting smarter. Enjoy it's new commands.

QUICKLY RESTORING THE SCREEN TO DEFAULT SETTINGS
by: Anthony Goceliak

There has been mention in recent editions of the escape of a wish for programs to 'restore things to normal' on exit, on the whole a noble desire. Sometimes however, this is easier said than done, although when writing a program you tend to pay attention to little things like whether you have silenced the end of the line bell, it is not as easy to keep track of as perhaps a screen window or shift to graphics mode. And when 'fixing' some long-ago written gem, it may take literally hours, and countless program runs to track down the last default change.

I recommend writing with the aid of a small scratchpad to keep note of any special screen treatment, but for programs which have already been written, passing over some of the lesser-impact features is easy. As long as you can find all the program exits however, the following goodie will re-initialize the screen - clearing it and eliminating windows, reversed screen, graphics mode, bell suppression, etc. Note carefully that it affects only the screen treatment and does NOT affect anything else. (Programs are NOT 'newed', nor files closed, etc.) Furthermore, since the code is a jump through the CBM kernal jump table to another jump through the kernal low-jump table to the 'real code', it should be supported by any version of the b, and I suspect, some other Commodore 8-bit machines.

Readers who have implemented my program 'screen configure', the pseudo b/256 screen for the b/128 should remember that this will return the screen display to the mode available on power-up, disconnecting the special screen treatment exactly as if you had powered down and up again.

Insert the following line either at every place your program can end, or preferentially, direct all possible program exits to one place and include a single standardized 'orderly shutdown' section. The line number may obviously be made to any line 63999 or less.

```
1000 bank15:sys 65406:end
```

M/L AIN'T HARD UNLESS YOU FOLLOW A BAD EXAMPLE
by: Anthony Goceliak

Ordinarily I don't like to point out shortcomings in other people's work, mostly because my own work is so riddled with the same, but since receiving a torrent of letters (two) on the same trouble, here goes.

Apparently several CBUG members are attempting to learn m/l programming, a highly commendable effort. Having read a book or two and feeling shaky but willing to try a more difficult task than the examples that were in the books, they turned to the Protecto Programmer's Reference Guide and attempted to assemble the 'example program using kernal function' found on page 233- in appendix a.

They failed miserably, exactly as I did when I first got the PRG. Oh yes, the bloody thing assembles, but it is frankly RIDDLED with bugs, some obvious, and quite a few more subtle. Apparently the listing was borrowed from another example for another Commodore Computer and not adapted at all or the original listing instead of the corrected one was published in error.

Therefore, if you are a nascent m/l programmer, don't lose heart because you couldn't get the program to run. Probably THE toughest thing to do in m/l programming is to correct someone else's work. And the second toughest thing is to 'save a lot of effort' by using pre-canned routines (as in the kernal calls here) when the descriptions of the kernal calls are 20-30% wrong. As a simple example which I haven't seen mentioned before, the routine 'readst', mentioned on page 207 should be shown to include the following tidbit.

If the routine is called with the carry bit clear, then it will SET, not read the st byte.

I never use this particular routine, although the 'example' did, but simply replace it with lda $9c, two bytes, instead of sec:jsr $ffb7 which runs four. Oho, you say, what about portability between Commodore computers and the universality of the kernal jump table? I don't own any other Commodore computers, so I don't worry about it, and until you get Quite a few m/l programs under your belt, I reccommend that you don't either.

LIVING WITH PRINTERS

by: Mark Schwarzbauer

What good is a computer without a printer? Not much. In order to be a complete "system" a printer is a necessity. Protecto bundles printers with most all of the systems they sell. Currently they are bundling a thermo printer made by Cannon for the IBM P.C. Jr. called the "Big Blue" printer. It is the epitomy of a "cheap" printer.

Printers, like disk drives, have moving parts which can wear out or fail if mis-used. When purchasing printers they normally stipulate how many characters can be expected to be printed by the print head before it dies. Some are so bold as to claim a "lifetime" warranty. I think most of the manufacturers offering a lifetime warranty assume you will update before you reach the point where the print head fails. Other parts can fail also in printers. Motors can burn out, logic boards can have a chip go bad, gears can stripped, rollers can wear out, and the list goes on.

In this article I want to address some of the common printer problems people have been calling me with. I'm going to deal with the CBM products, for the most part. However, what goes wrong on these can also go wrong in other brands as well. I know most people, if not all, save their issues of the "ESCAPE" for future reference. This article may come in handy down the road so keep track of it so you can read it before you call me with a problem.

First, let's deal with our dot matrix printers. Both the 4023, and the 8023 are for the most part well built printers. The 4023 is much less "quality" compared to the 8023 by virtue of the fact that it is designed for home use. However, compared to many other models of "home user" printers the 4023 is far superior. The 8023 is designed much more ruggedly. It accepts wide carriage paper for ledger reports whereas the 4023 is strictly designed for the typical fan fold letter size paper. The larger carriage on the 8023 reflects the "business" nature of it.

The 8023 was actually designed, or should I say redesigned for the B series. Bruce at Northwest Music has a version of the 8023 that has a case changed to match the smooth contours of the B series computers. remember, the B series was designed to be a business machine.

There are several important points to remember in the operation of these printers. Your manual touches on some of them but leaves out others. Let's take a look at a few important tips on the care and operation of your printer.

1. NEVER, NEVER, NEVER, TURN THE ROLLER BY HAND WHEN THE PRINTER IS ON. You will eventually strip the gears. The 4023 users have reported more problems in this area than the 8023 users. Perhaps the 8023 users know better. Maybe that is why they are now using an 8023! <<8023's have hand clutch like typewriters.>>

2. NEVER USE INK RIBBONS IN THE 4023. O.K. I know that someone told you it was o.k. to use Epson FX refills in your 4023 ribbon cartridge. But you will eventually mess up the print head. Ink ribbons will clog the pins on the print head and the printer will eventually stop printing. So how do you fix this. Yes, it is possible to fix this problem. You can either clean up the current print head or buy a new one. Instead of buy a new one I recommend whenever you have a 4023 die that you upgrade to an 8023. To repair the old print head start by removing the ribbon. Next, take a cotton swab dipped in rubbing alcohol and wipe off the face of print head. Continue to do this until you no longer get ink off the print head. Now with the ribbon removed we are going to clean the print head further by printing without the ribbon onto paper. The simplest way to do this is to run the printer self test by holding the advance button down while you turn the printer on. As the pins come out to print they will be pushing out ink that remains. Run it like this till most of the paper is clean after the print head has passed by. I recommend you repeat the procedure again before reinstalling the new carbon ribbon.

3. NEVER USE CHEAP RIBBONS IN YOUR PRINTER. I know it feels nice to save money but using a cheap ribbons can strip gears! If the ribbon jams the advance gears will begin to grind. I had this problem with a CERTRON brand ribbon. The ribbon was not cheap at $14.00 but it was cheap in quality. On the 8023, I recommend purchasing ribbons from Great Lakes Ribbon Company. They sell a very good quality ribbon in boxes of 6 for around $4.00 per ribbon. The 8023 uses Centronics 150 ribbons which can be purchased at many office supply stores under the Centronics 150 designation. Cheap 8023 ribbons have a tendency of the front foil piece coming off especially during the printing of labels. Of course Murphy's law of selective gravity details that when the piece comes off it will fall into the place most likely to jam the printer and that is exactly what happens. In the 6400 ribbon selection is a challenge. I have yet to find a good Diablo Hytype II ribbon that works consistently well in the 6400. I go through one every two weeks or so and I have tried a number of different brands.

4. DO NOT LEAVE 2 DEVICES OFF WHEN TRYING TO LOAD PROGRAMS. If you have several printers on line like I do you have probably already discovered that you cannot just turn on the computer and the disk drive and load when both printers are off. The result will be that the program will mis-load and of course not run. Good to remember for the future.

5. DO NOT PURCHASE PRINTERS THAT HAVE IEEE NOT MADE BY COMMODORE. I purchased an Epson DX20 because of it's IEEE on the API (all purpose interface). In the users guide to the printer it stated that it would run on CBM machines. They were very wrong. The serial interface also was not properly operating on the DX20. Since I spent all that money on the Epson because I thought I was buying a trusted name, I have noticed the DX20 is no longer for sale by the companies that were carrying it. Even though I was able to sell it (at a loss of course) to someone who could use it on the parallel interface I still have a $50.00 cable that is utterly useless. Thanks a lot Epson!

You can buy Serial or parallel port printers and connect to the rs232 or use an IEEE to Centronics adapter and most of them will function just fine. The SUPERPRINT DISK details the hookup procedure for a number of other printers.

6. REPAIRING SHEET FEEDERS. For those who purchased a 6400 with a sheet feeder or another machine with a sheet feeder you will eventually run into a misfeed problem. The problem begins as the paper doesn't advance when the grip rollers are turning. It grows worse into constant misfeeds. But there is a simple solution. You can be extravigant and replace the gripper wheels (if you can find new ones). Or you can follow this simple procedure to repair them. Notice the wheels have treads on them that grip the paper. These wear down and need to be roughed up again. Remove the sheet feeder for convenience. Start by cleaning them with rubbing alcohol. Next, use a heavy grit sandpaper and rub the wheels horizonally to etch in new tread grips into the wheels. Replace the sheet feeder and you are ready to print without those frustrating misfeeds.

7. PROTECT THOSE DAISY WHEELS. If one of the fingers on the wheel becomes out of line with the wheel it will grab your ribbon and push the ribbon behind the wheel thus printing nothing onto the paper or it may cause the ribbon to keep jumping around and skipping letters. If this happens to you it is most likely a faulty daisy wheel.

These are a few of the very important things to note about your printers. Currently, Northwest Music is offering a printer buffer that will work on our IEEE interface. This means that instead of the computer waiting for the printer to finish before you can use the computer again it loads the text for the printer into the memory of the buffer and instantly frees up the computer while you printer goes on printing.

Is there anyone out there that knows about the CBM 6400 (C-Itoh 10-40) buffer expansion board? Let me know if you do.

There is more information about interfacing different printers on the rs232 and on the IEEE located on the "SUPERPRINT" disk available exclusively through CBUG. That's it for this month. If you have any questions give me a call and I will try to help.

Mark Schwarzbauer
414-743-4151 Before 9pm central time please

◆◆◆◆◆◆◆◆◆

A FIX FOR THE 8050
by: Gene Lambert

I purchased my B-128 system in stages, First the computer, then the 8050 drive and monitor. (I already had a LQ printer.)

When I got my 8050, it had been through heavy use as the main drive in a bbs, and needed an alignment. After this was done, I naturally figured that would be the end of any problems, and sure enough, it was....for a while, that is. About a week after getting my just aligned drive back, I started getting strange problems, like when I turned on my 8050, both drive motors would run continuously, or I would get strange and erratic blinking of the drive lights halfway through a hot session with Superscript. Upon investigation, I found the problem.

Namely, there are several cables with connectors that tend to develop open circuits. They are one female ten circuit connector, P4 on the digital board, and two smaller female 6 circuit connectors that interconnect the analog and digital boards, P6 on the analog board, and P3 on the digital board. Both P3 and P4 are on the digital board, at the left along the bottom. (The analog board is fastened to drive 0 on the 8050, 8250, and to drive 1 on the 4040, the digital board is always the large square board that is fastened to the top cover.)

The best way to troubleshoot your 8050, 8250, or 4040 is to remove the two phillips screws on the side, then prop open the cover, and being careful not to touch any exposed connectors, wiggle every wire going into P3, P4, and P6, with the drive on. (The numbering on the 4040 may be different.)

If you find a connector that fails this test, now is the time to replace it. It's very simple, and will definitely save you a large repair bill at your local friendly Commodore repair facility!

Call your local electronics stores, until you find a retailer who sells the "Molex" brand of connectors, distributed by Waldom. Ask if they carry the six-circuit .156 connectors, package number WMLX-212, and the ten-circuit connector, package number WMLX-206. I recommend replacing all three connectors if you find one bad one.

If you decide to replace one or more connectors, please draw a diagram of all wires before you start, and only fix one connector at a time, to avoid mistakes. If you have the correct crimping machine, then use it. However, I recommend soldering.

After your through replacing the connectors, check your work very carefully before appying power to your drive. If you don't, the repair shop may get your drive after all!

One other thing...I always clean all male connectors, and gold fingers with a common pink eraser. It is the preferred method of cleaning contacts, and will not damage delicate connectors. I even use it on IC pins, although you have to be extremely careful when doing this, to avoid breaking the pins.

Gene Lambert, 322 S. Clark
Orange, Ca. 92668
(714) 771-3593

◆◆◆◆◆◆◆◆◆

INSTRUCTIONS FOR "data cards.jcl" PROGRAM
by: Mathew Goldstein

<<This and other material from Mathew are available in this issue's library section on CBUG #M80, stock #12064>>

PROGRAM FEATURES: disk resident database with user definable fields of up to 255 characters per record. From one to 29 records per set, with as many record sets as the available disk space permits. Each record is called a card, and the cards that make up a set are assigned page numbers from one to 29. The sets are also numbered sequentially. Each time you create a new database file you decide how many records there will be in a set, whether or not the database will be unrestricted in size, the size of each card in a set and it's design including the location and size of fields, the type of field (alphanumeric, alph, numeric, date, or empty), and whether or not the card is to be framed. Sets can be relocated using the Move command, sorted by the first field of any of the available pages, and erased. Cards can be copied, and deleted. You can manuver thru the database either within a set or from set to set within any page. You can also traverse the pages within the sets in any order you choose. You can search a particular page of all the sets of the database to locate particular character strings including searches to match the contents of more than one field simultaneously. Pages can be printed. The contents of any card can be changed.

When creating a new database you will be prompted for the top, bottom, left and right borders for each card page. The top row must be within the 2nd thru 10th rows, the bottom row must be between the 11th and 20th row, the left column must be between the 2nd and 20th columns and the right column must be between the 60th and 78th column. Therefore the maximum card size is 2nd thru 20th row and 2nd thru 78th column and the minimum card size is 10th to 11th row and 20th·the 60th column inclusive. The larger the card size and the more pages per set the more space the file will use on the disk. Careful pre-planning of the design of the database before you create it is necessary is the best way to avoid regrets later. The databases will be loaded and stored on drive #1. Try created several trial databases to get a better picture of what is involved.

This database is particular for situations where it is preferable or necessary to maintain data sets so that information in one part of the set is not visible untill after you have looked at another part of the data set. An example of such a situation is a database designed to teach vocabulary or a foreign language. Such a database could contain three pages per set. The first card could have fields for the word and it's pronunciation, the second card could have several sentences using the word, and the third card could have a field for the part of speech and several fields for definitions. A one page database could be used to maintain an ongoing newsletter or magazine index or to develop an algorithm written in psuedocode for a computer program. The cards could be printed out and cut to size and handled like a deck of index cards.

This program is a good example of the JCL Workshop extended BASIC screen input system in action.

COMPUTED GOTO and SOME 8432 QUESTIONS

by: Mathew Goldstein

I prefer the on/goto and on/gosub syntax to computed goto because it does not restrict the target line numbers to numbers that fit some particular formula. Nevertheless, the computed goto as explained by Mr. Goceliak does offer the advantage of being more compact to code in situations in which there are 25 or more different directions to go to. In such cases the on/goto and on/gosub routine would look something like this:

```
80 gt% = 0 : input gt%
90 if ((gt% < 1) or (gt% > 25)) then 80
100 on (int(gt% / 5) + 1) goto 110,120,130,140,150:stop
110 on gt% gosub 1111,2222,3333,4444:goto 200
120 on (gt% - 4) gosub 5555,6666,7777,8888,9999:goto 200
130 on (gt% - 9) gosub 10000,11111,12120,13130,14140:goto
       200
140 on (gt% - 14) gosub 15150,16160,17170,18180,19190:goto
       200
150 on (gt% - 19) gosub 20200,21210,22220,23230,24240,25250
200 rem continue
```

If there were one hundred branches the first on/goto would divide gt% by 20 and the five lines that the first on/goto branched to would each divide gt% less an appropriate multiple of 20 by four etc.

There is another, simpler way to implement a computed goto on Commodore computers that have the trap statement. Unlike the goto and gosub statements, the trap statement will accept integer variables as line numbers. Therefore, the following should work:

```
100 trap (gt% * 100) : syntax error here to force the jump
        to gt%*100
```

Many of the articles and software contributions are excellent, as a contributor I know first hand how much time goes into the writing. Let's keep Norman busy.

I would like to convert one of my programs to run on the new 8432 emulator. I have some questions. How are relative files implemented without the use of the record# keyword? Is there any way to quickly load the directory without erasing the current program? How can a program determine which unit it is running in? Can a program running in one unit load another program into a different unit and transfer control to the other unit? Are there 8032 programmer's manuals for sale that are written in English? What ram addresses are available for m/l programming and what m/l subroutines are available?

INPUT WITH COMMAS AND QUOTES

by: Mathew Goldstein

There have been several proposals for writing BASIC code to handle data input.

There are several problems with the input and input# commands. The input command always puts a question mark on the screen, and does not allow the programmer any control over the size or nature of the data enter. The input# command treats the comma as a field delimiter. Liz Deal came up with a screen input routine that works although a doubt there are many people alive today who know how it works. I am not expert enough to be so fancy but I may be able to offer some suggestions for others interested in taking control of data input using standard BAISC programming techniques.

Appendix D of the B Series Commodore User's Guide lists the chr$ codes. One of the curious features of the chr$ codes is the duplication of character numbers 33 through 63. The duplicates are located 64 locations up at character numbers 96 through 127. These are the character codes for all of the non-graphic, non-alphanumeric characters. Included are the troublesome comma, semi-colon and quotes characters. Therefore, the comma can be represented by both chr$(44) and chr$(108) and the quote by chr$(34) and chr$(98). By shifting all of the character numbers from the 32 through 63 range to the 96 through 127 range all of the input# problems associated with these troublemakers disappear.

Using the get command allows character by character control of the input although it will require more effort to code. Some of the options which you may want to control as the programmer include the maximum and minimum length of the input data, the location the input appears on the screen, and the display of a default input response. Here is an example of code that I believe will be able to handle such features:

```
5 rem sample alphanumeric input subroutine example
19 rem
20 rem calling routine
30 lt% = 9:lm% = 8:sc% = 30:sr% = 12:df% = 1:df$ = "default
     data"
40 qu%=0:rem optional count number of quotes in input
     string
50 gosub 100:if in$ = "" then in$ = df$:rem subroutine call
60 rem continue with main routine
99 rem
100 rem subroutine starts here
130 in$ = "":get g$,g$,g$,g$,g$,g$,g$,g$,g$:rem clears
     keyboard buffer
135 poke 202,sr%:poke 203,sc%:print " ";chr$(157);
140 if (df% <> 1 or len(in$) <> 0) then 155
145 poke 202,sr%:poke 203,sc%
150 print chr$(27);"q";df$;:poke 202,sr%:poke 203,sc%
155 g$ = ""
160 get g$:if g$ = "" goto 160
165 if g$ <> chr$(13) then 180:rem for end of input
     <return>
170 if (len(in$) >= lm% or (in$ ="" and df% = 1)) then 520
175 goto 155
180 if g$ <> chr$(20) then 200
185 if len(in$) < 1 then 155
190 in$ = left$(in$,len(in$)-1):print g$;:goto 140
200 rem other routines for character numbers less than 32
```

```
300 if g$ < chr$(32) then 155
310 if len(in$) >= lm% then 155
320 if g$ = chr$(34) then qu% = qu% + 1:rem optional
    counts quotes
330 if g$ < chr$(64) then g$ = chr$(asc(g$)+64):rem chr$
    code shift
400 rem other routines for character numbers 96+ including
    graphics
500 if len(in$) > 254 then 155:rem 255 is max allowed char.
    string length
510 in$ = in$ + g$:print g$;:goto 155
520 if qu%/2 <> int(q%/2) then in$ = in$ + chr$(27):rem
    optional
450 return
```

lt% and lm% are the maximum and minimum input character string lengths, sr% and sc% are the screen and row locations where the input data is displayed, if df% = 1 the default input data stored in df$ will be displayed. My experience with the poke 202, poke 203 commands is that a print 'space cursor left' is necessary to ensure that the first character displayed is at the poked screen row and column. The chr$(27)"q" is an escape code that erases the screen to the end of the current line. Escape codes are listed in the silver B Series Commodore Users Guide. The above code shifts all of the characters in the 32 to 64 range even though it is not necessary to shift more than the comma, semicolon, and quotes under most circumstances. Line# 400 is redundent with line# 210 so it can be omitted. The chr$(27) added to in$ cancels the quote mode. Most programs can be written with about two to four such subroutines including an alphanumeric input subroutine, a numercial input subroutine, and maybe one or two specialized input subroutines.

Generally speaking if you use a comma when entering information that you are prompted for by the program that you are using you are taking a risk. Many programs, especially public domain programs, not including my latest versions of the checkbook program, will either not accept the input from the comma onwards, treat the input as mulltiple inputs, print an error message (such as 32 type mismatch, or 34 fle data, or 15 extra ignored), or numerous other undesired mishaps. If you accidently use a comma when entering data for a file that is saved to disk and find that you get a file data or type mismatch error when the file is read from the disk you should edit the file with a wordprocessor such as superscript and remove or replace the commas that were entered during data input. The chr$(44) type commas are treated as delimiters by the computer and disk drive which means the computer will conclude that it has finished the input task whenever the comma is encountered.

The JCL Workshop Extended BASIC is particularly strong in the area of data input. The respond and getkey are improvements over the corresponding input and get but they both have limitations. Getkey in particularly cannot handle the delete key, for example. The special screen input system commands available in the JCL Extended BASIC package have no equivelant in Commodore BASIC and are particularly useful for programming database type screen input with pre-arranged fields as in Superbase. My 'data cards.jcl' program is a good illustration of this capability.

◆◆◆◆◆◆◆◆◆

?? QUESTIONS ??

by: Gene Lambert

Here is a list of questions that I, (and perhaps others), would like answers to.

What are the differences between the Superscript II and III? Why do most prefer Superscript II? Was the key-bounce problem and other bugs fixed in SS III? <<See previous articles on SSIII in THE CBUG ESCAPE>>.

What about other word processors? Are there any that are menu-driven? Who sells them if there are? <<Word Result from Handic -- compatable with Calc Result. Available from N.W. Music -- see ads>>.

C/PM: I am confused about the 8088 upgrade, versa the Z-80 upgrade. I always thought that Z80-based machines ran C/PM, and 8088 machines ran MS DOS. Please explain. <<8088 family chips also run CP/M 86 which is a version of the Z80 CP/M adapted to run on 8088 type chips. The software to be used is also re-stated in the CP/M 86 variant>>.

The infamous "Alternate character set". Is anyone even talking about implementing it, and/or fixing the equally infamous upper-case "O"?

Could someone please take the time to fully explain the features available in all of the different term programs out for the "Bee"? I want to know about protocols, baud rates, bbs menu's, or lack of, autodialing from within, etc. For an example, my favorite C-64 term program, Commterm 3, has 2 bbs "pages", that allow you to set your baud rate, name, number, protocol, (x-modem or new Punter) that are dialed, all independently. You can dial multiple bbs's sequentially, all if you want. Please tell if menu driven, or command driven.

Ok, here comes the dreaded hardware questions: Please bear with me.....
I have a D9090. Can anyone tell me what are the model and catalog numbers for the drive itself & the controller board? (I have the motherboard and chassis.)
What is the clock speed of the 6509? (you wouldn't believe the amount of people who have asked me that!<<Check with N.W. Music regarding the D9090>> <<The clock speed of the 6509 as implemented in B128's is reported as 2 Mhz in most machines and 1.8 Mhz in a few of them.>>

Are schematics, etc available to help us "techhies" build our own memory boards, 8088 and Z80-based co-processor boards, etc? Also, it would be very nice to have a CBUG disk-based issue devoted just to hardware hacks of our favorite machine. <<CBUG published way way back the schematics for the co-processor board Archive #001 stock # 12370, $25.00 including preliminary code listings. You must have a confidentiality agreement on file with CBUG or submit one with the order.>> <<I'd advise against trying to build such high density boards except on etched foil boards.>>

<<I think this type of interchange is an excellent idea -- and in line with our regular author's call for input. Lets see more of this!.>>

Gene Lambert, 322 S. Clark
Orange, Ca. 92668 U.S.A.
(714) 771-3599 7-10pm PST.

◆◆◆◆◆◆◆◆◆

HARRISON's HACKER CORNER

by: Howard Harrison & Dave Porter

Dave Porter and I thought it would be a good idea to see small entries in the ESCAPE containing ideas and future projects. This article is intended to get the attention of other b-128 "hackers" that may not enjoy writing. I enjoy programming and I dislike writing, but I feel there is a great need to communicate with others. Please respond to me with a letter, a phone call, or a disk, containing tiny explanations of personal projects. Even if you just have ideas, incomplete programs, bad english, or any other excuses!!!! Just let me know what is going on. If you send a letter or a disk, please include a phone number, because I send very few letters!

I have started many projects on the b-128. Many of these are incomplete. Please respond even if your projects don't seem as "advanced" as the following, because external input is very important. Here is a small list of projects that I am doing or have done.

Hardware:

"bank8" - Developed by Dave Porter and I. A device that allows the I/O portion of the bank 15 memory map to be available in banks 8 - 14. The user adds the additional memory. In this system, there is little need to use transfer sequences to use kernal routines. Just poke into the execution register to change banks! Think of the flexability, just copy the kernal from bank 15 to your user bank & fly. Think about multitasking (my multitasking program has tremendious overhead.) -- I have completed this one!

Software:
"active user" - An all-purpose multitasking program. A user program can attach to this and submit jobs to run in the background. I can run my ASSEMBLER from superscript and then return.

"scrhost" - This nifty code allows a user to remotely control the b-128. Unique screen update only sends the changes in screen memory to the terminal device. This will soon be compatible with VT52 or VT100 terminals.

"qdms" - A buffer-oriented data base manager. Allows applications to share a common data base program using a command channel. I use disk cache for index and data records to allow high-speed access. I intend to use this with my assembler to allow an unlimited number of labels and object file linking. I may soon need other applications to make this usable in the real world!... Any ideas?

"spooler" - This multitasking subprogram allows the user to use the disk drive while spooling, remove the disk, spooler recovers after replacing it! Once this is modified to run with "active user", this will be able to coexist with superscript.

"page/print" - This allows multiple column printing if your text is narrow. Page/print is extremely useful for directory, assembler, and disassembler listings. - This has been active for a couple of years in my system!

"newopen" - This allows a user to set up many "logical" device names to access disk drives, and printers. It also scans multiple disk devices to find a given filename, this is useful to me because I have an 8050, 9090, 2031, and 2 sfd-1001's all with different device numbers!

Howard Harrison   Dave Porter
418 Centre Ave. Jeffersonville PA 19403
(215) 630-6693 -- 7pm - 11pm (EST)

HOW TO PREPARE THE 9060/9090 HARD DRIVES FOR TRANSPORTATION

by: Howard Harrison

How to Prepare the 9060/9090 hard drives for Transportation

I am one of the few 9090 hard drive users. I never thought Commodore had implemented the ability of preparing it for transportation, in other words, "park the heads". I had discovered this important, undocumented feature by mere LUCK! In order to park your heads, just type (in basic):

```
        dclear d0 <cr>
          *** or ***
 open 1,8,15,"i0":close1 <cr>
**** AND THEN TURN POWER OFF!! ******
**** any following commands will "unpark" the heads!!!  ****
```

When power is restored, the 9060/9090 will unpark itself.

If you recall the Disk System User Reference Guide, Commodore implies the "uselessness" of this command for hard drives! I purchased my 9090 nonfunctional -- bad disk drive. I had a replacement drive waiting for it. After servicing the drive, I had formatted it and sent an "i0" to the command channel, the head mechanism went to the innermost track and stayed there, in effect, parking the heads.

Howard Harrison
418 Centre Ave. (215) 630-6693
Jeffersonville PA 19403

◆◆◆◆◆◆◆◆◆

ICPUG SUPERBASE and SUPERSCRIPT CORNER ARTICLES

<<These are provided to us from the Independent Commodore Products Users' Group of London, England. CBUG member Roy Sherman has spent many hours going thru this information to boil it down to materials of direct interest to CBUG members.>>

The article on direct data read was left off completely since no version is available for the B.
Maybe someone would like to adapt it?

Super* Corner Sep/Oct Pertains to the C64, but should work on the B.

<<The full text files of ICPUG SuperBase/Superscipt files is included on CBUG #80 Misc, stock # 12064>>

SUPER* CORNER, MarApr87

edited by Jim Kennedy

THE RETURN OF THE CARRIAGE
Paul Blair from Down Under sent the following short article which I shall pass on unexpurgated.
This all has to do with the unwanted carriage return and its sworn enemy, the semi-colon. Superbase does a lot of jobs for me, among them a mailing list or two. Way back when my old PET printer did the work, speed was not so critical. But with more jobs to do, I try to get a tiny bit efficient.
Rather than use Superbase sort etc. to fling words out to my printer I find it much quicker (by about 50%) to write a sequential file and sort and print labels from BASIC. The Superbase commands "OPEN" and "DETAIL" work quickly and accurately, so I do (or rather, did) this:

```
10 file "members"
20 open "h8grabbit":select f
30 detail &[surname]&[adr1]$[adr2]&[adr3]
40 select n:eof 200
50 goto 30
200 close:across:wait:menu
```

That worked fine. Now some BASIC. As there are 4 fields written out, I recall the fields in groups of 4 strings, do some setup, and print them. All that is fine until you get to the last record.
My BASIC uses INPUT# for speed reasons and I test the variable ST for the end of each file. But Superbase has done it! After the very last complete record is a carriage return - just one - that gets in the way. INPUT# and ST get tangled up and my BASIC can't recognize the end of the file. Woe!!!
Sure, I could use GET# in place of INPUT#, but the only reason for all this is a whack of speed... so cancel GET#. But here is where my trusty semi-colon comes in. If I rewrite line 30 to

```
30 detail &[surname]&[adr1]$[adr2]&[adr3];
```

everything works for me. You did notice the semi-thingo on the end?

After some other excursions into this sort of thing, I suggest you include the semi-colon when writing sequential files in this way. Just why the last field in a file made this way is a carriage return only Precision know. Maybe its needed, but if not, perhaps they could be persuaded to amend Superbase to get rid of it.

Super* Corner Editors Notes to above. The carriage return is used not only as a field and a record separator but a file separator as well. In the Amiga version one can change the field separator default, a comma, and record separator, a carriage return. I would use the OUTPUT TO command to avoid the carriage return problem. Please be aware that some versions of Superbase 128 have a bug in the OPEN command which I understand Precision are working on fixing. This presents few serious problems as one can always print to disk using

```
10 pdev 8,8,8 )
20 print "listing"
30 list
40 pdev 0
```

which would print the current program listing to a disk file. See Superbase The Book page 159 for more details.

COLLECTION OF PRECISION SOFTWARE LTD NOTES
Pete Maclaurin of Precision Software Ltd contributed the following notes to the column for this month. These notes are based on the notes which PSL send out to customers. I shall include them with only minimal editing so as to insure they remain similar to those distributed by PSL.

SUPERBASE LABELS PROGRAM
Extracting information from the database for printing labels (when doing a 'mailshot' for example) requires three operations:

1) Create a file of information in the database and make a find list or sort list containing the record keys that you wish to print labels for.

2) Set up a label definition to match the required format.

3) Merge the data (selected by the key list if required) into the label format and print the result.

To cover these points in more detail, first create the data. To do this all your data must be in one file of the database. This file should be the current file, and you must know the names of the fields you used when formatting the file. You must have enough room on the disk to take the label definition, the labels program, the makelabels program and the key list if you intend using one. Use the find and/or sort commands as described in the Reference section of the Superbase Manual if you want to select records instead of printing the whole file

Secondly, define the label format. Move both 'labels' and 'makelabels' from the Superbase System disk to your work disk by loading them from the system disk then saving them on the workdisk. (Or use the utility or Copy-All if you have twin drives). Execute the labels program. Follow the prompts to create a definition for your labels, noting in particular:

a) Use an existing layout file? ... answer n first time or if you don't want to re-use a definition created earlier. Answer y if you do.

b) Layout filenames ... lstore is the default name, but you can use any name that you have not already used for a file, keylist etc.

c) Lines between labels ... do not choose 2, even if the version of the program you are using prompts you to do so!

d) Data types ... key fields are Text (t) type.

Third, print the labels. Having defined the format, the labels are printed by the Labels.p. This program will run the program Makelabels.p to define the format, Makelabels.p will in turn run Labels.p once more, which does the printing. The program is in two parts to allow it to run in machines where the program space would otherwise be too small.

SEARCH DATES FOR A MONTH
Using Superbase to search dates for a month (year independent searching) is as below and the format of [date] is English rather than North American. The file definition must include an extra field at the bottom called [flag] or have available an unused text field to use instead.

```
10 ask "Month to report on" b$ : date "01"+b$+"86",a :
    if a=0 then 10
20 select f
30 convert [date],a$
40 a$=mid$(a$,3,3)
50 if a$=b$ then calc [flag] = "1" : store
60 eof goto 1000
70 select n : goto 30

1000 find "" where [flag] is "=1"

2000 batch all [flag] = ""

3000 menu
```

This generates a key list of all records where the month matches and clears the flag field for a subsequent enquiry.

DETAILS OF SUPERSCRIPT FILE STRUCTURE USING TABDATA
In Superscript end of text is always marked with a null/zero byte ($00). If TabData follows then a 255 byte follows the null byte ($FF). (TabData is not saved with a ranged block, nor with early versions of the wordprocessor program. EasyScript does not save TabData for example.) After the 255 byte there are a further 152 bytes of Tab information as follows:-

a) Byte 1 is a count byte for the number of Horizontal tabs set. (default 25).

b) Byte 2 is a count byte for the number of Vertical tabs set. (default 0).

c) Bytes 3 to 52 (50 bytes) are the Horizontal tab information. (see below).

d) Bytes 53 to 102 (50 bytes) are the low bytes of Vertical tab information.

e) Bytes 103 to 152 (50 bytes) are the corresponding high bytes.

f) Horizontal tabs: 255 indicates no tab. Other numbers up to 254 indicate the column in which the tab is set.

g) Vertical tabs: 32767 or 65535 indicates no tab. Numbers grater than 32767 indicate that the corresponding Horizontal tab is a text (as opposed to numeric) tab.

In other words the first 15 bits (0 to 14) are the vertical tab position. The highest (16th bit 15) position indicates the type of Horizontal tab (Numeric if clear, Text if set). Default tabs are set as Horizontal Text Type at positions 0,9,19,29,...etc. up to 139. All tabs can be cleared or set from the SuperScript menu.

MORE ON MAIL MERGE
Using Superbase and Superscript together for Mail Merge is handled by extracting information from the database for use in a standard letter (when doing a 'mailshot' for example) and requires three operations:

1) Create a file of information from the database. This is a 'sequential' or 'fill' file.

2) Set up a word processed document to match the format of the file.

3) Merge the fill file into the document and print the result.

To cover these points in more detail we first create the fill file. Here all your data must be in one file of the database. This file must be the current file, and you must know the names of the fields you used when formatting the file. You must have enough room on the disk to take the fill file, bearing in mind that the fill file is not compacted in any way. Use the output command as described in the Reference section of the Superbase Manual. Use the fill option.

```
   output all fill to "fillfile" [name][street][town]
```

This creates a file containing the fields in the order you specify (this may not be the order they appear in the database), empty fields appearing as blank lines. In addition an extra blank line is used to separate one record from the next.

You may wish to select only those records you have previously found by using the find command. For example:

```
   find "findlist" where [county] is "=Bucks"
then
   output from "findlist" fill to "fillfile" [name][street]
     [town]
```

Having already created a fill file you may wish to add to it. This may be done by using the append syntax, for example:

```
   output from "extralist" to "fillfile,a" [name][street]
     [town]
```

The ',a' here will add the information found by "extralist" to the existing fill file "fillfile". This is called appending.

Remember that if the fill file you create is longer than the maximum text length allowed in the wordprocessor you will not be able to edit it. If this is a problem, create more than one fill file (each of a shorter length with a unique name) by using key lists created by the find command.

Secondly, you must set up the document to receive the data from the fill file. This is described in both the Reference and Tutorial sections of the wordprocessor manual. One further point should be noted however. Because the records in a fill file are separated by a blank line, it is often necessary to 'trap' this line in a 'dummy' variable block:

```
   *nb this is a dummy block <  >
or
   *cm this is a dummy block [ ]
```

The exact syntax depends on the particular version of Superscript you are using. The comment and the 'dummy' block should be placed after all the 'genuine' blocks in the document, the best place is the end of the document. Placing a block in a comment in this way 'traps' the blank line in the document and does not print it.

Thirdly, you must merge the document with the fill file. Once again this is described in the manual. Several points are worth emphasizing however. If you 'sample' the output, remember to clear the document and reset the fill file to the start before you do the final printing operation. Merging in this way does not involve the naming of the fill blocks at the top of the document.

Finally, should the result still not be as you expected, examine the fill file by loading it as if it were an ordinary document. Remember that the format of each record in the file must be the same and that the format must match the merge document.

SFD 1001 DISK DRIVES

Think of the SFD as half of an 8250 drive! Commodore supply the 'Disk System User Reference Guide' with the SFD 1001, but it hardly mentions the SFD! No problem if you know that an SFD is half an 8250 (The 8250 is a dual drive - rather like having two SFDs in one box). OK. Now when reading the manual you can read about the 8250 - ignoring all references to the second drive of the 8250 (Drive 1).

The Commodore 64 uses BASIC 2.0 not 4.0. so commands like HEADER are not available. You will find that some of the commands in Appendix A are available on the Commodore 64 and some are not. This is because some have been introduced in later versions. Ignore references to DOS SUPPORT unless you have a PET/CBM 2000, 3000, 4000 & 8000 series computer.

'Two single drives do not a dual drive make'. A dual drive is one that was manufactured as such. Two independent drives connected together do not constitute a dual drive. You cannot use commands like BACKUP and COPY from one independent drive to another - these commands only work with dual drives. (3040, 4040, 8050, 8250 etc.) To use two drive units together one unit must have its device number changed. This can be done in one of two ways: Either permanently (by an engineer making changes on the circuit board), or temporarily (by running a program to change the device number).

To change the device number in software: Load and run the 'change address', 'change device' or 'change unit' program (the same program has several different names). You should find this program on the Test/Demo disk supplied with each drive.

All the programs on the Test/Demo disk for the SFD are not suitable for your computer. Because the SFD is supplied for use with several different computers, there are programs on the disk to match all the possible computers. Please do not assume they will all work on your machine. The programs on any two Commodore Test/Demo disks may not be the same if the disks were supplied with different disk drive models.

The SFD requires Double-Sided Double-Density (DSDD) soft sectored 80 track disks. (Quad density should also be acceptable.) The SFD stores half the information on the second side of the disk, so Single-Sided disks that are OK for a 1541 are not OK with an SFD!

Do not buy cheap disks - they will give trouble in the long run. In particular do not buy disks without a hub reinforcement ring. (Super * Corner editors note: I agree that hub rings are a good idea and should be on all disks. All 5 1/4 inch disks seem to work OK in either single or double sided format on my 1541, 1571 and 8250 disk drives. However, others have complained that only quality disks work well in their higher density SFD 1001, 8050 and 8250 drives. My advice is if you can afford the time to experiment then try the cheapo disks and, if they work OK, then use them, if not try, the more expensive specially checked ones. This is a part of the continuing saga of whether there is such a thing as a single sided 5 1/4 inch disk. No person has ever claimed the free case of larger from me for being the first to prove conclusively that a 5 1/4 single sided disk isn't just a relabeled double sided one.)

Like the 8250 the SFD will sense the presence of an 8050 format disk. 8050 data disks are single-sided. This means that you will sometimes have to command the disk drive twice to get it to read an 8050 disk (during the first attempt to obey your command the drive senses an 8050 format and changes mode so that the second time the command works as expected).

Transfer of files to the SFD 1001 is covered in the three sections below.

1) DATABASES. Transfer of database files can only be done by using the Utility program. This program comes on the Superbase System disk. If you are using a Commodore 128 you will need to GO64 to use this program. The program requires a previously prepared (formatted) disk in the destination drive. The program must be run with only one database on the data disk. (Remember that the TRAINING database may be on the disk.) If you have more than one database on the source disk you should make a backup of it, scratch all but one database, and then run Utility on that database.The program will run with only one drive if you wish. If you are using two units remember to change the device number of one of them first!

2) SUPERBASIC PROGRAMS, HELPFILES, etc. (Anything except databases). There are several ways of transferring these files:

a) Use the Utility program

b) Use Unit to Unit (if appropriate for your computer)

c) Use Copy/all, Copy.all, Copy all, or one of the many other variations on the original Jim Butterfield program.

d) Use another disk utility program of a similar type (01copy, shuttle, tcopy, etc.)

All of these programs allow copies of files to be made by using the computer as a temporary storage device between reading the source disk and writing to the destination disk.

3) PROTECTED PROGRAMS. Copying protected programs is illegal. If you damage your program disk, most companies will sell you a new version at cost. Precision Software do. (At the time of writing the price is ten pounds).

THE CBUG LIBRARY

It was a real Christmas for CBUG -- so many authors sent in materials for the Library. Wow, look at it! Twenty Five, yup 25 new disks under a total of 15 new titles. And only one upgrade in the whole lot.

Let's see if our next issue can be even better blessed! Keep the new materials coming in as soon as possible. Please try for a March 15, 1987 closing date.

The entire Bible in a set of 9 disks appropriately deep discounted. What an horrendus project -- copying, naming, linking 1020 separate disk files. That's some truly dedicated work!

Mr. Goceliak has again out done himself. As usual, it is impossible to say which are the most important files on his disk -- but I know from the many phone calls received that his programs for doing backups (vs. copying) between separate drives -- SFD1001 to SFD1001 or many other combinations will be a lifesaver. These are true backup programs, random files and all, not just another copy program.

Say, you can't find that information you knew you read in an Escape two years (maybe) ago? Just haul out Clyde Northrup's index disk.

The shortest disk we, and probably any users' group has ever published is brought to us by Rev Mark Schwarzbauer. A deprotection program for Superoffice. Sometimes the impossible program can only be written by an amateur -- then it becomes exquisitely simple --- sometimes, as Mark points out "with a little help from upstairs." Incidentally several people have commented regarding the royalties on Mark's submissions. I need to mention that those monies do NOT go to Mark, rather the checks made directly to Family Worship Center Church to help reimburse the able assistance of church members and staff who help Mark in his CBUG responsibilities.

Another index, this one of ML Commands along with an elaborate partially complete de-protection program for study purposes only is donated by John Berezinski.

We have three disks of assorted materials, some of which must be of value to nearly every member. And then, four more disks of CP/M 86 materials for those interested in the 8088 coprocessor projects.

HELP -- past contributors. CBUG is going to republish all current library library listings in a single booklet. If your disk was listed without an anotated directory or is in need of additional information for prospective subscribers, please send us a disk CLEARLY LABELED with CBUG number and stock number, your name, phone & zip code, and conspicuously marked as LIB DIR UPDATE. We want complete this project by Summer 1988.

GOCELIAK'S THIRD MINE CBUG #72 NEW RELEASE #11982

from: Anthony Goceliak

Just a quick note to tell you a bit about the organization of this disk. Each section separated by a file titled "   " of type seq is independant of the rest of the disk. For instance if you wish to set up a shift/run-able way to backup your disks from one sfd to another, just format a disk and copy to it all the files beginning with "backup unit x y" and ending with "instructions.bak".

The only exception is the group of files associated with the directory disk suite of programs. By shift/run-ning the existing disk and then selecting the function "need more room!", the program itself will create a new directory disk suite without any of my directories on it completely automatically.

<<This is a particularly notable disk in that it contains a series of backup programs allowing SFD1001 to SFD1001 backup operations as well as backups between most other similar and dis-similar IEEE drives. See the article this issue (Fall 1987). Mr. Goceliak is most certainly living up to his reputation for expertise and highly useful programs for CBUG members! Take a look at the disk indexing program -- what a help to members who have large collections of data or CBUG disks. Those of you who have enjoyed and used Tony's writings and programming ought what to send him a note of thanks. His work is in both quantity and quality beyond belief. As per the note last issue, probably better just to send a note than to phone --his address is in the article section. Thanks again, Tony.>>

```
1 "directory disk  " x7 2c
8    "begin here (c) "  prg  directory disk shift/run function select
7    "need more room!"  prg
8    "add to directory" prg  store, update, and find a file from a
9    "update directory" prg  library of up to 37 of YOUR disks with a
12   "search directory" prg  single fast search!  Includes as a demo
5    "display  headers" prg  all 33 CBUG disks I own
4    "erase  directory" prg
29   "instructions.dir" seq  SS II readable instrns for dir disk suite
1    "&move it"         usr  disk program automatically called when needed
1    "                " seq
1    "-never collect--" seq  warning message due to rnd-file storage
1    "--or validate---" seq  of the up to 37 directories stored
1    "---this disk----" seq
1    "                " seq
1    "escape  articles" seq  these are SS II readable articles
24   "goldware"         seq  my review of the reviewers
20   "b-128 bug!"       seq  crash your b with a one-line request
7    "reversed SS II"   seq  black text on bright background
21   "8050 drive belt"  seq  a failure-mode not checked by cbm diagnostics
9    "m/1 programming"  seq  the hard way
8    "distributedsmart" seq  program the drive with NO computer
9    "default screen"   seq  easy return to power-up screen state
18   "default f-keys"   seq  ditto for the function keys
48   "f-key info"     seq  un-new basic,automatic verify,append basic,etc
18   "restoring f-keys" seq  returning keys to what they were,general case
6    "true reset"       seq  how to power-up your b
18   "seq to prg"       seq  listed seq file to program from basic
1    "                " seq
1    "associated demos" seq  short programs to illustrate the above
5    "function keys²²²" prg  programs all 20 with my layout
4    "f-key defaults"   prg  restores default settings at will
```

```
2    "screen defaults"  prg  reverse, graphics, windows, all gone!
1    "reversed screen"  prg  bringing up SS II with black text on bright
5    "f-restorer demo"  prg  alter, then restore non-standard f-keys
3    "f-key restorer"   prg
2    "basic seq 2 prg"  prg  from basic, leaving program in memory
1    "b bug demo pgm"   prg  ask the right question, and stump your b!
1    "                " seq
1    "ieee unit 2 unit" seq
19   "backup unit x_y"  prg  backup unit 8 (an 8050) to unit 9 (an sfd)!
                             or backup one sfd to another sfd!
21   "2031-8050 backup" prg  backup a 2031 to an 8050
1    "                " seq
5    "8blk revl.obj"   prg distributed code automatically called as needed
32   "instructions.bak" seq  SS II instructions for backup programs
1    "                " seq
1    "all work and no play?
9    "life for std. b"  prg  with no cartridge needed
9    "life-needxtraram" prg  a bit faster if you own a ram cartridge
2    "bank1 life"       prg  m/1 automatically called
2    "m/1 life"         prg  ditto
5    "on screen clock"  prg  or the $1000 plug-in digital watch!
1    "clock $0700 m/1"  prg  m/1 automatically called
2    "crazy screen.g"   prg  a video crazy-patch quilt
2    "crazy screen"     prg  bi-stable quilt
2    "med res chaos"    prg  plot mathematical entropy
1    "                " seq
11   "lethe for sfd/d0" prg  obliterates unwanted files without trace
5    "irq any pg disp"  prg  want to watch your b as it works?
2    "any page m/1 irq" prg  m/1 automatically called as needed
15   "directory+notes"  seq  this file
477 blocks free.
```

◆◆◆◆◆◆◆

THE ESCAPE INDEX v.Sum87 CBUG #73 NEW RELEASE #11997

from: Clyde Northrup

The full text description of this disk is published in this issue (Fall 1987) as an article. Please check the index for Clyde Northrup's article this subject. This disk is a Superbase application containing a complete index to all issues to date of the CBUG escape. There is also a smaller database similar to a magazine index which had been adapted to tracking product advertisements by product name, class and price. There are numerous pre-executed searches located in the help files as well. This is an invaluable aid to those members who regularly reference The ESCAPE for previously published information.

```
1 "sb7 data 1    " s1 2c   1  "h.1541"          seq   1  "h.boot"       seq   1  "h.library"     seq   1  "h.screen"      seq
8  "loader"            prg   1  "h.2040"          seq   1  "h.bounce"     seq   3  "h.line"        seq   5  "h.script"      seq
37 "read me ESCAPE"    seq   37 "ESCAPE read file" seq  1  "h.bracket"    seq   1  "h.link"        seq   1  "h.scroll"      seq
8  "hsuperbaase"       seq   11 "test.p"          seq   1  "h.buffer"     seq   1  "h.list"        seq   1  "h.search"      seq
1  "COMPUTER"          seq   27 "current"         seq   4  "h.bug"        seq   2  "h.load"        seq   1  "h.send"        seq
8  "start.p"           seq   1  "h.2400"          seq   1  "h.bus"        seq   1  "h.loop"        seq   1  "h.sensor"      seq
6  "prices"            seq   1  "h.4040"          seq   2  "h.cable"      seq   1  "h.maketable"   seq   1  "h.seq"         seq
5  "p.r.125column.p"   seq   1  "h.4565"          seq   1  "h.cabs"       seq   1  "h.membership"  seq   1  "h.sid"         seq
6  "escape"            seq   1  "h.6400"          seq   2  "h.calc"       seq   1  "h.meter"       seq   1  "h.signon"      seq
2  "h.64"              seq   1  "h.6432"          seq   1  "h.cartridge"  seq   1  "h.micro"       seq   1  "h.sigs"        seq
8  "hprice"            seq   1  "h.8032"          seq   1  "h.catalog"    seq   1  "h.ml"          seq   1  "h.soft"        seq
14 "p.e.escape.p"      seq   4  "h.8050"          seq   3  "h.cbug"       seq   1  "h.music"       seq   1  "h.sort"        seq
3  "p.r.price.p"       seq   1  "h.8088"          seq   1  "h.channel"    seq   1  "h.new"         seq   2  "h.speed"       seq
6  "menu.p"            seq   1  "h.8250"          seq   1  "h.check"      seq   1  "h.office"      seq   1  "h.spell"       seq
42 "print file"        seq   1  "h.8280"          seq   1  "h.chip"       seq   1  "h.open"        seq   6  "h.ss"          seq
7  "hescape"           seq   1  "h.8432"          seq   1  "h.close"      seq   1  "h.page"        seq   1  "h.stop"        seq
8  "hshopper"          seq   1  "h.9060"          seq   1  "h.clr"        seq   1  "h.pay"         seq   1  "h.submission"  seq
```

```
3  "dir.p"            seq   1  "h.9090"          seq   1  "h.cobol"         seq   1  "h.pet"         seq   1  "h.sys"        seq
8  "hsuperbase"       seq   12 "p.datasearch.p"  seq   1  "h.compatabil"    seq   2  "h.port"        seq   2  "h.tele"       seq
7  "hlater"           seq   1  "h.access"        seq   1  "h.compile"       seq   1  "h.portab"      seq   2  "h.term"       seq
10 "hsearch"          seq   1  "h.account"       seq   2  "h.copy"          seq   1  "h.power"       seq   1  "h.tokenize"   seq
7  "p.r.search.p"     seq   2  "h.address"       seq   13 "h.co-processor"  seq   7  "h.print"       seq   1  "h.trans"      seq
3  "instructions"     prg   1  "h.algorithym"    seq   1  "h.cpm"           seq   3  "h.prog"        seq   1  "h.tutor"      seq
8  "hhelp"            seq   1  "h.align"         seq   2  "h.data"          seq   1  "h.progres"     seq   1  "h.type"       seq
8  "henter"           seq   1  "h.alphabetise"   seq   1  "h.date"          seq   1  "h.protocol"    seq   1  "h.underline"  seq
8  "hindex"           seq   1  "h.amort"         seq   1  "h.dattaset"      seq   1  "h.queue"       seq   1  "h.unscratch"  seq
8  "h125"             seq   1  "h.analog"        seq   2  "h.delphi"        seq   1  "h.rate"        seq   2  "h.use"        seq
8  "hhelps"           seq   1  "h.answer"        seq   2  "h.dir"           seq   1  "h.read"        seq   2  "h.utili"      seq
8  "hdir"             seq   1  "h.ascii"         seq   6  "h.disk"          seq   1  "h.rebuild"     seq   1  "h.var"        seq
1  "hpricelist"       seq   1  "h.assemble"      seq   1  "h.download"      seq   1  "h.record"      seq   1  "h.vidio"      seq
4  "hfind"            seq   1  "h.audio"         seq   5  "h.drive"         seq   1  "h.religion"    seq   1  "h.volt"       seq
1  "hpresort"         seq   1  "h.avatex"        seq   1  "h.dump"          seq   1  "h.rem"         seq   1  "h.wait"       seq
1  "p.p.sort.p"       seq   1  "h.backup"        seq   3  "h.file"          seq   1  "h.rename"      seq   1  "h.warm"       seq
37 "CBUGpublish this" seq   1  "h.bank"          seq   1  "h.fkey"          seq   1  "h.resolution"  seq   1  "h.wild"       seq
7  "hsorth"           seq   4  "h.base"          seq   1  "h.fre("          seq   1  "h.restore"     seq   1  "h.window"     seq
6  "CBUGpublishThis"  seq   2  "h.basic"         seq   1  "h.goto"          seq   1  "h.return"      seq   1  "h.word"       seq
6  "h.128"            seq   1  "h.bbs"           seq   2  "h.graph"         seq   1  "h.rgb"         seq   2  "h.write"      seq
22 "hsearch&report"   seq   1  "h.beeline"       seq   1  "h.hard"          seq   1  "h.roundoff"    seq   1  "h.zener"      seq
7  "hfuture"          seq   1  "h.beginner"      seq   1  "h.initial"       seq   1  "h.run"         seq   1  "h.zip"        seq
8  "hkey"             seq   1  "h.bible"         seq   1  "h.input"         seq   1  "h.rvs"         seq   789 blocks free.
1  "h.1001"           seq   1  "h.binary"        seq   1  "h.insert"        seq   2  "h.save"        seq
1  "h.1200"           seq   1  "h.bios"          seq   2  "h.interface"     seq   1  "h.sb"          seq
1  "h.1345"           seq   1  "h.bmon"          seq   1  "h.label"         seq   1  "h.scratch"     seq
```

◆◆◆◆◆◆◆

SUPEROFFICE SCRUBBER   **CBUG #74**   **NEW RELEASE**   **#12007**

from: Mark Schwarzbauer

CBUGS SHORTEST DISK EVER! In just four lines of code is built a complete SUPEROFFICE DEPROTECTION/DECRYPTION program. Easy to follow instructions are included in a "superscript" file making this program very simple to use. When completed in less than 5 minutes you will have a deprotected version of SUPEROFFICE that will load in only 53 seconds instead of the usual 2 minutes. This version will allow you to change function keys in Superscript, (an article to be published in the escape) and also add on new features yet to be developed like an integrated SUPEROFFICE with BTERM! The new DEPROTECTED SUPEROFFICE just shift/runs like the original.

```
1 "so decryptor    " 02 2c      1  "deprotector 3"  prg      2  "soloader"  prg      12  "instructions"  seq
1    "initializer"   prg        1  "deprotector 2"  prg      2  "start.p"   seq      2033 blocks free.
```

◆◆◆◆◆◆◆

BEREZINSKI MACHINE LANGUAGE INDEX DATA BASE   **CBUG #75**   **NEW RELEASE**   **#12012**

from: John Berezinski

This disk contains a database of the ml commands for the b128. It can be used to sort commands by name, by registers affected, by flags affected, or even by the bit pattern of the command.
 The name of the data base is mldb.
 The name of one file is mlcommands. It uses command names for key.
 The name of other file is binary. It uses the binary number of the command for the key.
Included on the disk are some help files on command types.

```
1 "ml cmds dB     " md 2c     1  "himplied"    seq     4  "hindirectindex"  seq     2    "intro.basic.ss"  seq
3    "intro.basic"   prg      2  "hindex"      seq     2  "hindexindirect"  seq     6    "start.p"         seq
2    "mlcommands"    seq      1  "hzeropage"   seq     2  "hindirection"    seq     1    "MLDB"            seq
111  "mlcmds"        seq      2  "habsolute"   seq     1  "hlist"           seq     1601 blocks free.
2    "himmediate"    seq      2  "binary"      seq
```

One of the more profound needs expressed by a number of CBUG members is the inability to get into some of the commercial software to modify it for specialized or more advanced applications. A notable example is Superbase which would be greatly improved if it had more than one key-name permitted; i.e. using for example last name, zip and invoice numbers as selectable key names. To do this we have to get into the program which may be a harder feat than altering the compiled machine language program itself. John set about working on this project some time ago from the vantage point of the SuperOffice program. Following is a partially completed effort in that direction which along with its bank switching and trading routines sheds some light on undocumented capabilities of the B128. The basic premiss of operation is to load the program, move it to a spare bank, reset vectors and resave. We've included this interesting material as study information ONLY. No representation is made as to its workability, reliability or possible bugs or bug generation. There are known shortcomings in this unfinished project.

```
1 "twozero      " 20 2c     1  "s2"        prg     2  "soloader"  prg     4    "loader"      prg
2    "inst."     prg        1  "s3"        prg     1  "mlloader"  prg     34   "&&instruct"  seq
1    "s1"        prg        3  "ssloader"  prg     2  "start.p"   seq     2001 blocks free.
```

This series of nine disks contains the text of the King James Bible, both Old and New Testaments. The original copying to disk was done by Randall J. Bernard, of Morenci, Arizona, for the Commodore 64 and distributed free of charge by him. The 1541 Speedscript (prg) files were then converted to 8050 Superscript (seq) files by Ed Rhyner of CBUG. Marilyn Gardner of CBUG did final editing and added link specifications.

This set of disks is offered with no royalty charge. Since we did not pay for the original, we should not charge extra for the copies.

Each book of the Bible is a separate sequential file, generally named by the first three letters of the book name (using no capital letters), followed by a space, then the chapter number. Exceptions are phm for Philemon (to avoid confusion with Philippians), jdg for Judges (to avoid confusion with Jude), aand rth for Ruth. Books with a numeric prefix use an arabic rather than Roman numeral and have no space between the number and the letters of the book name, such as 1ti for 1 Timothy.

These disks are useful for a number of purposes, including: searching for words or phrases with the Superscript search feature; printing out a series of references for a Bible study or sermon; and, if you can't stay away from your computer, you can still have personal devotions with the Scriptures in your machine!

We have tried to check thoroughly to make sure the copies are accurate. If any errors are noted, please contact Marilyn Gardner, 1630 Madison Street, Evanston, Illinois 60202.

<<Think about it, friends, going thru each of these files, converting them to 80 column, naming and linking each file!!>>

<<Order the entire set for $40.00 using stock #12026, or one disk at a time for the standard $9.00 library fee using the .X suffixed stock numbers.>>

Genesis 1 - Leviticus 23 Stock #12026.1

1 "old testament" o1	24 "gen 19" seq	18 "gen 38" seq	16 "exo 7" seq	22 "exo 26" seq	16 "lev 5" seq
18 "gen 1" seq	11 "gen 20" seq	14 "gen 39" seq	21 "exo 8" seq	13 "exo 27" seq	20 "lev 6" seq
14 "gen 2" seq	17 "gen 21" seq	13 "gen 40" seq	21 "exo 9" seq	28 "exo 28" seq	24 "lev 7" seq
15 "gen 3" seq	14 "gen 22" seq	31 "gen 41" seq	20 "exo 10" seq	30 "exo 29" seq	22 "lev 8" seq
14 "gen 4" seq	12 "gen 23" seq	22 "gen 42" seq	8 "exo 11" seq	23 "exo 30" seq	14 "lev 9" seq
12 "gen 5" seq	39 "gen 24" seq	21 "gen 43" seq	33 "exo 12" seq	10 "exo 31" seq	14 "lev 10" seq
13 "gen 6" seq	17 "gen 25" seq	19 "gen 44" seq	15 "exo 13" seq	24 "exo 32" seq	26 "lev 11" seq
13 "gen 7" seq	20 "gen 26" seq	16 "gen 45" seq	21 "exo 14" seq	15 "exo 33" seq	6 "lev 12" seq
13 "gen 8" seq	27 "gen 27" seq	18 "gen 46" seq	16 "exo 15" seq	23 "exo 34" seq	39 "lev 13" seq
15 "gen 9" seq	14 "gen 28" seq	21 "gen 47" seq	23 "exo 16" seq	19 "exo 35" seq	36 "lev 14" seq
13 "gen 10" seq	19 "gen 29" seq	14 "gen 48" seq	11 "exo 17" seq	21 "exo 36" seq	21 "lev 15" seq
14 "gen 11" seq	23 "gen 30" seq	18 "gen 49" seq	17 "exo 18" seq	17 "exo 37" seq	25 "lev 16" seq
12 "gen 12" seq	32 "gen 31" seq	16 "gen 50" seq	16 "exo 19" seq	20 "exo 38" seq	12 "lev 17" seq
10 "gen 13" seq	18 "gen 32" seq	11 "exo 1" seq	13 "exo 20" seq	24 "exo 39" seq	16 "lev 18" seq
14 "gen 14" seq	12 "gen 33" seq	14 "exo 2" seq	19 "exo 21" seq	19 "exo 40" seq	20 "lev 19" seq
11 "gen 15" seq	18 "gen 34" seq	17 "exo 3" seq	17 "exo 22" seq	12 "lev 1" seq	19 "lev 20" seq
10 "gen 16" seq	15 "gen 35" seq	19 "exo 4" seq	19 "exo 23" seq	11 "lev 2" seq	13 "lev 21" seq
16 "gen 17" seq	20 "gen 36" seq	14 "exo 5" seq	12 "exo 24" seq	11 "lev 3" seq	20 "lev 22" seq
19 "gen 18" seq	21 "gen 37" seq	17 "exo 6" seq	21 "exo 25" seq	25 "lev 4" seq	27 "lev 23" seq
					4 bcks free.

Leviticus 24 - Judges 6 stock #12026.2

1 "old testament" o2	27 "num 14" seq	24 "num 32" seq	16 "deu 14" seq	29 "deu 32" seq	6 "jos 16" seq
13 "lev 24" seq	24 "num 15" seq	24 "num 33" seq	16 "deu 15" seq	18 "deu 33" seq	15 "jos 17" seq
33 "lev 25" seq	30 "num 16" seq	15 "num 34" seq	16 "deu 16" seq	7 "deu 34" seq	18 "jos 18" seq
28 "lev 26" seq	8 "num 17" seq	22 "num 35" seq	16 "deu 17" seq	13 "jos 1" seq	24 "jos 19" seq
21 "lev 27" seq	25 "num 18" seq	11 "num 36" seq	14 "deu 18" seq	17 "jos 2" seq	7 "jos 20" seq
31 "num 1" seq	16 "num 19" seq	27 "deu 1" seq	14 "deu 19" seq	13 "jos 3" seq	24 "jos 21" seq
20 "num 2" seq	18 "num 20" seq	23 "deu 2" seq	14 "deu 20" seq	16 "jos 4" seq	29 "jos 22" seq
30 "num 3" seq	21 "num 21" seq	18 "deu 3" seq	16 "deu 21" seq	12 "jos 5" seq	13 "jos 23" seq
33 "num 4" seq	26 "num 22" seq	34 "deu 4" seq	19 "deu 22" seq	20 "jos 6" seq	23 "jos 24" seq
20 "num 5" seq	17 "num 23" seq	20 "deu 5" seq	16 "deu 23" seq	21 "jos 7" seq	22 "jdg 1" seq
17 "num 6" seq	15 "num 24" seq	14 "deu 6" seq	15 "deu 24" seq	26 "jos 8" seq	16 "jdg 2" seq
45 "num 7" seq	11 "num 25" seq	19 "deu 7" seq	13 "deu 25" seq	18 "jos 9" seq	19 "jdg 3" seq
16 "num 8" seq	35 "num 26" seq	13 "deu 8" seq	15 "deu 26" seq	31 "jos 10" seq	17 "jdg 4" seq
16 "num 9" seq	14 "num 27" seq	21 "deu 9" seq	14 "deu 27" seq	16 "jos 11" seq	18 "jdg 5" seq
20 "num 10" seq	18 "num 28" seq	14 "deu 10" seq	46 "deu 28" seq	11 "jos 12" seq	28 "jdg 6" seq
23 "num 11" seq	23 "num 29" seq	21 "deu 11" seq	19 "deu 29" seq	20 "jos 13" seq	14 blocks free.
9 "num 12" seq	11 "num 30" seq	23 "deu 12" seq	15 "deu 30" seq	11 "jos 14" seq	
17 "num 13" seq	29 "num 31" seq	14 "deu 13" seq	23 "deu 31" seq	26 "jos 15" seq	

Judges 7 - 2 Kings 5 stock #12026.3

1 "old testament " o3	17 "rth 2" seq	25 "1sa 15" seq	16 "2sa 1" seq	24 "2sa 18" seq	28 "1ki 11" seq
21 "jdg 7" seq	18 "rth 3" seq	15 "1sa 16" seq	21 "2sa 2" seq	31 "2sa 19" seq	23 "1ki 12" seq
22 "jdg 8" seq	14 "rth 4" seq	38 "1sa 17" seq	25 "2sa 3" seq	18 "2sa 20" seq	24 "1ki 13" seq
38 "jdg 9" seq	17 "1sa 1" seq	19 "1sa 18" seq	10 "2sa 4" seq	17 "2sa 21" seq	22 "1ki 14" seq
11 "jdg 10" seq	25 "1sa 2" seq	16 "1sa 19" seq	15 "2sa 5" seq	22 "2sa 22" seq	21 "1ki 15" seq
28 "jdg 11" seq	12 "1sa 3" seq	29 "1sa 20" seq	16 "2sa 6" seq	20 "2sa 23" seq	22 "1ki 16" seq
10 "jdg 12" seq	19 "1sa 4" seq	11 "1sa 21" seq	19 "2sa 7" seq	19 "2sa 24" seq	15 "1ki 17" seq
16 "jdg 13" seq	10 "1sa 5" seq	17 "1sa 22" seq	11 "2sa 8" seq	33 "1ki 1" seq	30 "1ki 18" seq
15 "jdg 14" seq	17 "1sa 6" seq	19 "1sa 23" seq	9 "2sa 9" seq	33 "1ki 2" seq	16 "1ki 19" seq
14 "jdg 15" seq	12 "1sa 7" seq	15 "1sa 24" seq	14 "2sa 10" seq	19 "1ki 3" seq	32 "1ki 20" seq
21 "jdg 16" seq	12 "1sa 8" seq	31 "1sa 25" seq	18 "2sa 11" seq	17 "1ki 4" seq	20 "1ki 21" seq
9 "jdg 17" seq	21 "1sa 9" seq	19 "1sa 26" seq	22 "2sa 12" seq	12 "1ki 5" seq	33 "1ki 22" seq
22 "jdg 18" seq	19 "1sa 10" seq	9 "1sa 27" seq	25 "2sa 13" seq	23 "1ki 6" seq	14 "2ki 1" seq

```
24 "jdg 19"   seq    20 "1sa 11"   seq    30 "1sa 28"   seq    24 "2sa 14"   seq    33 "1ki 7"    seq    18 "2ki 2"    seq
33 "jdg 20"   seq    17 "1sa 12"   seq    10 "1sa 29"   seq    25 "2sa 15"   seq    48 "1ki 8"    seq    19 "2ki 3"    seq
17 "jdg 21"   seq    16 "1sa 13"   seq    21 "1sa 30"   seq    16 "2sa 16"   seq    19 "1ki 9"    seq    28 "2ki 4"    seq
14 "rth 1"    seq    36 "1sa 14"   seq    8  "1sa 31"   seq    21 "2sa 17"   seq    19 "1ki 10"   seq    21 "2ki 5"    seq
                                                                                                       0 blocks free.
```

2 Kings 6 - Esther 9   Stock #12026.4

```
1 "old testament" o4  22 "2ki 25"  seq    7  "1ch 20"   seq    12 "2ch 11"   seq    18 "2ch 31"   seq    14 "neh 5"    seq
22 "2ki 6"    seq    18 "1ch 1"    seq    20 "1ch 21"   seq    12 "2ch 12"   seq    25 "2ch 32"   seq    13 "neh 6"    seq
17 "2ki 7"    seq    21 "1ch 2"    seq    14 "1ch 22"   seq    15 "2ch 13"   seq    17 "2ch 33"   seq    30 "neh 7"    seq
21 "2ki 8"    seq    9  "1ch 3"    seq    16 "1ch 23"   seq    10 "2ch 14"   seq    27 "2ch 34"   seq    16 "neh 8"    seq
26 "2ki 9"    seq    21 "1ch 4"    seq    14 "1ch 24"   seq    12 "2ch 15"   seq    21 "2ch 35"   seq    31 "neh 9"    seq
26 "2ki 10"   seq    16 "1ch 5"    seq    14 "1ch 25"   seq    11 "2ch 16"   seq    17 "2ch 36"   seq    16 "neh 10"   seq
16 "2ki 11"   seq    32 "1ch 6"    seq    18 "1ch 26"   seq    11 "2ch 17"   seq    9  "ezr 1"    seq    19 "neh 11"   seq
15 "2ki 12"   seq    20 "1ch 7"    seq    19 "1ch 27"   seq    23 "2ch 18"   seq    26 "ezr 2"    seq    24 "neh 12"   seq
17 "2ki 13"   seq    13 "1ch 8"    seq    19 "1ch 28"   seq    8  "2ch 19"   seq    13 "ezr 3"    seq    22 "neh 13"   seq
20 "2ki 14"   seq    23 "1ch 9"    seq    22 "1ch 29"   seq    26 "2ch 20"   seq    18 "ezr 4"    seq    17 "est 1"    seq
24 "2ki 15"   seq    9  "1ch 10"   seq    12 "2ch 1"    seq    14 "2ch 21"   seq    13 "ezr 5"    seq    19 "est 2"    seq
15 "2ki 16"   seq    23 "1ch 11"   seq    14 "2ch 2"    seq    10 "2ch 22"   seq    18 "ezr 6"    seq    13 "est 3"    seq
28 "2ki 17"   seq    23 "1ch 12"   seq    11 "2ch 3"    seq    17 "2ch 23"   seq    19 "ezr 7"    seq    12 "est 4"    seq
27 "2ki 18"   seq    9  "1ch 13"   seq    14 "2ch 4"    seq    21 "2ch 24"   seq    22 "ezr 8"    seq    11 "est 5"    seq
26 "2ki 19"   seq    9  "1ch 14"   seq    11 "2ch 5"    seq    22 "2ch 25"   seq    14 "ezr 9"    seq    11 "est 6"    seq
15 "2ki 20"   seq    17 "1ch 15"   seq    31 "2ch 6"    seq    17 "2ch 26"   seq    23 "ezr 10"   seq    8  "est 7"    seq
17 "2ki 21"   seq    20 "1ch 16"   seq    17 "2ch 7"    seq    6  "2ch 27"   seq    9  "neh 1"    seq    15 "est 8"    seq
16 "2ki 22"   seq    17 "1ch 17"   seq    13 "2ch 8"    seq    20 "2ch 28"   seq    15 "neh 2"    seq    23 "est 9"    seq
32 "2ki 23"   seq    10 "1ch 18"   seq    20 "2ch 9"    seq    27 "2ch 29"   seq    21 "neh 3"    seq    2 blocks free.
14 "2ki 24"   seq    15 "1ch 19"   seq    13 "2ch 10"   seq    21 "2ch 30"   seq    16 "neh 4"    seq
```

Esther 10 - Proverbs 31   Stock #12026.5

```
1 "old testament"o5  10 "job 37"   seq    8  "psa 33"   seq    11 "psa 71"   seq    13 "psa 109"  seq    8  "psa 147"  seq
3  "est 10"   seq    16 "job 38"   seq    8  "psa 34"   seq    9  "psa 72"   seq    4  "psa 110"  seq    5  "psa 148"  seq
14 "job 1"    seq    12 "job 39"   seq    13 "psa 35"   seq    10 "psa 73"   seq    5  "psa 111"  seq    4  "psa 149"  seq
9  "job 2"    seq    9  "job 40"   seq    6  "psa 36"   seq    10 "psa 74"   seq    5  "psa 112"  seq    3  "psa 150"  seq
10 "job 3"    seq    12 "job 41"   seq    17 "psa 37"   seq    5  "psa 75"   seq    4  "psa 113"  seq    13 "pro 1"    seq
8  "job 4"    seq    10 "job 42"   seq    9  "psa 38"   seq    5  "psa 76"   seq    3  "psa 114"  seq    8  "pro 2"    seq
11 "job 5"    seq    3  "psa 1"    seq    6  "psa 39"   seq    8  "psa 77"   seq    7  "psa 115"  seq    13 "pro 3"    seq
11 "job 6"    seq    5  "psa 2"    seq    9  "psa 40"   seq    29 "psa 78"   seq    7  "psa 116"  seq    10 "pro 4"    seq
9  "job 7"    seq    4  "psa 3"    seq    6  "psa 41"   seq    7  "psa 79"   seq    1  "psa 117"  seq    9  "pro 5"    seq
8  "job 8"    seq    4  "psa 4"    seq    6  "psa 42"   seq    8  "psa 80"   seq    11 "psa 118"  seq    13 "pro 6"    seq
13 "job 9"    seq    6  "psa 5"    seq    3  "psa 43"   seq    7  "psa 81"   seq    60 "psa 119"  seq    10 "pro 7"    seq
10 "job 10"   seq    4  "psa 6"    seq    11 "psa 44"   seq    3  "psa 82"   seq    3  "psa 120"  seq    14 "pro 8"    seq
8  "job 11"   seq    8  "psa 7"    seq    8  "psa 45"   seq    7  "psa 83"   seq    3  "psa 121"  seq    7  "pro 9"    seq
10 "job 12"   seq    4  "psa 8"    seq    5  "psa 46"   seq    5  "psa 84"   seq    3  "psa 122"  seq    13 "pro 10"   seq
10 "job 13"   seq    9  "psa 9"    seq    4  "psa 47"   seq    5  "psa 85"   seq    3  "psa 123"  seq    13 "pro 11"   seq
10 "job 14"   seq    8  "psa 10"   seq    6  "psa 48"   seq    7  "psa 86"   seq    3  "psa 124"  seq    12 "pro 12"   seq
14 "job 15"   seq    4  "psa 11"   seq    8  "psa 49"   seq    3  "psa 87"   seq    3  "psa 125"  seq    10 "pro 13"   seq
9  "job 16"   seq    4  "psa 12"   seq    10 "psa 50"   seq    7  "psa 88"   seq    3  "psa 126"  seq    14 "pro 14"   seq
6  "job 17"   seq    3  "psa 13"   seq    8  "psa 51"   seq    21 "psa 89"   seq    3  "psa 127"  seq    13 "pro 15"   seq
8  "job 18"   seq    4  "psa 14"   seq    4  "psa 52"   seq    8  "psa 90"   seq    3  "psa 128"  seq    13 "pro 16"   seq
11 "job 19"   seq    3  "psa 15"   seq    4  "psa 53"   seq    7  "psa 91"   seq    3  "psa 129"  seq    12 "pro 17"   seq
12 "job 20"   seq    5  "psa 16"   seq    3  "psa 54"   seq    6  "psa 92"   seq    3  "psa 130"  seq    10 "pro 18"   seq
13 "job 21"   seq    7  "psa 17"   seq    10 "psa 55"   seq    3  "psa 93"   seq    2  "psa 131"  seq    12 "pro 19"   seq
12 "job 22"   seq    21 "psa 18"   seq    5  "psa 56"   seq    9  "psa 94"   seq    7  "psa 132"  seq    12 "pro 20"   seq
7  "job 23"   seq    7  "psa 19"   seq    6  "psa 57"   seq    5  "psa 95"   seq    2  "psa 133"  seq    12 "pro 21"   seq
11 "job 24"   seq    4  "psa 20"   seq    5  "psa 58"   seq    6  "psa 96"   seq    2  "psa 134"  seq    12 "pro 22"   seq
2  "job 25"   seq    6  "psa 21"   seq    8  "psa 59"   seq    5  "psa 97"   seq    8  "psa 135"  seq    13 "pro 23"   seq
5  "job 26"   seq    13 "psa 22"   seq    5  "psa 60"   seq    4  "psa 98"   seq    9  "psa 136"  seq    14 "pro 24"   seq
9  "job 27"   seq    3  "psa 23"   seq    3  "psa 61"   seq    4  "psa 99"   seq    4  "psa 137"  seq    12 "pro 25"   seq
11 "job 28"   seq    5  "psa 24"   seq    6  "psa 62"   seq    3  "psa 100"  seq    4  "psa 138"  seq    11 "pro 26"   seq
9  "job 29"   seq    8  "psa 25"   seq    5  "psa 63"   seq    4  "psa 101"  seq    10 "psa 139"  seq    11 "pro 27"   seq
12 "job 30"   seq    5  "psa 26"   seq    5  "psa 64"   seq    11 "psa 102"  seq    6  "psa 140"  seq    13 "pro 28"   seq
16 "job 31"   seq    8  "psa 27"   seq    7  "psa 65"   seq    9  "psa 103"  seq    5  "psa 141"  seq    11 "pro 29"   seq
9  "job 32"   seq    5  "psa 28"   seq    8  "psa 66"   seq    14 "psa 104"  seq    4  "psa 142"  seq    15 "pro 30"   seq
13 "job 33"   seq    5  "psa 29"   seq    3  "psa 67"   seq    16 "psa 105"  seq    6  "psa 143"  seq    12 "pro 31"   seq
15 "job 34"   seq    5  "psa 30"   seq    17 "psa 68"   seq    19 "psa 106"  seq    7  "psa 144"  seq    183 blocks free.
7  "job 35"   seq    11 "psa 31"   seq    16 "psa 69"   seq    17 "psa 107"  seq    8  "psa 145"  seq
12 "job 36"   seq    6  "psa 32"   seq    3  "psa 70"   seq    5  "psa 108"  seq    5  "psa 146"  seq
```

Ecclesiastes 1 - Lamentations 2   Stock #12026.6

```
1 "old testament o6  5  "isa 4"    seq    19 "isa 28"   seq    10 "isa 52"   seq    15 "jer 10"   seq    19 "jer 34"   seq
9  "ecc 1"    seq    20 "isa 5"    seq    17 "isa 29"   seq    9  "isa 53"   seq    17 "jer 11"   seq    15 "jer 35"   seq
16 "ecc 2"    seq    8  "isa 6"    seq    23 "isa 30"   seq    12 "isa 54"   seq    13 "jer 12"   seq    24 "jer 36"   seq
11 "ecc 3"    seq    15 "isa 7"    seq    7  "isa 31"   seq    9  "isa 55"   seq    17 "jer 13"   seq    14 "jer 37"   seq
9  "ecc 4"    seq    13 "isa 8"    seq    11 "isa 32"   seq    9  "isa 56"   seq    16 "jer 14"   seq    22 "jer 38"   seq
12 "ecc 5"    seq    14 "isa 9"    seq    14 "isa 33"   seq    13 "isa 57"   seq    15 "jer 15"   seq    13 "jer 39"   seq
7  "ecc 6"    seq    21 "isa 10"   seq    12 "isa 34"   seq    12 "isa 58"   seq    16 "jer 16"   seq    15 "jer 40"   seq
14 "ecc 7"    seq    12 "isa 11"   seq    7  "isa 35"   seq    14 "isa 59"   seq    19 "jer 17"   seq    15 "jer 41"   seq
11 "ecc 8"    seq    3  "isa 12"   seq    15 "isa 36"   seq    15 "isa 60"   seq    15 "jer 18"   seq    16 "jer 42"   seq
12 "ecc 9"    seq    13 "isa 13"   seq    25 "isa 37"   seq    9  "isa 61"   seq    12 "jer 19"   seq    10 "jer 43"   seq
10 "ecc 10"   seq    19 "isa 14"   seq    13 "isa 38"   seq    8  "isa 62"   seq    13 "jer 20"   seq    27 "jer 44"   seq
6  "ecc 11"   seq    6  "isa 15"   seq    6  "isa 39"   seq    13 "isa 63"   seq    11 "jer 21"   seq    4  "jer 45"   seq
9  "ecc 12"   seq    10 "isa 16"   seq    18 "isa 40"   seq    8  "isa 64"   seq    19 "jer 22"   seq    19 "jer 46"   seq
8  "son 1"    seq    10 "isa 17"   seq    18 "isa 41"   seq    17 "isa 65"   seq    27 "jer 23"   seq    5  "jer 47"   seq
```

```
8   "son 2"     seq   6   "isa 18"    seq   15  "isa 42"    seq   18  "isa 66"    seq   8   "jer 24"    seq   27  "jer 48"    seq
7   "son 3"     seq   16  "isa 19"    seq   16  "isa 43"    seq   12  "jer 1"     seq   26  "jer 25"    seq   27  "jer 49"    seq
9   "son 4"     seq   4   "isa 20"    seq   20  "isa 44"    seq   23  "jer 2"     seq   18  "jer 26"    seq   33  "jer 50"    seq
9   "son 5"     seq   10  "isa 21"    seq   17  "isa 45"    seq   18  "jer 3"     seq   17  "jer 27"    seq   42  "jer 51"    seq
7   "son 6"     seq   15  "isa 22"    seq   8   "isa 46"    seq   20  "jer 4"     seq   13  "jer 28"    seq   24  "jer 52"    seq
7   "son 7"     seq   11  "isa 23"    seq   11  "isa 47"    seq   20  "jer 5"     seq   23  "jer 29"    seq   18  "lam 1"     seq
8   "son 8"     seq   14  "isa 24"    seq   14  "isa 48"    seq   19  "jer 6"     seq   16  "jer 30"    seq   19  "lam 2"     seq
18  "isa 1"     seq   8   "isa 25"    seq   19  "isa 49"    seq   21  "jer 7"     seq   29  "jer 31"    seq   16 blocks free.
13  "isa 2"     seq   13  "isa 26"    seq   8   "isa 50"    seq   16  "jer 8"     seq   32  "jer 32"    seq
14  "isa 3"     seq   9   "isa 27"    seq   17  "isa 51"    seq   18  "jer 9"     seq   19  "jer 33"    seq
```

Lamentations 3 - Matthew 10   Stock #12026.7

```
1 "old testament"o7    21  "eze 21"    seq   20  "eze 45"    seq   11  "hos 9"     seq   9   "mic 2"     seq   15  "zec 8"     seq
21  "lam 3"     seq   19  "eze 22"    seq   19  "eze 46"    seq   10  "hos 10"    seq   8   "mic 3"     seq   12  "zec 9"     seq
14  "lam 4"     seq   31  "eze 23"    seq   18  "eze 47"    seq   7   "hos 11"    seq   10  "mic 4"     seq   9   "zec 10"    seq
7   "lam 5"     seq   17  "eze 24"    seq   24  "eze 48"    seq   8   "hos 12"    seq   10  "mic 5"     seq   11  "zec 11"    seq
19  "eze 1"     seq   12  "eze 25"    seq   14  "dan 1"     seq   10  "hos 13"    seq   10  "mic 6"     seq   11  "zec 12"    seq
7   "eze 2"     seq   16  "eze 26"    seq   34  "dan 2"     seq   6   "hos 14"    seq   13  "mic 7"     seq   8   "zec 13"    seq
18  "eze 3"     seq   21  "eze 27"    seq   24  "dan 3"     seq   12  "joe 1"     seq   9   "nah 1"     seq   17  "zec 14"    seq
12  "eze 4"     seq   18  "eze 28"    seq   29  "dan 4"     seq   21  "joe 2"     seq   9   "nah 2"     seq   11  "mal 1"     seq
14  "eze 5"     seq   16  "eze 29"    seq   23  "dan 5"     seq   13  "joe 3"     seq   12  "nah 3"     seq   12  "mal 2"     seq
11  "eze 6"     seq   17  "eze 30"    seq   21  "dan 6"     seq   11  "amo 1"     seq   10  "hab 1"     seq   13  "mal 3"     seq
17  "eze 7"     seq   15  "eze 31"    seq   21  "dan 7"     seq   10  "amo 2"     seq   13  "hab 2"     seq   4   "mal 4"     seq
14  "eze 8"     seq   24  "eze 32"    seq   19  "dan 8"     seq   9   "amo 3"     seq   12  "hab 3"     seq   12  "mat 1"     seq
9   "eze 9"     seq   23  "eze 33"    seq   22  "dan 9"     seq   10  "amo 4"     seq   12  "zep 1"     seq   14  "mat 2"     seq
15  "eze 10"    seq   21  "eze 34"    seq   14  "dan 10"    seq   15  "amo 5"     seq   11  "zep 2"     seq   9   "mat 3"     seq
16  "eze 11"    seq   9   "eze 35"    seq   34  "dan 11"    seq   9   "amo 6"     seq   13  "zep 3"     seq   13  "mat 4"     seq
18  "eze 12"    seq   26  "eze 36"    seq   9   "dan 12"    seq   11  "amo 7"     seq   10  "hag 1"     seq   24  "mat 5"     seq
17  "eze 13"    seq   19  "eze 37"    seq   8   "hos 1"     seq   9   "amo 8"     seq   15  "hag 2"     seq   18  "mat 6"     seq
17  "eze 14"    seq   17  "eze 38"    seq   15  "hos 2"     seq   12  "amo 9"     seq   14  "zec 1"     seq   14  "mat 7"     seq
5   "eze 15"    seq   20  "eze 39"    seq   4   "hos 3"     seq   15  "oba 1"     seq   7   "zec 2"     seq   18  "mat 8"     seq
42  "eze 16"    seq   34  "eze 40"    seq   11  "hos 4"     seq   11  "jon 1"     seq   7   "zec 3"     seq   19  "mat 9"     seq
17  "eze 17"    seq   18  "eze 41"    seq   9   "hos 5"     seq   5   "jon 2"     seq   8   "zec 4"     seq   21  "mat 10"    seq
21  "eze 18"    seq   14  "eze 42"    seq   6   "hos 6"     seq   6   "jon 3"     seq   8   "zec 5"     seq    9  blocks free
8   "eze 19"    seq   20  "eze 43"    seq   10  "hos 7"     seq   7   "jon 4"     seq   10  "zec 6"     seq
35  "eze 20"    seq   23  "eze 44"    seq   8   "hos 8"     seq   10  "mic 1"     seq   9   "zec 7"     seq
```

Mathew 11 - Acts 15   Stock #12026.8

```
1 "new testament"o8    36  "mat 26"    seq   35  "mar 14"    seq   18  "luk 14"    seq   33  "joh 6"     seq   16  "act 1"     seq
15  "mat 11"    seq   32  "mat 27"    seq   22  "mar 15"    seq   16  "luk 15"    seq   23  "joh 7"     seq   24  "act 2"     seq
27  "mat 12"    seq   10  "mat 28"    seq   11  "mar 16"    seq   17  "luk 16"    seq   28  "joh 8"     seq   14  "act 3"     seq
31  "mat 13"    seq   22  "mar 1"     seq   36  "luk 1"     seq   18  "luk 17"    seq   19  "joh 9"     seq   20  "act 4"     seq
17  "mat 14"    seq   16  "mar 2"     seq   25  "luk 2"     seq   20  "luk 18"    seq   18  "joh 10"    seq   23  "act 5"     seq
19  "mat 15"    seq   16  "mar 3"     seq   22  "luk 3"     seq   24  "luk 19"    seq   27  "joh 11"    seq   9   "act 6"     seq
16  "mat 16"    seq   20  "mar 4"     seq   23  "luk 4"     seq   22  "luk 20"    seq   24  "joh 12"    seq   32  "act 7"     seq
14  "mat 17"    seq   21  "mar 5"     seq   21  "luk 5"     seq   19  "luk 21"    seq   18  "joh 13"    seq   21  "act 8"     seq
19  "mat 18"    seq   30  "mar 6"     seq   27  "luk 6"     seq   32  "luk 22"    seq   16  "joh 14"    seq   24  "act 9"     seq
17  "mat 19"    seq   19  "mar 7"     seq   27  "luk 7"     seq   26  "luk 23"    seq   13  "joh 15"    seq   25  "act 10"    seq
18  "mat 20"    seq   19  "mar 8"     seq   32  "luk 8"     seq   24  "luk 24"    seq   17  "joh 16"    seq   16  "act 11"    seq
26  "mat 21"    seq   26  "mar 9"     seq   32  "luk 9"     seq   23  "joh 1"     sec   14  "joh 17"    seq   15  "act 12"    seq
20  "mat 22"    seq   27  "mar 10"    seq   23  "luk 10"    seq   12  "joh 2"     seq   22  "joh 18"    seq   29  "act 13"    seq
20  "mat 23"    seq   18  "mar 11"    seq   30  "luk 11"    seq   18  "joh 3"     seq   23  "joh 19"    seq   15  "act 14"    seq
24  "mat 24"    seq   24  "mar 12"    seq   31  "luk 12"    seq   25  "joh 4"     seq   17  "joh 20"    seq   22  "act 15"    seq
23  "mat 25"    seq   19  "mar 13"    seq   20  "luk 13"    seq   22  "joh 5"     seq   16  "joh 21"    seq   5 blocks free.
```

Acts 16 - Revelations 22   Stock #12026.9

```
1 "new testament"o9    12  "rom 14"    seq   9   "2co 9"     seq   7   "1th 3"     seq   9   "heb 8"     seq   13  "rev 1"     seq
22  "act 16"    seq   17  "rom 15"    seq   11  "2co 10"    seq   9   "1th 4"     seq   16  "heb 9"     seq   18  "rev 2"     seq
20  "act 17"    seq   13  "rom 16"    seq   17  "2co 11"    seq   10  "1th 5"     seq   19  "heb 10"    seq   14  "rev 3"     seq
16  "act 18"    seq   15  "1co 1"     seq   13  "2co 12"    seq   7   "2th 1"     seq   22  "heb 11"    seq   8   "rev 4"     seq
23  "act 19"    seq   9   "1co 2"     seq   7   "2co 13"    seq   9   "2th 2"     seq   17  "heb 12"    seq   10  "rev 5"     seq
20  "act 20"    seq   11  "1co 3"     seq   11  "gal 1"     seq   8   "2th 3"     seq   12  "heb 13"    seq   12  "rev 6"     seq
25  "act 21"    seq   11  "1co 4"     seq   13  "gal 2"     seq   11  "1ti 1"     seq   13  "jam 1"     seq   12  "rev 7"     seq
17  "act 22"    seq   8   "1co 5"     seq   15  "gal 3"     seq   6   "1ti 2"     seq   12  "jam 2"     seq   9   "rev 8"     seq
21  "act 23"    seq   11  "1co 6"     seq   14  "gal 4"     seq   8   "1ti 3"     seq   9   "jam 3"     seq   13  "rev 9"     seq
15  "act 24"    seq   21  "1co 7"     seq   11  "gal 5"     seq   8   "1ti 4"     seq   9   "jam 4"     seq   8   "rev 10"    seq
17  "act 25"    seq   7   "1co 8"     seq   8   "gal 6"     seq   12  "1ti 5"     seq   11  "jam 5"     seq   14  "rev 11"    seq
18  "act 26"    seq   14  "1co 9"     seq   12  "eph 1"     seq   12  "1ti 6"     seq   15  "1pe 1"     seq   11  "rev 12"    seq
24  "act 27"    seq   15  "1co 10"    seq   11  "eph 2"     seq   10  "2ti 1"     seq   13  "1pe 2"     seq   12  "rev 13"    seq
19  "act 28"    seq   16  "1co 11"    seq   10  "eph 3"     seq   12  "2ti 2"     seq   13  "1pe 3"     seq   15  "rev 14"    seq
17  "rom 1"     seq   14  "1co 12"    seq   15  "eph 4"     seq   8   "2ti 3"     seq   11  "1pe 4"     seq   6   "rev 15"    seq
15  "rom 2"     seq   7   "1co 13"    seq   14  "eph 5"     seq   10  "2ti 4"     seq   8   "1pe 5"     seq   13  "rev 16"    seq
13  "rom 3"     seq   21  "1co 14"    seq   11  "eph 6"     seq   9   "tit 1"     seq   12  "2pe 1"     seq   12  "rev 17"    seq
13  "rom 4"     seq   26  "1co 15"    seq   14  "phi 1"     seq   7   "tit 2"     seq   15  "2pe 2"     seq   17  "rev 18"    seq
11  "rom 5"     seq   10  "1co 16"    seq   14  "phi 2"     seq   8   "tit 3"     seq   12  "2pe 3"     seq   14  "rev 19"    seq
11  "rom 6"     seq   13  "2co 1"     seq   12  "phi 3"     seq   11  "phm"       seq   6   "1jo 1"     seq   11  "rev 20"    seq
13  "rom 7"     seq   9   "2co 2"     seq   11  "phi 4"     seq   8   "heb 1"     seq   16  "1jo 2"     seq   23  "rev 21"    seq
20  "rom 8"     seq   9   "2co 3"     seq   15  "col 1"     seq   11  "heb 2"     seq   12  "1jo 3"     seq   13  "rev 22"    seq
17  "rom 9"     seq   10  "2co 4"     seq   12  "col 2"     seq   9   "heb 3"     seq   11  "1jo 4"     seq   7 "BIBLE NOTES"seq
11  "rom 10"    seq   11  "2co 5"     seq   11  "col 3"     seq   9   "heb 4"     seq   11  "1jo 5"     seq   87 blocks free.
19  "rom 11"    seq   9   "2co 6"     seq   9   "col 4"     seq   7   "heb 5"     seq   7   "2jo"       seq
10  "rom 12"    seq   11  "2co 7"     seq   6   "1th 1"     seq   10  "heb 6"     seq   7   "3jo"       seq
9   "rom 13"    seq   13  "2co 8"     seq   11  "1th 2"     seq   15  "heb 7"     seq   15  "jud"       seq
```

WANT ADS

RATES: $10.00 per nominal line of 110 characters & spaces or fraction thereof. CBUG may alter printing layouts and spacing, make minor modifications to standard abreviations, etc.

FOR SALE

1.) B-128 System, approx 5 hrs use. B-128, Zenith Amber Monitor, 4023, 8050, SSII SB & Calc Result. 8050 needs adjustment. $500 or B.O. Bob Maxon 615 690 9463 6:00 - 9:00 pm EST.

2.) B-128 system complete hardware & software. $500.00. Send stamped envelope for list & details. Carl Myers, 134 Knollwood Ln, Greenvile S.C. 29607. 803 233 6931.

3.) (2) B128''s, 4023, 8023P, SFD1001, Modem/300, 1530 Datasette, IEEE cables, Protecto B128 Prog Guide. Lots of software. R. Moyer, 215 432 0732.

4.) 5 brand new copies Superscript II, 4 brand new copieis Superbase. $6.50 ea. Warren Swan, 1 N. 114 Woods Ave., Wheaton Il. 60188. 312 665 1514.

5.) B128, 8050, 4023, USI green monitor, cables, etc. Roy Sulistijono, 144 S. Hamlin çC, Orange, Ca. 92669. Little use. Make offer.

6.) Std. Protecto B128 package w/ much software - $500. Tom Millsaps, 1245 Sharon Dr., Titusville, Fl. 32796. 305 267 9334.

7.) Barely used B128 system, amber monitor, 8050 drive, printer, 300 baud modem, software & manuals. $550 plus shipping or offer. Ron Hale 1637 W. Addison, Chicago 60613.
8.) B128 working extra system: 15" monitor, Micropolis Safari type 8050 drive, $600. Will Split. 2031 drive & svc info $65, Calc Result & The Power of Calc Result $50, 8050/8250 Service Manual $20.00. New power supply $30.00. Offers. Warren Kernaghan, 901 E. 108th St., Kansas City, Mo. 64131. 816 348 9615.

9.) B128, 8250, 8023P, MONITOR, SSII, SBI, CALC RESULT, ACCT. REC, ACCT. Pay, GEN. LED, AND MORE - $750 - 509 345 2455.

◆◆◆◆◆◆

SERVICE CENTER:

TYCOM INC
503 East Street
Pittsfield, MA 01201
(413) 442-9771

Authorized Commodore service center, we repair B128, 8032, 4023, 8023, 6400, 4040, 8050, 8250, 2031, C-64, C-128, etc. Rates are $50/Hr plus parts. We will diagnose & quote repair cost for $25. Normal turn-around time is 5 business days. We ship via UPS. Limited supply 8032's & 4023's (new) at $189 each.

◆◆◆◆◆◆

RIBBONS - MEMORY KITS - FANS - POWER SUPPLIES

"IF IT PRINTS, I HAVE IT."

				12+	1-11
CBM	4023	Multistrike	ea.	$5.50	$6.00
CBM	8023	Nylon Fabric	ea.	$5.00	$5.50
CBM	1525	Nylon Fabric	ea.	$5.75	$6.25
CBM	1526	Multistrike (Carbon)	ea.	$5.50	$6.00
CBM	MP801	Nylon Fabric	ea.	$6.50	$7.15
CBM	PET	Nylon Spool	ea.	$2.90	$3.20
CBM	PET	Multistrike	ea.	$4.05	$4.40
CBM	6400	Nylon Fabric	ea.	$4.45	$4.95
CBM	6400	Multistrike	ea.	$4.45	$4.95

Minimum Postage: $2.00 plus $0.25 each ribbon
Please give Street Address for UPS shipments -- Foreign Add 35%

Memory Expansion Kits (each)	($3.00 UPS)	$75.00
Memory Upgrade (In Shop)		$100.00
Fan Kits	($5.00 UPS)	$25.00
B128-80 Power Supplies	($5.00 UPS)	$50.00
SuperPet	($25.00UPS)	$375.00

Will accept COD's

S & W Supply Company
Jim L. White
5308 Timberline Trail
Rapid City, South Dakota 57702
(605) 348-3696

***

CompuServe #70007,1147
[Prices Subject to change without notice.]

CUSTOM IEEE CABLES

CBUG has received numerous calls from members wishing to have special IEEE cables. Longer, extra plugs, etc.

My most complex system has a B128, 2 8050's, a 4040, plus printers -- 8300P, 8023P a star NL-10 and daisywriter 2000. Some of the equipment is on the buss all the time, some others I'd like to have on all the time; and a few have to remain off buss if they are not turned on. NOTE: Connecting more than two un-powered devices will devistate the buss -- you will get data errors, erase wrong disks, etc. Thus, shutting down 2 of three printers as long as everything else on the line is powered is usually OK. One exception though, the 6400 printer and some early CBM dot matrix printers will lock up the buss in certain modes.

Right along with such a complex system is the length of cable runs -- often the 1 meter cables just don't reach. Plus it is expensive to have a cable for each device.

SOLUTION: multiple male (and female) connectors on a long flat cable. CBUG has acquired the equipment to prepare special IEEE cables to your sketch. Since all CBM IEEE periferals receive information via a female socket on the rear, normally there is no reason to have even one single female on the cable -- mere plug the first male into the back of your P to I cable at the first periferal, and string it along one device to the next.

To order make a sketch similar to that below:

```
M---------------M---------------M---------F--------M-----------------------------M
       4'             3.5'          1'        2'                7.5'
```

The sketch does not need be to scale. Be sure to allow slack between each run for servicing your installation and to allow for turning plugs in the proper attitude device to device -- atleast 6" each side of a plug.

PRICING:		
	BASE CHARGE PER CORD	$15.00
	PER RUNNING FOOT OF CABLE	1.00
	PER CONNECTOR	5.00
	Ill. Residents add 7% sales tax	
	Shipping & Handling per order	2.00

NOTE: This is a flat cable crimp on connector system. Connectors are single faced, either Male or Female. Not both as on the regular round wire IEEE cables. We suggest you do not exceed 30' prox in overall cable on your system. Do not use flat cable in close proximity to your wife's favorite TV, etc.

Please order on a separate sheet of paper including your name, address and phone even if submitted with a regular CBUG order form. Cable orders will usually be shipped separately due to fabrication time -- allow extra week to 10 days.

This offer is for IEEE connectors on the cable ONLY. We do not do the Pet to IEEE end. You only need one of those -- for the computer in any event.

PHYSICAL EXAM

NOW FOR THE 8250 & SFD 1001

$35.00 ea, from CBUG

8250 or SFD 1001 order #12192; 8050 order #12219; 4040 order #12238
1541 order 12223; 1571 order #12242

!!!!!!!!!! B-1024 1 MEGABYTE MEMORY EXPANSION CIRCUIT BOARD !!!!!!!!!!

* In the Low Profile B-128, Implements Banks 0 through 14 with 256K by 1, 150ns socketed Dynamic Memory. This increase provides 8 Documents with SuperScript II; 28,560 Input Values with Calc Result; 14 8032s with the 8432 Emulator; and enables the operation of the CBM-256 Operating System with the Alternate Operating System Expansion Circuit Board!

* Fully Assembled and Tested. Plugs internally onto pin fields in the low profile B-128. You can install in minutes at your site!

* The original configuration of a stock B-128 can be restored at any time by simply removing the board. This is a non-destructive upgrade!

* Includes Pin Fields for future I/O, Static RAM for $0800 to $1FFF in Bank 15, Documentation for Installation, Parts Layout, Schematics, and a Machine Language DRAM Memory Test Utility on 8050 Diskette!

!!!!!!!!!! ALTERNATE OPERATING SYSTEM EXPANSION CIRCUIT BOARD !!!!!!!!!!

* B-128 "Low Boys" with 256K Memory can run the CBM-256 Operating System and CBM-256 "High Boys" can run the B-128 Operating System!

* 8050 Diskette Included! Load In and Boot Up a new Alternate Operating System in seconds with a Shift-Run from Drive 0!

* Sockets for your Original COMMODORE ROMS plus Two Alternate Areas that can be set up as STATIC RAM (one area supplied), or PROM (see below)!

* Execution of Original and Alternate Areas, Read/Write selection and protection of on board STATIC RAM, and generation of a Hardware Cold Start Reset are all selectable through software!

* Mounts internally on your computer's main circuit board or on the B-1024 1 Megabyte Memory Expansion Circuit Board!

* B-128s with 128K Memory (banks 1&2) require an additional 128K Memory (banks 3&4) to run the 256K Operating System. Naturally we recommend the B-1024 1 Megabyte Memory Expansion Board to implement this increase!

* Alternate 2764-25 PROM Set is available that plugs into one of the Alternate Areas, jumpers select which PROM Set boots from power up!

ALTERNATE OPERATING SYSTEM EXPANSION BOARD for "CBM" (hi profile)	$ 89.95
ALTERNATE OPERATING SYSTEM EXPANSION BOARD for "B" (low profile)	$ 89.95
ALTERNATE 2764-25 PROM SET with above (not available separately)	$ 19.95
US Funds, Iowa add 4%, Shipping/Handling	$ 7.00

B-1024 1 MEGABYTE MEMORY EXPANSION CIRCUIT BOARD for the Low Profile B-128!

B-1024 1 MEGABYTE MEMORY EXPANSION BOARD	$349.00
US Funds, Iowa add 4%, Shipping/Handling	$ 10.00

ANDERSON
COMMUNICATIONS
ENGINEERING
2560 Glass Rd. NE.
Cedar Rapids, Iowa
U.S.A. 52402

24K RAM/ROM CARTRIDGE with Case for the B-Series:

Assembled+Tested (Socketed RAM Memory ICs)	$ 39.95
Bare Circuit Board and Case	$ 14.95
US Funds, Iowa add 4%, Shipping/Handling	$ 3.00

If you may need lots of accurate information by telecommunication searching of databases from Dialog, BRS, SDC, NLM et al. (Delphi etc can be used with LitSearch), unique dedicated State-of-the-art software is now available to save time and money$:

LitSearch.

For this task, LitSearch is the best software available on ANY computer - most intelligent & money saving & error saving & professionally flexible & fast & automatic & fun & easy on eyes & informative & and easy to use - of any software of its kind, on ANY computer - but it runs ONLY on the B series (and maybe B series clones, if IBM ever comes up with one).

FEATURES:

■ 1 key (RUN) AUTOMATIC operation, or manual, or in between (you select the portions you want automated)
■ Sequence of UNLIMITED number of searches each with their own database vendor , (1 key automatic ok)
■Real-time downloading SIMULTANEOUSLY to disk, screen and printer at 1200 baud
■UNLIMITED downloading file length (no buffers or xoff to mess with)
■Records elapsed timeS/expenses so you can improve next time and save more $
■Uses strategies you've prepared with SUPERSCRIPT - keeps in RAM
■TRANSLATES most of your strategy appropriate to each specific vendor
■Powerful EDITING capabilities online - even what host just sent you
■CALCULATES formulas in your strategy (for record # ranges to be requested)
■RENUMBERS search set numbers within your strategy while online (if needed)
■PROTECTION of password (searching is fun as a "joint" effort with someone)
■Petspeed compiled. No copy protection other than your honesty
■Print any part(s) of downloaded file, to screen or printer (matrix or daisy wheel) or alternating
■Over 45 pages of documentation (minimizes computer jargon), on disk

SPECIAL NOTE: See article this issue (Fall 1987) by Harlan Schmidt, esq. regarding Litersearch used in legal research.

Purchasing:

Requires B128 (expanded ok), CBM IEEE drive (e.g. 8050), Hayes compatible modem.

Provide the following information along with personal check or money order for $175.00: Disk drive (for format), printer (specify interface), modem (brand, model & baud), & Custom screen label desired (e.g. your name).

Mail to: Bing Hart
1408 E 19th St.
Lawrence, Ks., 66046

Tel. 913 843 7668 after 10 am CST for more information

COL.J.E.O'HALLORAN
HORSAO FARMS
RT.2 OWL CREEK ROAD
HIAWASSEE,GA. 30546

16 December 1987

Mr Norman Deltzke
4102 N Odell
Norridge, IL 60634

Dear Norm,

The Tax Programs for the B128 and B1024s for the 1987 tax year are ready.
Several new features have been added this year including automatic calculation of your standard deduction and automatic selection of either the standard deduction or the itemized deduction, whichever is the greater. Your tax will be automatically computed IF you meet certain criteria and MAY be used in any case to get an approximately correct figure which you can compare to the figure you get from the tables before manually entering the table figure. Of course the program has been completely redone to reflect the changes made by the 1986 Tax revision.

The Price has not changed and you may have your program customized to include all the forms necessary for your filing. A program of up to 10 pages may be handled with a B128 while more than 10 will require added memory of at least 256K. (The Programs are all developed on a B1024 using the Anderson 1 meg board expansion.) For any program, you may deternmine the price by counting the pages required and multiplying that figure by $5 and adding $20. The standard program---1040, A, B, C, F and 4797 (9 pages) is $65.00 for example.

I have also added a very special and valuable feature which will allow you to very closely estimate your 1988 taxes and you may even make monthly entries during 1988 to track your 1988 tax in order to make your plans as you go through the year from the very beginning. There will no doubt be some changes made before the 1988 tax year which will require some or all of the program to be updated but it is expected that the planning you will be able to do with this program will be accurate enough to Make or SAVE you some good money.

One thing I would call to your attention for 1988 tax year planning--- IF you expect to have TAXABLE INCOME in excess of $171,090 AND you have dependent children OVER 14 you should look very carefully at the fact that an additional $10,920 of income PER CHILD will be taxed at 33% rather than 28%. If a solution is not apparent to you, I would suggest that you discuss this with your CPA to determine the best way to eliminate or lessen that penalty.

Please send your order, with specifications and check, as early as possible.

Sincerely,

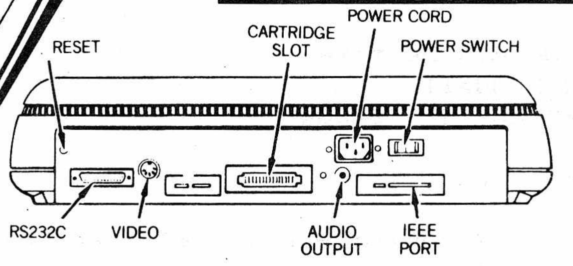
64K IEEE Print Buffer

Only $199

If you are tired of waiting, get the Microshare 64K Print Buffer.

Features

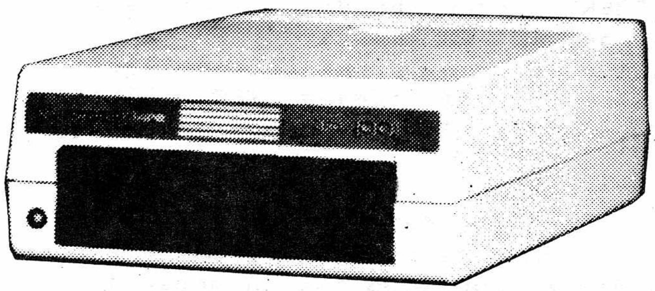
- Frees your computer for other tasks
- Stores almost 30 minutes of printing
- Helps make you and your employees more productive
- Copy and clear buttons
- IEEE and parallel Centronix output



B-128 Lo Profile

We have just purchased the last of the lo and hi profile B-128 computers from Commodore. We have both new and used models of both of these computers. If you would like to have a backup computer for your system then this is it! When these are gone there are no more! At this time it is certainly very prudent to have a backup computer. If your computer goes down and has to be repaired, you will certainly feel more secure knowing you have a replacement if it takes time to find parts for the repair!

PRICED AT **$145** US
ADD $10.95 SHIPPING US



SFD 1001 1 Megabyte Drive double sided 8250 format IEEE interface

PRICED AT **$149.95** US
ADD $9.95 SHIPPING US

COMMODORE 8000-9000 SOFTWARE & MISC.

9000 Superpet	$199.95
64K exp for 8032	**$125**
Pet Switch	**$199**
Pet Daughters	**$105**
BPI General Ledger	**$25**
BPI Accts Payable	**$25**
BPI Job Cost	**$25**
BPI Accts Receivable	**$25**
BPI Inventory	**$25**
Superscript 8032	**$79**
Superbase 8096	**$79**
OZZ Database	**$25**
Legal Time Acc.	**$25**
Dow Jones Program	**$25**
Info Designs 8032 Accounting System	**$50**
Superoffice 8096	**$149**
Calc Result 8032	**$89**

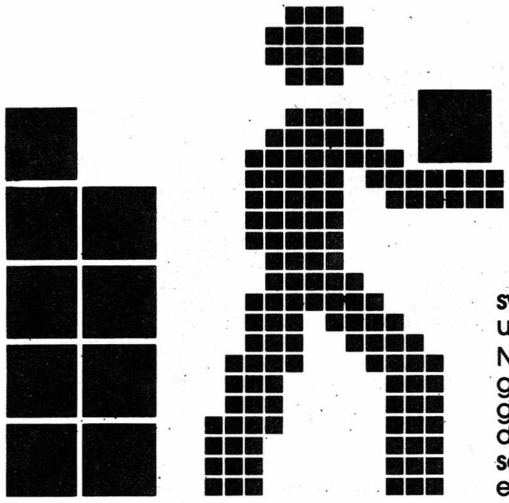
ORDER NOW WHILE STOCK LASTS!

Send or call your orders to: **Northwest Music Center, Inc. 539 N. Wolf Rd., Wheeling IL 60090. 312-520-2540** For prepaid orders add 16.90 for 8023p, $25 Superpet, 9.95 SFD 1001, 10.95 B-128. 9.95 4023p, 15.95 9090 and 4.95 64K memory expansion. For software add $3.00 for first and $1.50 for each additional book or program. Canadian shipping charges are double U.S. International orders, call for shipping. For C.O.D. orders add 1.90 per box shipped. All orders must be paid in U.S. funds. Include phone numbers with area codes. Do not use P.O. Box, only UPS shippable addresses. A 2 week hold will be imposed on all orders placed with a personal or business check. C.O.D. orders shipped in U.S. only and cash on delivery, no checks. 30 day warranty on all products from NWM, Inc. No manufacturer warranty. NWM reserves the right to limit quantities to stock on hand and adjust prices without notice!
All prices quoted in US dollars.

International Orders Call For Shipping Charges

NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) **Phone (312) 520-2540**
Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)



NWM's INVENTORY CONTROL SYSTEM*

This system was developed by a user for users. It seems that most systems are developed by computer engineers that never see or use the product when finished.

NWM, inc. chose the toughest software analyst we could find to give a complete and unbiased review. Bob Loefler is the gentleman that tore apart the CABS Accounting system and documented all the bugs in most of the modules. To have someone of this caliber make suggestions means better, more efficient software available for CBUG members.

- loads program modules in less than **8 seconds** (superbase 2) to main menus in **3 seconds** or less
- on screen **pop-up calculator** in transaction modules
- most data centered functions use the **calculator keypad**
- versatile report features allow for **3 ways** to print the same report. User selects the fastest method
- built in sophisticated export program allows for **complete packing** of the database
- type ahead feature allowed
- you can display reports on screen
- access to superbase menu for user developed applications

Now it's even better!

Listen to how NWM has improved this already great Inventory Control System...

- **Partial-match key search now allowed in transaction modules.**
- **User can search forward and backward through file while in transaction modules.**
- **Elimination of most tiresome prompts for faster movements between subfunctions and menus.**
- **Simplification of entering prices and costs that have not changed when entering transactions.**
- **User programmable calculator allows you to turn the calculator function on or off within the transaction modules. If the user does not need the calculator, it saves time by eliminating the prompt and bypasses the calculator function.**

Only a $10 upgrade charge for owners of previous versions of NWM Inventory.

WE SUPPORT OUR CUSTOMERS!

And listen to our prices!

B Version 1 8050	**$39.95**	C-128 Version 1 1571	**$24.95**
B Version 2 8050	**$39.95**	B-128 Version 1&2 8050...	**$44.95**

U.S. shipping and handling charge $3.95 (Prepaid)

NWM, inc. • 539 N. Wolf Rd. •Wheeling, IL 60090

(312) 520-2540 Hours— (Voice) **Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00** (24 Hour Order Recorder)

Superbase is a registered trademark of Precidion Software.
B128 and C128 are registered trademarks of Commodore Business Machines Limited.
©1986, NWM, Inc.

*Requires use of Superbase®

Super Office
List
$~~149~~00

SALE PRICE
$49.95* U.S.

SUPER OFFICE

WITH SUPERBASE 2

ALL THE GREAT FEATURES OF SUPERSCRIPT II AND SUPERBASE INTEGRATED IN ONE FANTASTIC PACKAGE

Store important data then print out personalized custom reports inserting totals and names in the proper places. Do invoices, past due notices and many other things with Super Office. With Super Office you can do anything a small business needs.

Super Office is the ultimate in integrated programs for the "B" series market. Some features include left and right margin settings, tabs, decimal alignment, right justification, underlining, bold print, page numbering and a whole lot more. Super Office is easy to use with simple function key commands and an easy to use manual. There is even a spelling checker to keep you from making embarrassing spelling mistakes. We challenge you to find an easier integrated program or one with more features than Super Office. Includes the ultimate data base at an affordable price. You can do anything that requires data storage with Super Office. Simply set up your screens for easy entry of information (up to 4 screens). Then set up the way you'd like the information to be printed. Super Office does the rest! This program has full calculation capabilities and full data storage manipulation. You do not have to reload either program to access the other. Automated merging through the Super Office application generator.

(must have 256K-1 Meg. upgrade to operate)

* Purchase subject to signed disclaimer.

TAKE A LOOK AT THESE GREAT SUPER OFFICE FEATURES

- Help Screens
- Bibliographies
- Document Chaining
- Home Budget
- Sales record
- Built-in programming language
- Inventory
- Highlighting
- Up to 127 Fields per Record
- Formatting
- Calculation capabilities
- Up to 4 Data Entry Screens per File
- Form Letter Printout
- Mailing list
- Better than dbase II
- User definable screen formats
- Applications Generator
- Up to 15 files per Database
- Keyed Access
- Report Generator
- Macro Capability

Shipping and handling charges $3.95

IF YOU VALUE YOUR TIME, THEN IT'S TIME TO BUY SUPER OFFICE

MWM Rebate Plan

Super Office is shipped with the Superscript II, Superbase I and Super Office Insert

Buy one of our expanded B-256 computers and purchase Super Office for only:

$39.95 with SBI and SSII Manuals

$29.95 without SBI and SSII Manuals

Without B-256 Purchase:

$49.95 with SBI and SSII Manuals

$39.95 without SBI and SSII Manuals

NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

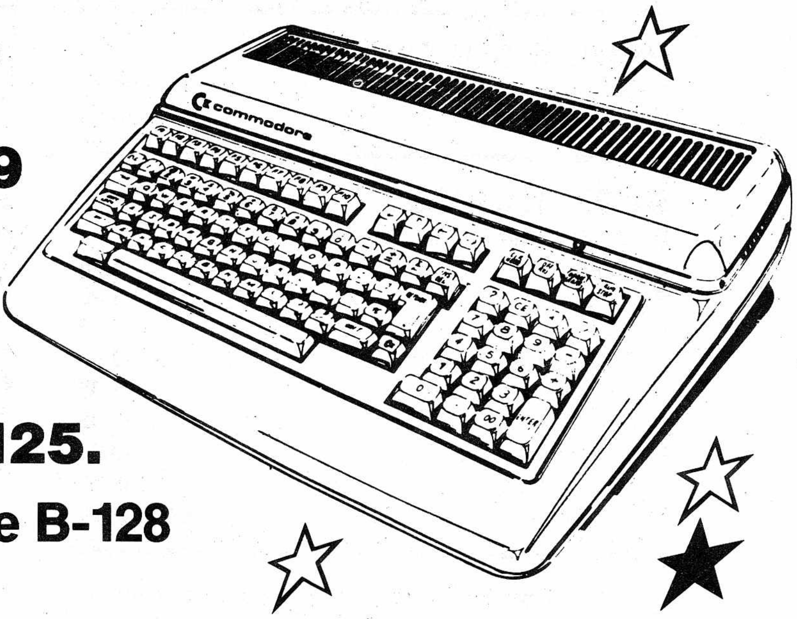
Never before at such incredibly low prices!!!

Buy Superoffice and get the NWM Inventory System for only $19.95

B-128 lo profile $99
(Commodore rehabbed)

B-256 upgrade $199
(Commodore rehabbed)

With any purchase of a B-128 or B-256 purchase an SFD-1001 for only $125.



1 megabyte hi-profile B-128
(very limited quantities)

Computer binders as used in the CABS Inventory system, fits most commercial manuals only 99¢ each (minimum purchase 10)

With any purchase over $25 get a Free Programmers Reference Guide or a Free Superbase program.

CP/M-86 / MS-DOS Corner

limited quantities on new and used programs

Calc Result/Word Result IBM pack	$119.00
Dbase II	$ 99.00
Move-it Communication prg	$ 99.00
Wordstar	$ 99.00
DR Graph IBM CP/M-86	$ 25.00
C Compiler	$ 99.00
C Basic	$ 59.95
C Basic Compiler	$ 99.00
Personal Pearl Database	$ 79.95
Crosstalk IBM CP/M 86	$ 59.95

64K IEEE Print Buffer

Only $199

If you are tired of waiting, get the Microshare 64K Print Buffer.

Features
- Frees your computer for other tasks
- Stores almost 30 minutes of printing
- Helps make you and your employees more productive
- Copy and clear buttons
- IEEE and parallel Centronix output

YOU DON'T NEED A CENTRONIX INTERFACE

NORTH WEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

Hacker's Corner

One of a Kind • Surplus • Monthly Special • Closeouts

Limited quantities to stock on hand

PRECISION SOFTWARE

Superbase	**$ 9.95**
Superscript	**$19.95**

HANDIC SOFTWARE

Calc Result	**$79.95**
Word Result (special order)	**$79.95**

C.A.B.S. ACCOUNTING

General Ledger	**$ 9.95**
Accounts Receivable	**$ 9.95**
Accounts Payable	**$ 9.95**
Order Entry	**$ 9.95**
Payroll	**$ 9.95**
Buy 5 for only	**$24.95**

B-128 GOODIES

B-128 Programmable Reference Guide was $29.95	**$ 5.95**
P-I	**$24.95**
I-I	**$29.95**
SFD 1001 1 meg drive	**$149.95**
Superbase The Book	**$14.95**
Superpet 9000	**$199.95**
Pet Switch	**$199.00**
Pet daughters	**$105.00**
8050 rehabs from	**$250-$400**

CALL FOR INFORMATION ON 8000 & 9000 SERIES SOFTWARE AND MISC.

SUPER DISK DOC

It's your choice, you can get it now . . .

If you want control over your disks then you need Super Disk Doc!

operates from easy to use duck shoot menus
examines bytes on your disks
interprets in English, hex or ascii
modify and save disk data
recovers previously unreadable files

Never before has there been such a powerful and easy to use disk utility like this for Commodore computers!

or . . . you can wait until its too late!

PRINTERS

8300p Xerox Diablo 630 prt	**$499.00**
4023p 100 cps rehab	**$99.00**
Gemini 15 x 120 cps	**$159.95**
Smith Corona DM200 RS232 and parallel	**$179.95**

ONE OF A KIND ITEMS

MJ-10 COLOR MONITOR	**$125.00**
AVATEX 1200 MODEM (Hayes Compatible)	**$109.95**
INTERFACE SERIAL to IEEE	**$59.95**
MONOCHROME MONITOR	**$69.95**
INTERFACE IEEE to PARALLEL	**$89.95**
9090 7.5 meg HARD DRIVE, REHAB	**$495.00**

ORDER NOW WHILE STOCK LASTS!

Send or call your orders to: **Northwest Music Center, Inc. 539 N. Wolf Rd., Wheeling IL 60090. 312-520-2540** For prepaid orders add 16.90 for 8023p, $25 Superpet, 9.95 SFD 1001, 10.95 B-128. 9.95 4023p, 15.95 9090 and 4.95 64K memory expansion. For software add $3.00 for first and $1.50 for each additional book or program. Canadian shipping charges are double U.S. International orders, call for shipping. For C.O.D. orders add 1.90 per box shipped. All orders must be paid in U.S. funds. Include phone numbers with area codes. Do not use P.O. Box, only UPS shippable addresses. A 2 week hold will be imposed on all orders placed with a personal or business check. C.O.D. orders shipped in U.S. only and cash on delivery, no checks. 30 day warranty on all products from NWM, Inc. No manufacturer warranty. NWM reserves the right to limit quantities to stock on hand and adjust prices without notice!

All prices quoted in US dollars.

NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

CBM 256-80 w/8088 Co-processor

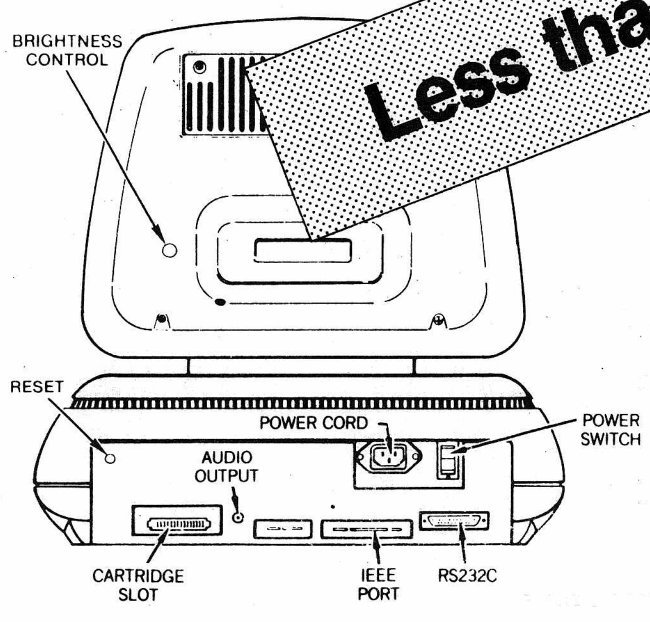
Call for Details
Very Limited Quantities

We are offering a limited number of used CBMX 256-80 computers for only $400. These come installed and tested with the original 8088 co-processor boards that until recently were hidden in the Commodore research labs.

The CP/M 86* operating system was implemented on the B system and has been tested and found to be very reliable. It had previously been stated that the co-processor would only work with the CBM 256-80. This is not true! The PLA that is installed on the hi-profile motherboard determines whether the operating system will run on that particular machine. We will also be offering the 8088 co-processor board for $150 providing the purchaser sends in their working hi-profile motherboard with the correct PLA. Call for details!

All the generic CP/M 86 software that we have tested will operate on the B series machine with an 8050 drive. The SFD and the 8250 can also be used with some restrictions. These same programs can also be run under the Digital Research CP/M 86 and Concurrent PC Dos operating system on an IBM. Therefore an investment in software is not wasted as it can be ported to other compatible computers.

*CP/M 86 is a trademark of DRI Inc.



CBM 128 & 256 features:

- SWIVEL MONITOR
- ADJUSTS HOR. and VERT.
- DETACHABLE KEYBOARD
- 9 x 14 PIXEL DISPLAY
- INCREDIBLE RESOLUTION
- DESIGNED FOR 2 INTERNAL DRIVES

Priced from $225 to $400 US
SHIPPING CHARGES EXTRA!

You really have to see the green phosphor display to believe it. This model has at least as good a display as the other company with those three big letters.

NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thur. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

*	COMPANY		PART NAME	
*	CMD	✓	THE MANAGER	
*				
*		✓	VISICALC	
*	COMMODORE	✓	DOW JONES	
*	COMMODORE	✓	LEGAL TIME ACCOUNTING	
*	COMMODORE	✓	WORDCRAFT 80	
*	COMMODORE	✓	PERSONAL TAX CALULATOR	
*	COMMODORE	✓	ASSEMBLER DEVOLPMENT SYST	
*	COMMODORE	✓	PASCAL DEVELOPMENT SYSTEM	
*	COMMODORE	✓	OZZ INFORMATION WIZZARD	
*	CYBERIA		FARMERS WORK BOOK	
*	CYBERIA	✓	CYBER-FARMER 80	
*	S.O.S. INC	✓	SILICON OFFICE	
*	PROF.SOFT	✓	WORDPRO 4 PLUS	
*	BPI	✓	JOB COST	
*	BPI	✓	GENERAL ACCOUNTING	
*	BPI	✓	PAYROLL	
*	BPI	✓	INVENTORY CONTROL	
*	INFODESIGN	✓	DEALER TRAING MANUAL	DEMO PGMS
*	INFODESIGN	✓	INVENTORY MANAGEMENT	
*	INFODESIGN	✓	PAYROLL	
*	INFODESIGN	✓	GENERAL LEDGER	
*	INFODESIGN	✓	ORDER ENTRY/POINT OF SALE	
*	INFODESIGN	✓	ACCOUNTS RECEIVABLE/BILL	
*	INFODESIGN	✓	ACCOUNTS PAYABLE/CHECK	
*				
* *	901001-01	✓	12" CRT	
* *	901000-04	✓	9" CRT	
* *	8032	✓	CBM COMPUTER	
* *	8050	✓	DISK DRIVE	
* *	~~8052~~ 8250	✓	DISK DRIVE 1MEG.	
* *	2031040-01	✓	2031 PCB PET	
* *	8032090-07	✓	PCB ASSY. SUPER PET	
* *	8032090-05	✓	4016 PCB UNIV	
* *	8032090-01	✓	8032 PCB ASSY	
* *	320284-00	✓	BUS. KEYBOARD	
* *	8032090-03	✓	4032 PCB UNIV	
* *	320122-00	✓	GRAPHIC KEYBOARD	
* *	8032051-01	✓	POWER PACK CPU	
* *	8050029-01	✓	POWER PACK FLOPPY	
* *	324038-01	✓	BMB PCB 4040 FLOPPY	
* *	805000-04	✓	DIGITAL MAIN PCB 8050	
* *	950550-00	✓	SHUGART DRIVE	
* *	8050044-01	✓	ANALOG PCB 8050 TAN	
* *	320820-02	✓	DIGITAL PCB 4040	
* *	805006-01	✓	ANALOG PCB 8050 MIC.DRV.	
* *	320817-01	✓	ANALOG PCB 4040	
* *	321444-00	✓	DISPLAY PCB 12" CRT	
* *	321443-00	✓	DISPLAY PCB 9" CRT	
* *	8050031-01	✓	MICROPOUS DRIVE	
* *	8050002-01	✓	DIGITIAL PCB 8050 MIC.DRV	
* *	324042-01	✓	COMPU-THINK PCB SUPER PET	
* *	8050047-00	✓	TANDON DRIVE	
* *	324056-01	✓	SECURITY PCB SUPER PET	
* *	9000022-00	✓	SWITCH SET/PCB SUPER PET	
* *	9000092-00	✓	ROM SELECT MOD KIT	
* *	C-64 00-00	✓	VIC SWITCH MULTI USER	
* *	8032 CBM	✓	HORZ. TRIPLE FLIP QTY. 2	
* *	ADA 1600	✓	PARRELL PRINTER ADPATER	
* *	ADA 1450	✓	SERIAL PRINTER ADAPTER	

Contact:
Thomas W. Burkholder, Pres.
Champaign County Telephone
Sales & Service
114 Scioto Street
Urbana, Oh. 43078
513 653 7605

COL. J. E. O'HALLORAN
HORSAO FARMS
RT.2 OWL CREEK ROAD
HIAWASSEE,GA. 30546

22 November 1987

Dear Norm,

I noticed an error in the printout of my symbols program in the latest CBUG ESCAPE on page 31. The back-slash symbol had been inadvertently converted to the British Pound sign, (ESC 2). This could have occured when whoever typed the program for publication might have used ESC SHIFT 4 rather than ESC 2. I recalled my letter to see if I had made the error and I had not. (I am enclosing copy of original letter with a copy of page 31 with the errors marked in the publication.

ESC SHIFT 4 will produce the British Pound sign from the Keyboard WITHOUT any 'MICRO' being used, ei., ESC SHIFT 4 = £. IF the article was taken directly from the disk to computer to printer and photographed for publication---I am a loss to explain how the errors occured.

Incidentally, while I am 'on the air' I would like to pass along a few tips to CALC RESULT users:

1. IF you have an apparent refusal to SAVE (CTRL, D,S) with error statements such as "Write protection on", Read Error, Disk I.D. Mismatch, and your feel certain that none of these errors apply---inset TWO disks, Formatted and Program backup, Type CTRL Q(uit) and Y(es) and the file in memory will be saved. Otherwise you will have wasted your time in preparing the spreadsheet.

2. When entering RELATIVE formulae the computer should be in AUTO CALC MODE or you must hit F5 (recalc) several times.

3. DO NOT use a DISK SPEED UP program with CALC RESULT. It will interfere with calculations and formula movements and generally foul up your work.

I am intrigued by the reactions of some to the HEAVY work and writings done by Tony Goceliak and by his reaction to that. I have read his lengthy and highly technical writing without fail even though much of it was of no benefit to me---AT THAT TIME. Having had to wait until past middle age to get my college education, I was forced to learn Aeronautics, Meteorology, Navigation and other disciplines through reading on my own. Based on my self study I was able to pursue a very fine career in Commercial and military aviation and business. I have been an avid reader of a myriad of subjects all my life and have been much enriched because of that, in several fields both vocation and avocation. Those who do not wish to read Tony's sometimes lengthy and technical articles do NOT HAVE TO READ them but many who do may benefit. I am reminded of the quote from some wise person, "those who WON'T read have no advantage over those who CAN' T".

Keep up the fine work, we appreciate it.

COL.J.E.O'HALLORAN

HORSAO FARMS
RT2 OWL CREEK ROAD
HIAWASSEE,GA 30546

25 June 1987

Re:Addenda & errata 4023/8023 lpi CODES.

Dear Norm,

I have reviewed the writings of the others referred to in my letter of the 21st and have the following to add:

Addenda:For SUPERSCRIPT II, NO mandatory blank lines are required preceding the embedding of the codes as David Ritterbusch says ARE required for SSIII.

Errata:It was David Ritterbusch who wrote about the 8023P codes, NOT Warren Swan as stated in my letter of the 21st.

You are welcome for the addenda and I am sorry for the error.

LINE SPACING COMMANDS FOR 4023 Printer/SSII

```
FOR 3 lines:REV(*)sa6,66<
For 6 lines:REV(*)sa6,33<
For 9 lines:REV(*)sa6,22<
```

8023P LINE SPACING CODES in SSII

```
FOR 6 lines per inch:REV(*)sa6,12<
FOR 8 lines per inch:REV(*)sa6,9<
FOR 9 lines per inch:REV(*)sa6,8<
FOR 3 lines per inch (for double spaced format):REV(*)sa6,24<
```

Multiply length of paper in inches by number lines per inch = page length code.

EDWIN R. BOWERMAN
47 PARSONAGE LANE
TOPSFIELD, MA 01983
(617) 887-5591

August 27, 1987

CBUG, Inc.
c/o Norman Deltzke
4102 N. Odell
Norridge, IL 60634

Dear Norman:

Using the Commodore 4023 printer for utilities and plotting frequently involves changing the lines/inch or upper/lower case defaults. The 4023 remembers many of these conditions and it is irritating to me to try to read a program listing printed at 12 lines/inch because I previously used a plot routine with that value.

A solution would be to precede the BASIC program with a printer reset command. The Model 4023 User's Manual says that sending a secondary address of 10 will reset the printer. This does not work at the beginning of a program because the printer takes more time to reset than the program allows. My solution is to insert a delay time after the reset command to allow the printer head to finish its excursion. A simple FOR/TO/NEXT loop is one way of doing this.
I use:

```
7 OPEN10,4,10:PRINT#10:CLOSE10:FOR J=1 TO 800:Z=300/J:NEXTJ:REM RESET PRINTER
```

This solution is reminiscent of the "clean-up" routines that we had to use in the early days of programming before the high level languages automatically set all arithmetic variables to zero and text variables to blanks. Of course the reset could be done at the end along with closing of all open logical files.

Yours cordially,

Edwin R. Bowerman

```
 5 rem field poster
 10 ask "(y or n) Select new file?                    FIELD POSTING UTILITY";a$:if
 a$="y"then file
 15 plen 66:tlen 66:display chr$(147)
 20 ask "field to post (do not enclose in brackets)";i$
 25 display "          Enter a set of two quote marks to delete contents of a fiel
d."
 30 ask "data to post";h$
 32 display chr$(147)
 35 a$="["+i$+"]=h$"
 40 ask "(y or n) Enter data into all records?";b$:if b$="n"then 100
 50 select f:select c:goto 57
 52 select n:eof 140
 53 select c
 57 do a$
 60 store the record
 65 select c
 70 goto 52
 100 ask "key name ('e' to end)";k$:if k$="e"then 140
 105 select k$:select c
 107 nmat display chr$(147):display "No key ... re-enter":for i=1to 500:next i:d
isplay chr$(147):goto 100
 110 do a$
 120 store the record:select c
 130 goto 100
 140 ask "(r)eturn to menu  (c)hoose another file  (re)peat program";a$
 150 if a$="r"then menu
 160 if a$="c"then run
 170 if a$="re"then 15
```

Dear Norman:

If it isn't too late, you might want to publish this program for use with SUPERBASE. I included this with SUPER TEACHER but it is usable with any SB files program. It allows batch posting of files or posting of individual files and especially is useful to correct file posting errors in any SUPERBASE program.

The program also demonstrates the use of the DO command which is the only command that allows the SB user to address fields from the key board instead of having to go to the line number to insert the field name desired for use in the program.

I hope this isn't too late for publication in the Summer issue. But if so, then it can make the next one.

[illegible]

Jon Whatley

I have volume 7 of the cbug escape in my hot little hands! Thanks again. When I saw Anthony Goceliak's column on a secret b-128 key , it was as if someone was reading my mind! I too was working on finding a use for the tab key, but was working on it from the opposite end. I found out that by pokeing certain values to locations 929 to 938 , I could program the tab key. By pokeing values of 1, 2, 4, 8, 16, 32, 64 and 128 to each of these locations I was able to make very limited use of the tab key. But now that i've read this article it is so easy to program the tab key, I can hardly beleive it. Here are three short one-liners as examples:

```
     10 for i = 10 to 80 step 10:printtab(i)"shift/tab":next
```

Entering this line early in your program will set the tab key to tab 10 spaces each time it is pressed. To return to the default setting, simply enter the line again where you want to go back to the default setting.

```
     10 for i = 20 to 80 step 20:printtab(i)"shift/tab":next
```

This line will do the same thing only now the tab key will tab 20 spaces instead of 10. Whatever you make the step number, use the same number after the = (equal) sign.

If you do not want the tab settings to be the same all the way across the screen, then using the following line, you can set the tabs at whatever amounts you would like.

```
10 printtab(5)"shift/tab"tab(20)"shift/tab"tab(40)"shift/tab"tab(70)"shift/tab"
```

note: shift/tab means to hold down the shift key and press tab. You should see a reversed "I" on the screen between the quotes. +see below|

```
 10 for i = 10 to 80 step 10:printtab(i)"▮":next
ready.


 10 for i = 20 to 80 step 20:printtab(i)"▮":next
ready.


 10 printtab(5)"▮"tab(20)"▮"tab(40)"▮"tab(70)"▮"
ready.
```

'Here are a some pokes to locations 929 thru 938 that allow you to set the tab key to a pre-set tab.

poke 929,128=1	poke 930,128= 9	poke 931,128=17	poke 932,128=25	poke 933,128=33
" " ,64 =2	" " ,64 =10	" " ,64 =18	" " ,64 =26	" " ,64 =34
" ,32 =3	" ,32 =11	" ,32 =19	" ,32 =27	" ,32 =35
" ,16 =4	" ,16 =12	" ,16 =20	" ,16 =28	" ,16 =36
" ,8 =5	" ,8 =13	" ,8 =21	" ,8 =29	" ,8 =37
" ,4 =6	" ,4 =14	" ,4 =38	" ,4 =30	" ,4 =38
" ,2 =7	" ,2 =15	" ,2 =23	" ,2 =31	" ,2 =39
" ,1 =8	" ,1 =16	" ,1 =24	" ,1 =32	" ,1 =40

poke 934,128=41	poke 935,128=49	poke 936,128=57	poke 937,128=65	poke 938,128=73
" " ,64 =42	" " ,64 =50	" " ,64 =58	" " ,64 =66	" " ,64 =74
" ,32 =43	", ,32 =51	" ,32 =59	" ,32 =67	" ,32 =75
" ,16 =44	" ,16 =52	" ,16 =60	" ,16 =68	" ,16 =76
" ,8 =45	" ,8 =53	" ,8 =61	" ,8 =69	" ,8 =77
" ,4 =46	" ,4 =54	" ,4 =62	" ,4 =70	" ;4 =78
" ,2 =47	" ,2 =55	" ,2 =63	" ,2 =71	" ,2 =79
" ,1 =48	,1 =56	" ,1 =64	" ,1 =72	" ,1 =80

Armand Carrier
20 Cole St
Torrington,Ct.   06790

COL.J.E.O'HALLORAN
HORSAO FARMS
RT.2 OWL CREEK ROAD
HIAWASSEE,GA. 30546

16 December 1987

Mr. Norman Deltzke
4102 N. Odell
Norridge, IL 60634

Dear Norm,

I know that in the past you have received a number of complaints and have even written notes in The CBUG ESCAPE concerning poor support from suppliers of services, software and hardware.

I would like to sound a pleasant note by remarking on a bit of extraordinarily excellent support which I recently enjoyed.

I was in the middle of the preparation of a LARGE and memory intensive project when I began having difficulties with SAVEs and LOADs. I also experienced the disappearance and re-appearance of entries already in the program and on the screen. I had ghostly appearances of characters NOT part of the program. I began a systematic elimination of possible causes using Physical Exam to see if the drives were culpable, switched computers and drives and finally ran the Memory Test that came with Gary Anderson's 1 meg board. The test looked great until on the walking 0 test there was a FAIL on bank 4 with 04 indicated as the type error, PASS through banks 5 & 6 then another FAIL on bank 7 with the same error indicated.

I immediately wrote Gary (this was on a Friday) outlining what I had found. Being eager to learn what had to be done to get my gear back in gear, I called him on Tuesday evening. He had received my letter and had already gotten the needed IC in the mail that day along with a shipping carton IN CASE the IC was not all that was needed. The IC arrived, I installed it, ran the memory test, it came up all PASSes and I went back to work with great gusto and much appreciation for the fine effort that Gary put forth to get me back on track.

All I can say is, "peoples' hearts really do go out to orphans". If anyone believes they could get this kind of support from one of the 'big' boys, I would like to talk to them about a great deal I have for them on stock in a company to build a bridge across the Bering Sea.

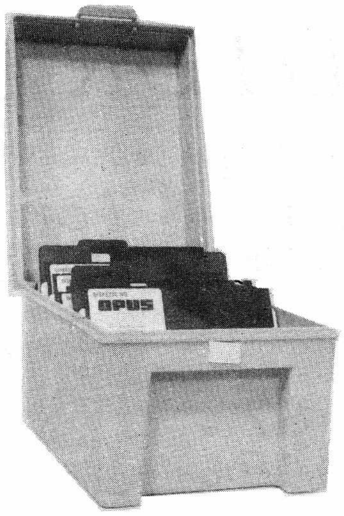
I am enclosing my check for $20 to cover my subscription.

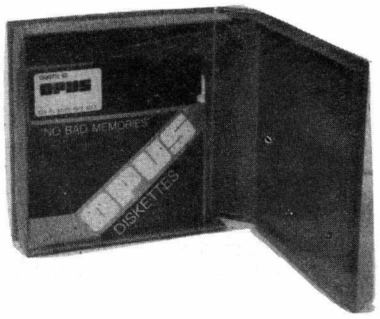
Warmest regards,

Jim

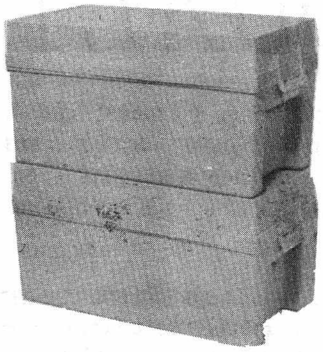
DATA CASE ORDER FORM

DD120L Holds 120 5¼ Disks
Locking Clear Top, Beige Body
10 Dividers w/ Labels
Order #10960

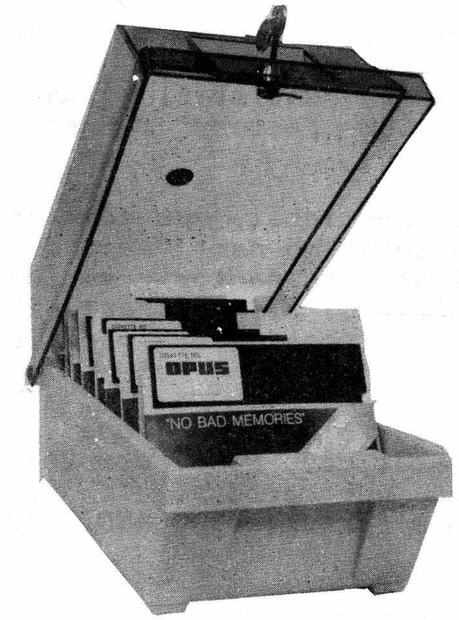




Library Shelf Box, Holds 10 5¼ Disk
Living Hinge Design. Pastel colors
- we'll ship all same in an order
DD10 Order #10989

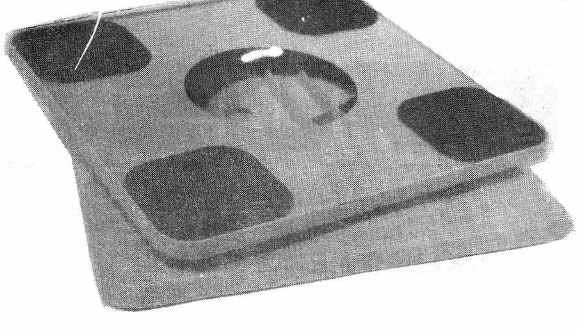


Heavy Duty, Compact, Stackable, Non-Breakable.
One Piece Living Hinge Design. Six Dividers
Factory Installed. Beige body and cover.
D55 Holds 60 5¼ Disks Order #10994
D33 Holds 40 3½ Disks Order #10918



DS100L Holds 100 5¼ Disks
Locking Clear Top, Beige Body
Useable Hinged or Lid Nested Underneath
8 Dividers w/ Labels Order #10975

NOTE: Due to shipping and Packaging costs, our data case products are not available outside N. America!



Extra Heavy Duty Monitor Swivel Stand
Holds Monitors up to 12" on Center Feet
Can be used locked in positon or Adjustable
Beige MS1212 Order #10937

We carry the entire line of OPUS Premium Disks, 3½", 5¼" & 8"
Single and Double sided, Single & Double Density; PLUS, quad
and High Density AT compatable 5¼ disks.

Item	Description	Quanty	Price	Extension
10960	DD120L locking 5¼ case	____	$16.00	______
10975	DS100L locking 5¼ case	____	12.50	______
10922	DD50L same as 10975; holds 50	____	8.00	______
10994	D55 stackable 5¼ case	____	6.45	______
10918	D33 stackable 3½ case	____	5.00	______
10641	DD5 5¼ mailer/tote case	____	1.00	______
10937	MS1212 Monitor Stand	____	16.00	______

Name__________________ Zip Code ________ TOTAL ________

DISK SALE

5.25"	Each	SAVE!		Each	SAVE!
SSDD	$.57	34%	for IBM AT	$1.70	43%
DSDD	.60	35%	DS4D	1.58	15%
			flippies	.61	27%
3.5"			8"		
SSDD	1.50	37%	SSDD	2.60	19%
DSDD	1.70	48%	DSDD	2.80	21%

OPUS QUALITY DISKS!

OPUS disks are without doubt the finest disks made. Mil spec forumulas and exotic materials, promise above spec performance even after 5,000,000 revolutions on each track! Heavy weight jacket and lubricated antistatic liner insure the ultimate in stability. Of course there is a 100% factory warranty backed by CBUG as well.

We are proud of our product. Each disk bears the OPUS corner label; you can be confident you are buying the real article -- no misrepresented no-name seconds. Each disk is inserted in its sleeve; labels + write protect tabs factory sealed in each package. SSDD and DSDD products come 10 in a poly bag; flippies are 25 in a bag. All others are 10 disks in a sealed retail chipboard or plastic package.

This is a special offer by CBUG, Inc. (The Chicago B128 User's Group), one of the world's largest user's groups. Our volume exceeds that of most computer stores and retailers. We pass our good fortune on to you, our members. If you can find OPUS disks in any store, you'll pay several times our price! AND, no one makes a more reliable or longer lasting disk!

------------------------ Jan 10, 1988 ------------------------

Desc	Qty	Stock#	/Pkg	Extension
5.25"				
SSDD	. .	10017	$ 5.70	$
DSDD	. .	10021	6.00	
DSDD Flippy	. .	10111	15.25	
DSQD	. .	10182	15.80	
For IBM AT	. .	10228	17.00	
3.5"				
SSDD	. .	10074	15.00	
DSDD	. .	10088	17.00	
8"				
SSDD	. .	10509	26.00	
DSDD	. .	10547	28.00	
Pkg 25 Tyvek sleeves	. .	10318	1.00	
TOTAL from other side this form				
TOTAL ORDER				
Ill. Residents add 7% sales tax				
COPY TOTAL TO MAIN ORDER FORM				$

PLEASE ENTER NAME & ZIP BELOW IN CASE SEPARATED FROM MAIN ORDER

Name ______________________

Zip Code ______________

OPUS®

"NO BAD MEMORIES"

THRU CBUG, PAY THE LEAST!

SUPER TEACHER - HI CAPACITY w/ UPGRADES **CBUG #77** **#12031**

from: Jon Whatley

This is the latest and a major upgrade on Mr. Whatley's work with educational record keeping. It includes many improvements such as expanded file record fields, improved quiz & review printing programs, etc. It can handle virtually any class load a teacher may be assigned. The titles of programs and files on the disk should give you an indepth idea of what all can be done with this suite. This is formatted in Superbase I so it will operate with either SB I or SB II. The program disk resides in drive 1 and the applicable data disk in drive 0 -- after you've loaded SB of course. It is highly recommended that you run with copies only, keeping your original CBUG disk only to make more operating copies. To that end, the set of 4 includes a package of 10 OPUS super premium SSDD disks. Individual disks can be ordered at the standard library price of $9.00 each -- just specify the .X stock number

The set of 4 together with 10 OPUS blank disks is only $24.00

```
                                          SUPER TEACHER PROGRAMS          stock #12031.1
1 "student records " 01 2c  4  "pd1 daily record" seq  6  "missing work.p"   seq  5  "hposter tips.p"   seq  3  "menu14.p"          seq
12  "report.p"          seq  4  "pd2 daily record" seq  6  "missing work lis" seq  6  "hcalculations.p"  seq  3  "menu15.p"          seq
3   "student list.p"    seq  4  "pd3 daily record" seq  5  "missing work2.p"  seq  7  "missing work3.p"  seq  3  "menu17.p"          seq
4   "grade list.p"      seq  4  "field poster.p"   seq  3  "sp prog rept lis" seq  4  "quiz format.p"    seq  3  "menu18.p"          seq
4   "ranked list.p"     seq  4  "menu2.p"          seq  3  "ID list.p"        seq  3  "key.p"            seq  3  "menu19.p"          seq
7   "daily record.p"    seq  14 "report3.p"        seq  2  "manual entries.p" seq  7  "instructions.p"   seq  3  "menu20.p"          seq
3   "ltrgd.p"           seq  14 "report2.p"        seq  3  "manual entry2.p"  seq  2  "study guide.p"    seq  3  "menu21.p"          seq
3   "absence list.p"    seq  14 "report1.p"        seq  3  "hprocedure.p"     seq  5  "menu6.p"          seq  3  "menu22.p"          seq
3   "absence poster.p"  seq  3  "progress reports" seq  5  "hposter.p"        seq  2  "final.p"          seq  5  "hltrgd.p"          seq
2   "clear files.p"     seq  3  "daily records.p"  seq  3  "features.p"       seq  6  "start II.p"       seq  5  "fexam calc.p"      seq
8   "grade poster.p"    seq  4  "grade posters.p"  seq  5  "hprogress.p"      seq  5  "menu7.p"          seq  6  "start III.p"       seq
9   "grade poster2.p"   seq  3  "lists.p"          seq  6  "habsences.p"      seq  5  "menu8.p"          seq  5  "review format.p"   seq
8   "grade poster3.p"   seq  2  "absence data.p"   seq  6  "hlists.p"         seq  5  "menu9.p"          seq  7  "clear prog"        prg
4   "start.p"           seq  4  "registration2.p"  seq  4  "hdaily records.p" seq  5  "menu10.p"         seq  5  "menu2a.p"          seq
3   "absence record.p"  seq  3  "registrations.p"  seq  5  "hfield poster.p"  seq  5  "menu11.p"         seq  2  "hmanual entries."  seq
5   "registration.p"    seq  5  "hregistration.p"  seq  5  "hmissing work.p"  seq  5  "menu12.p"         seq  6  "manual entry rec"  seq
4   "menu1.p"           seq  3  "absence list2.p"  seq  4  "hmissing work li" seq  5  "menu13.p"         seq  1  "hlist"             seq
                                                                                                             1656 blocks free.

                                          STUDENT RECORDS                 stock #12031.2
1 "student records " 01 2c  12 "period 2"    seq  12 "period 5"    seq  5  "start.p"   seq
1   "STUDENT"records   seq  12 "period 3"    seq  12 "period 6"    seq  1  "hlist"     seq
12  "period 1"         seq  12 "period 4"    seq  12 "period 7"    seq  1951 blocks free.

                                          REVIEW DATA                     stock #12031.3
1 "data bank      " 01 2c  3  "chapter 4"   seq  3  "chapter 9"   seq  3  "chapter 14"  seq  3  "chapter 18"  seq
1   "REVIEW"i          seq  3  "chapter 5"   seq  3  "chapter 10"  seq  3  "chapter 15"  seq  4  "start.p"     seq
3   "chapter 1"        seq  3  "chapter 6"   seq  3  "chapter 11"  seq  1  "REVIEW"ii    seq  1974 blocks free.
3   "chapter 2"        seq  3  "chapter 7"   seq  3  "chapter 12"  seq  3  "chapter 16"  seq
3   "chapter 3"        seq  3  "chapter 8"   seq  3  "chapter 13"  seq  3  "chapter 17"  seq

                                          QUIZ DATA                       stock #12031.4
1 "databank       " 01 2c  3  "chapter 2"   seq  3  "chapter 7"   seq  3  "chapter 12"  seq  3  "chapter 16"  seq
1   "QUIZ"data         seq  3  "chapter 3"   seq  3  "chapter 8"   seq  3  "chapter 13"  seq  3  "chapter 17"  seq
4   "start.p"          seq  3  "chapter 4"   seq  3  "chapter 9"   seq  3  "chapter 14"  seq  3  "chapter 18"  seq
3   "final"            seq  3  "chapter 5"   seq  3  "chapter 10"  seq  1  "QUIZ"data ii seq  1  "hlist"       seq
3   "chapter 1"        seq  3  "chapter 6"   seq  3  "chapter 11"  seq  3  "chapter 15"  seq  1962 blocks free.
```

GOLD COAST INSTRUCTIONAL **CBUG #78** **#12045**

from: Fred Peterson, esq.

This compilation of program and sequential files is submitted for CBUG use as an aid to a better understanding of the abilities of the B-128 machine.

As with all inanimate objects, you get back only what you put in - garbage in, garbage out!

The Compiler is an absolute neophyte in the computer world and these feeble attempts should therefore be reviewed with compassion. It was learned a long, long time ago that it is far better to profit by the works of others whenever possible rather than "re-inventing the wheel" a hundred times or more. As progress in basic programming is being slowly made, it has been found extremely helpful to keep on hand some "archive" disks containing various and sundry programming functions for easy reference when attempting a new project.

Hope these "golden nuggets" will prove of some help; incidentally, it's always an excellent idea to 'list' all programs and trace the workings to see how the end result is accomplished.

```
0 "GOLDCOAST Progs " 01 2c   disk name                                    4  "jandoc87d+"   seq  The remainder of these SS2 SPREADSHEET files
1   "     loader"      prg  Loads 'bootscrn' and 'instscrn'.              5  "jandoc87pc+"  seq   have been designed to give you a modified
8   " bootscrn   .scn" prg  Hello there!                                  5  "sumdoc87r+"   seq   version of the expanded spreadsheet program
8   " instscrn   .scn" prg  Brief introductory instructions.              4  "sumdoc87d+"   seq   for use with a printer accomodating only
6   " menu"            prg  Main menu - choose your program.              4  "sumdoc87pc+"  seq   8-1/2x11 inch paper.  All totalling is done
4   "   directory    " seq  Automated - add more progs to disk and        5  "sumdoc87t+"   seq   automatically, but it is still necessary to
```

			press 'd'.
1	"---------------"	seq	separator
6	"disk contents+"	seq	What you are now looking at.
2	" ------------ "	prg	You can 'load' and 'run' this for brief comments.
1	"----calendar----"	seq	separator
1	"SS2 CALENDAR"	seq	ditto
13	"instructions+"	seq	How to use all the SS2 CALENDAR sequential files
27	"!!--NOTICE--!!!"	seq	User CAUTION notice - please read carefully.
20	"31/sat+"	seq	Beginning of the fundamental SS2 CALENDAR files.
20	"30/sat+"	seq	>
20	"29/sat+"	seq	> saturday
20	"28/sat+"	seq	>
20	"31/fri+"	seq	>
20	"30/fri+"	seq	> friday
20	"29/fri+"	seq	>
20	"28/fri+"	seq	>
20	"31/thu+"	seq	>
20	"30/thu+"	seq	> thursday
20	"29/thu+"	seq	>
20	"28/thu+"	seq	>
20	"31/wed+"	seq	>
20	"30/wed+"	seq	> wednesday
20	"29/wed+"	seq	>
20	"28/wed+"	seq	>
20	"31/tue+"	seq	>
20	"30/tue+"	seq	> tuesday
20	"29/tue+"	seq	>
20	"28/tue+"	seq	>
20	"31/mon+"	seq	>
20	"30/mon+"	seq	> monday
20	"29/mon+"	seq	>
20	"28/mon+"	seq	>
20	"31/sun+"	seq	>
20	"30/sun+"	seq	> sunday
20	"29/sun+"	seq	>
20	"28/sun+"	seq	>
1	" ---------- "	prg	separator
20	"jancal88+"	seq	These are the <u>linked</u> monthly files
20	"febcal88+"	seq	for an entire year.
20	"marcal88+"	seq	
20	"aprcal88+"	seq	They were generated by using the above
20	"maycal88+"	seq	fundamental files and are joined by the 'lk' command.
20	"juncal88+"	seq	
20	"julcal88+"	seq	Now to acquire your very own personalized
20	"augcal88+"	seq	calendar for an entire year - you have only
20	"sepcal88+"	seq	to call month, fill in known holidays, birthdays, anniversaries, other events and
20	"octcal88+"	seq	expected appointments. After saving these
20	"novcal88+"	seq	using the 'F13' key, print with 'Esc-O-L-P'
20	"deccal88+"	seq	(linked file is 'jancal88'). You can 'load'
2	" ++++++++++++ "	prg	and 'run this for brief comments.
1	"--spreadsheet--"	seq	separator
1	"SS2 SPREADSHEET"	seq	ditto
22	"explainlet+"	seq	This detailed letter explains how SS2
4	"jandoc86r+"	seq	SPREADSHEET was conceived and how you can
5	"jandoc86d+"	seq	adapt it to small business or personal needs.
5	"jandoc86pc+"	seq	
5	"sumdoc86r+"	seq	In spreadsheet fashion, it will keep a
4	"sumdoc86d+"	seq	detailed monthly and yearly record of every
5	"sumdoc86pc+"	seq	penny you get and every penny you spend.
5	"sumdoc86t+"	seq	
6	"janbkdocrec86+"	seq	This is a bank reconcilliation document.
4	"jandoc87r+"	seq	
6	"janbkdocrec87+"	seq	manually type monthly totals to the summary spreadsheets.
1	" ############ "	prg	separator
1	"---training---"	prg	ditto
35	"learn"	prg	A math-typing program, ages 5 to 95.
12	"synonym/antonym"	prg	Learn your 'likes' and 'opposites'.
7	"flow chart"	prg	A fundamental of programming.
1	"timing loop"	prg	These small programs, with the exception of
2	"manipulate"	prg	the 'selection menu' (an example mini-menu
2	"draw"	prg	program) are all part of an archive
1	"timer test"	prg	collection for use or illustration, in
5	"arrange alpha"	prg	simple basic programming to aid in an
2	"controlled loop"	prg	avoidance of 're-inventing the wheel'.
2	"code"	prg	For the most part the program name is self-
1	"animation"	prg	explainatory.
3	"ranking"	prg	Enter names & scores - puts in order.
9	"selection menu"	prg	(see above comment)
1	"ftn example1"	prg	This deals with the 'for..to.next' command.
1	"ftn example2"	prg	ditto
1	"ftn example3"	prg	ditto
2	"array1"	prg	
1	"array2"	prg	
1	"read data prog."	prg	
2	"arrange numo"	prg	Puts list of numbers in numerical order.
1	"strange numbers"	prg	Just run.
1	"come together"	prg	ditto
1	"chop decimals"	prg	
3	"animation++"	prg	Load and run this and the other annimation programs
2	"sample array #1"	prg	
2	"sample array #2"	prg	
3	"varied meals"	prg	Plans varied meals for the week.
1	"nested loop"	prg	
1	"tab by variables"	prg	
2	"stop goto loop"	prg	
1	"subscript sample"	prg	
1	"timing test1"	prg	Timer demo.
1	"timing test2"	prg	ditto
1	"timing test3"	prg	ditto
2	"double sub array"	prg	
1	"practice gosub"	prg	
3	"ranking1"	prg	Similar to 'ranking'- handles up to 100 items.
1	"branching"	prg	
1	"a nested loop"	prg	
2	"nested loops"	prg	
3	"margins"	prg	A letter written without 'word processor' help.
1	"animation+"	prg	
2	"animation k"	prg	
2	"alpha/print"	prg	Will alphabetize and print the list.
12	"seq read/print"	prg	A convenience for seeing the seq files without SS.
1	"draw graphics"	prg	
1	"clear screen"	prg	Simple program when 'clear' key won't do the job.
1	"graphics/confitl"	prg	graphics programs - run to see.
3	"fpi logo-d"	prg	ditto
2	"graphics ways-d"	prg	ditto
1	"sine name-d"	prg	ditto
1	"cosine name-d"	prg	ditto
1	"mod name-d"	prg	ditto
2	"mean score"	prg	Delivers the 'mean score' from the bunch.
7	"powers of x"	prg	Simple math program.
1	"squares of x"	prg	ditto
1	"squares of xtoy"	prg	ditto (no, that's not a 'toy', it's x to y)
2	"sines"	prg	ditto
2	"cosines"	prg	ditto
2	"tangent"	prg	ditto
2	"logarithm"	prg	ditto

897 blocks free.

FALL 1987 PRINT FILES **CBUG #79** **#12168**

As usual, the print files of the library and articles and some other material of this issue of THE ESCAPE.

CBUG MISC. **CBUG #M80** **#12064**

Well, let's see what Santa brought for this potpourie:

from: Matthew Goldstein

Finally, that often requested program -- a golfer's handicap and scoring program!

'Mastermenu v2.0' allows you to rearrange the menu screens by adding, removing, changing the name of, and moving files, and adding dummy header titles using the second set of function keys. 'alarm interval' is a m/l routine that sets the alarm clock and detours the interrupt vector into code that monitors the alarm, ringing the bell and resetting the alarm when it goes off. 'BAM' is a simple two level nueral network that learns the input data patterns and recalls the data learned even if the match data is incomplete or inaccurate. 'data cards' is written in JCL extended BASIC and cannot be run without the JCL Workshop disk.

```
65   " Mastermenu v2.0" prg  many more features: add, remove        4    "vocabulary1..+sa" prg    etc.  program needs a few
6    "Masterdirectory"  rel   newname, move, header, print          3    "vocabulary1..+sb" prg     more adjustments.
1    "m/l print screen" prg    screen, etc.                         2    "vocabulary1..+sc" prg      vocabulary data base inc.
13   "alarm interval"   prg  recurring alarm at preset time         7    "BAM"              prg  bi-directional associative memory
1    "+alarm int.f0400" prg   interval untill you reset it          13   "data cards.ins"   seq
83   "golfhandicapSSGA" prg  maintains records of golfer's          33   "Master2.0.ins"    seq
6    "golfname.dat"     seq   scores and handicaps                  11   "golfhandicap.ins" seq
7    "disinterpreterb"  prg  this version works 100% on B           9    "computed goto"    seq
53   "data cards.jcl"   prg  index card type data base with         24   "input"            seq
594  "vocabulary1..+fi" rel   1 to 29 'sides' per card.                   936 blocks used
1    "vocabulary1..+fp" seq    sort, move, delete, edit,
```

From our good friends at ICPUG (Independent Commodore Products User Group), Jim Kennedy, President. They have provided us with disk files of their ongoing series on Superscript and Superbase from their perspective of C-64 and C-128 machines. While there is a great deal of similarity, there are some areas not applicable to the B-128. We are carrying these articles edited for B-128 interest in the article section of the current issue (Fall 1987). On this disk are the full unedited files as printed by our friends on London.

```
1 "icpug          " jk 2c    57   "SuperCornerMay"   seq   85   "SB-DirctDataRead" seq   4    "NovDecNote"      seq
77  "SuperCrnrMarApr"  seq   32   "SuperCrnrJul/Aug" seq   9    "SuperCrnrSepOct"  seq     266 blocks used
```

From: Dr. Ronald Servos

Interesting little program for generating a perpetual schedule for a rotating staff. In the example, Dr. Servos is scheduling several doctors in his practice about a 7 day schedule. Prints out month by month schedules. Obviously useful for all sorts of things from firemen to study hall supervisors, lifeguards, etc.

```
1  "schedule " up 2c    1  "lll"          prg    8  "logo1"        prg    14 "schedv1087"  prg    3  "setup-sched"  seq      26 blocks used
```

From: Dr. Harold Querner

A stand alone database (i.e. does not require Superbase or any other underlying program). In this case the program is designed to keep track of sports injuries, certain details and costs. Operates with a data disk in drive 1 (you supply blank formatted disk). Good example to learn from, adapt, etc. Has reporting, editing and other necessary features. Menu driven. No instructions needed. Load the first program in set to run.

```
1 "sportsmed/backup" MG 2c    66   "sportII"          prg   6    "initializer"      prg   1    "PROGRAM"         seq     73 blocks used
                                                                                                                 751 blocks free entire disk
```

◆◆◆◆◆◆◆◆

DAVID GREEN UPDATE **CBUG #M81** **#12079**

From: David Green

<<This is an upgrade etc. of materials submitted by Mr. Green which appeared on M54. If you are acquiring this as a replacement disk, send in the M54 label and extend the pricing at 50% off.>>

The following is a short explanation of some of the programs on the disk. I have written most of these programs myself; the exceptions are the six text adventures and the renumbering routine. The programs listed are something of a miscellaneous collection; there are a quite a number of games and several utilities as well. They are separated on the disk directory by null programs.

"run me.dg" is a program to read Superscript files like this one without accessing Superscript.
"adventures" is a loader program for the eight adventure games.
"3d tictactoe" is a game of 4x4x4 tictactoe; you can play against the computer or another human player.
"roadrace.dg" is a simple roadrace game; you control a car speeding down a road full of potholes. There is a finish line; its distance from START is determined by the skill level you choose.
"snakes.dg" is a game where you control a snake and try to avoid obstacles.

I got the following adventure programs from a games disk put out by BUG; I improved them considerably, adding upper/lower case and correcting errors.

"haunted.adv" is an adventure game in which you must explore a haunted house and bring out valuables and treasures.
"pirate.adv" is a terribly slow adventure where you labor to build a pirate ship and sail the seas to Treasure Island.
"cavern.adv" is a rather uninspired offshoot of the original Adventure.
"sorcerer.adv" also has roots in ideas from the first Adventure; however, it is original to a large extent and contains a few humorous touches.
"dogstar.adv" is a Star Wars adventure in which you must rescue Princess Leia.
"wisp.adv" is my personal favorite; it is fast and clever, though actually more of a spoof on adventures than a real adventure game.
"colossal.adv" is an abridged (but still enormous) version of the original Adventure program, written back in the dark ages of computing when B-128's weren't even thought of. Strangely, it's still one of the best in the collection -- difficult, though; the abridging process seems to have removed all reference to the necessary magic word "blast" . . .
"collab" is a three-dimensional maze game. It has two floors, each one being a 25x25 maze grid. There are a number of pits, elevators, teleport squares, objects, monsters, and other surprises hidden in various places. Making a map is advisable, but it will still take time to figure out how to escape.
"abrlab" is an abridged version of the above; everything in the larger maze is a 15x15 grid). This makes it less of a challenge but easier to explore a 2nd map.

"bigmaze.dg" is a simple maze game on a large grid (99x99). This program is not a very interesting game by itself.
"mouse.dg" is a program which allows you to create mazes and run simulated
"mice" through them. After the first run-through, you can have the mouse take the fastest path to the exit.
"taxman.dg" is an "educational game" involving factors.
"mazegen.redraw" is a maze generator that quickly and repeatedly redraws mazes. For all the "mazegen" programs, it is possible to control the direction the generator takes. Use (A) for up, (Z) for down, and < and > for left + right.
"mazegen2" draws mazes on a more condensed grid.
"mazegen3" is the same as "mazegen2", but with a different type of graphics.
"chase1" a rather abstract game program. The player is represented by a solid graphics block. He must maneuver (using (A), (Z), <, and >) and try to avoid monsters. There are obstacles -- land mines -- scattered around the field, and if either a monster or the player runs into one of these, it explodes. The object is to get rid of all the monsters by making them run into mines.
"chase2" is a version of "chase" where the obstacles do not explode; the object is to grab a number of '$$'s which are scattered around. If two monsters run into each other, they will explode. Again, the player must avoid the monsters.
"chase3" is yet another version; here, the player can push the obstacles around. The monsters can be killed by pushing objects into them, but every time this is done, the player must grab a '$$'. The object of this game is to kill all the monsters and also grab all the '$$'s.
"renumber.dg" is a utility which will renumber BASIC programs.
"hres editor.dg" allows you to draw pictures or designs and then print them out to a CBM 4023 printer. An entire screen comes out about two inches square.
"screen editor.dg" lets you design screen layouts for use in other programs.
"drawprint.dg" lets you draw on the screen and print the results on a printer in high resolution. This is a simpler version of the highres editor; it h
"diagprint.dg" prints out on the printer an N-sided polygon with all diagonals connected. This is a demonstration of my high-res SET/RESET/TEST/LINE drawing routines; you can remove the demo program and use the subroutines.
"graphics.dg" is a set of subroutines for onscreen, 80x50 SET, RESET, AND TEST.
"graphics2.dg" is the same as "graphics.dg", but with a matrix of 160x50.

1	"cbug offering v2"	o2 2c	85	"haunted.adv"	prg	147	"abrlab"	prg	7	"chase2"	prg	3	"graphics.dg"	prg
2	"run me.dg"	prg	72	"pirate.adv"	prg	6	"bigmaze.dg"	prg	7	"chase3"	prg	3	"graphics2.dg"	prg
3	"adventures"	prg	67	"cavern.adv"	prg	20	"mouse.dg"	prg	1	"---utilities---"	prg	9	"adirectory.dg"	seq
24	"readme.ss"	seq	60	"sorcerer.adv"	prg	11	"taxman.dg"	prg	6	"renumber.dg"	prg	1	"from.dg"	seq
1	"-----games-----"	prg	64	"dogstar.adv"	prg	4	"mazegen.redraw"	prg	18	"hres editor.dg"	prg	842 blocks free.		
36	"3d tictactoe"	prg	119	"wisp.adv"	prg	4	"mazegen2"	prg	5	"screen editor.dg"	prg			
13	"roadrace.dg"	prg	217	"colossal.adv"	prg	4	"mazegen3"	prg	7	"drawprint.dg"	prg			
16	"snakes.dg"	prg	158	"collab"	prg	6	"chase1"	prg	4	"diagprint.dg"	prg			

◆◆◆◆◆◆◆◆

CP/M 86 INFO & PROGRAMS #4 **CBUG #PR11** **#12083**

From: Col. John Wright

This and the next three disks are material for use with the 8088 co-processor and is intended for advanced programmers only.

TO NORMAN

Norm, here is disk four of CP/M-86 files that I have found. All these have come from a "Public Domain" archive file found on the Defense Data Net.

The last file on the disk is a listing of all the remaining CP/M 86 files that I can obtain form this one directory. As you can see, it is very lengthy. There are many other files for CP/M 80 and other CP/M operating systems. In fact all the files would probably fill one or two disks, and that would just be the file descriptions. If you want the CP/M 80 files as well (I have already provided a 80-86 emulator), let me know. Next disk will be coming shortly, I will be out of town this month, but will work while I am away.

I have not gotten any feed-back on the usefullness of the files I have already sent. Are they usefull?

Another question, will the 8088 cpu be available to us with the Protecto package so we can upgrade our systems? <<A qualified yes. All boards available are from N.W. Music (see ads). Some members here and in Europe are dsigning and planning to fabricate coprocessor boards -- most likely using the V20 chip>> From the description in the "GREY" Users Guide, it looks like the 8088 will also run MS DOS. <<The version of MS-DOS implemented is quite obsolete. Several members are dilligently trying to upgrade it to a current version as well as fix the operating system bugs. Prognosis is fairly good>> If that is really the case, then I have access to an even larger collection of these files.

Have fun wth this one, let me know if you have any special requirements or needs that I can help with.

0	"cp/m-86 "	04 2c		16	"flip44.aq6"	prg	Flips modem mode, CP/M 86 version
6	"to-norm.sii.4"	seq		44	"help15.aq6"	prg	Displays help files, CP/M 86 version
1	"directory.sii.4"	seq		11	"help15.cmd"	prg	/
10	"guide.sii.4"	seq		87	"pause.cmd"	prg	Pause during SUBMIT CP/M 86 version for
15	"addlf.cq"	prg	Adds LF after CR in text files	7	"pause86.cq"	prg	DRI C
80	"chat.cmd"	prg	RCPM CHAT utillity rewritten in	27	"tag3.aq6"	prg	Set attributes on CP/M 86 files for
12	"chat86.cq"	prg	DRI C	5	"tag3.cmd"	prg	RCPMs
6	"ctype.hq"	prg	RT-11 TO CP/M 86 disk transfer for DRI C	5	"u.cmd"	prg	Changes drive/user on CP/M 86 MP/M 86
4	"errno.h"	seq	/	27	"u-ccpm.aq6"	prg	/
7	"portab.hq"	prg	/	8	"usq.com"	prg	Unsqueezes squeezed files
23	"rt11.cq"	prg	/	15	"crc.cmd"	prg	CP/M 86 checksum program
102	"rt11.cqd"	prg	/	11	"usq.cmd"	prg	CP/M unsqueezer
21	"rt11.dqc"	prg	/	74	"lu86.cmd"	prg	CP/M 86 library utility
6	"rt11.h"	seq	/	77	"freebase.lbr"	prg	Data base program
1	"rt11.inp"	seq	/	15	"mem.lbr"	prg	Creates RAM disk
33	"rtfile.cq"	prg	/	333	"res86.lbr"	prg	CP/M 86 RESOURCE disassembler
13	"rtmisc.cq"	prg	/	45	"splitter.lbr"	prg	Segments files for copying
16	"rtmisc1.aq6"	prg	/	54	"tlabel86.lbr"	prg	CP/M 86 Label printer
5	"setjmp.hq"	prg	/	21	"wysefk21.aqm"	prg	
5	"stdio.h"	seq	/	117	"files.sii.1"	seq	CP/M 86 files yet to be downloaded
				687 blocks free.			

CP/M 86 INFO & PROGRAMS #5 CBUG #PR12 #12098

From: Col. John Wright

See above #PR12

Here is disk five of CP/M-86 files that I have found. All these have come from a "Public Domain" archive file found on the Defense Data Net.

I think that I have solved the seq file problem. It appears that when I use TELKLETERM80 to download, the "ALTERNATE" character set is sent. Must be an interface problem with DDN and TELE. The pgm files seem to work fine. In this disk I have solved the problem in two ways:

1. Used BEELINE to download into buffer then saved Buffer.
2. Used Kermit on my C-64 to download multifiles, then used Tele. to download form the C-64 using Punter protocole. The later seems faster even at 300 baud.

The SEQ files downloaded directly from DDN can be read using "SEQ LIST CBM/ASC" pgm found on Kernaghans Utility disk. I think the problem lies with the difference between ASCII files and CBM ASCII files.

At any rate, this disk contains a series of files downloaded from a library called "GASP (General Activity Simulation Program). I don't know exactly what it is supposed to do, but it is a very long library. In fact the remaining files are on disk #6. All files were provided by the Japan User Group, so your guess is as good as mine. Documentation is provided, so maybe someone can figure out what it does.

You will notice that there are several files with the "FOR" extension. I believe these are listings of subroutines written in FORTRAN. There are also assembyl files available (F86 and R86 extensions). These are supposedly executable program files.

Have fun with this one, and let me know what you find out.

0	"cp/m-86 "	05 2c	
7	"to-norm.sii.4"	seq	
8	"directory.sii.4"	seq	This file
11	"guide.sii.4"	seq	Updated guide for disk 5
131	"gaslib.r86"	prg	Relocatable GASP module for CP/M-86
2	"gasp11.dat.1"	seq	Data file for USER1.CMD
2	"gasp1.dat.1"	seq	"
2	"gasp12.dat.1"	seq	"
2	"gasp2.dat.1"	seq	Data file for USER2.CMD
4	"gasp3.dat.1"	seq	Data file for USER3.CMD
2	"gasp4.dat.1"	seq	Data file for USER4.CMD
158	"user1.cmd.1"	prg	Executable code of USER1.F86
162	"user2.cmd.1"	prg	Executable code of USER2.F86
161	"user3.cmd.1"	prg	Executable code of USER3.F86
158	"user4.cmd.1"	prg	Executable code of USER4.F86
15	"arrvl3.for.1"	seq	Simulating arrival event for USER3.F86
12	"arrvl4.for.1"	seq	Simulating arrival event for USER4.F86
7	"colct.for.1"	seq	Routine to Collect the sample data
23	"datan.for.1"	seq	Initialization routine of GASP variables
5	"drand.for.1"	seq	Generation of uniform randon numbers
9	"endsm3.for.1"	seq	END of Simulation routine for USER3.F86
7	"endsm4.for.1"	seq	END of Simulation routine for USER4.F86
11	"endsv3.for.1"	seq	END of Service routine for USER3.F86
13	"endsv4.for.1"	seq	END of Service routine for USER4.F86
8	"error.for.1"	seq	Error reporting routine
8	"evnts3.for.1"	seq	Events control routine for USER3.F86
8	"evnts4.for.1"	seq	Events control routine for USER4.F86
9	"exa1.for.1"	seq	Main of USER1.F86 and USER2.F86
9	"exa3.for.1"	seq	Main of USER3.F86
11	"exa4.for.1"	seq	Main of USER4.F86
8	"filem.for.1"	seq	To store the vector in file
150	"gaslib.f86.1"	prg	GASP subroutine library for CP/M-86
14	"gasp.for.1"	seq	Master control routine
2	"gasp1a.dat.1"	seq	Data file for USER1.F86 (Originally GASP1.DAT.
2	"gasp2a.dat.1"	seq	Data file for USER2.F86 (Same as above)
4	"gasp3a.dat.1"	seq	Data file for USER3.F86 (Same)
2	"gasp4a.dat.1"	seq	Data file for USER4.F86 (Same)
8	"histo.for.1"	seq	Histogram to tabulate data
9	"montr.for.1"	seq	Data monitoring routine for USER1.F86
10	"montr2.for.1"	seq	" USER2.F86
10	"montr3.for.1"	seq	" USER3.F86
11	"montr4.for.1"	seq	" USER4.F86
11	"otput3.for.1"	seq	Output routine for USER3.F86
8	"otput4.for.1"	seq	" USER4.F86
12	"prntq.for.1"	seq	Print routine
4	"randu.for.1"	seq	Generation of uniform random number
8	"rmove.for.1"	seq	Remove routine
31	"set.for.1"	seq	This routine is the heart of information
13	"sttup.for.1"	seq	Subroutine to reset simulation
14	"sumry.for.1"	seq	Routine for reporting results
7	"tmst.for.1"	seq	Computation of statistics
5	"tstrnd.for.1"	seq	Test program for DRAND.FOR
58	"user1.f86.1"	seq	Sample program No. 1 (Queuing) for F86
69	"user2.f86.1"	seq	Sample program No. 2 (Queuing) for F86
72	"user3.f86.1"	seq	Sample program No. 3 (Queuing) for F86
54	"user4.f86.1"	seq	Sample program No. 4 (Queuing) for F86
52	"user4.f86.2"	seq	" (Downloaded w/Buffer & Beeline)
84	"gasp.doc.1"	seq	Documentation of GASP package
10	"exa11x.f86.1"	seq	Main of Information System
9	"anser.for.1"	seq	Subroutine for EXA11
10	"arrvlx.for.1"	seq	/
8	"endsvx.for.1"	seq	/
8	"evntsx.for.1"	seq	/
15	"otputx.for.1"	seq	/
6	"rqest.for.1"	seq	/
12	"scan.for.1"	seq	/
174	"gaslibx.f86.1"	seq	Extended GASP subroutine library for F86
82	"user11x.f86"	seq	Sample program of Information System for F86
2	"gasp10.dat"	seq	Data file for USER1.CMD
2	"gasp11.dat"	seq	/
2	"gasp12.dat"	seq	/
0 blocks free.			

◆◆◆◆◆◆◆

CP/M 86 INFO & PROGRAMS #6 CBUG #PR13 #12101

From: Col. John Wright

Norm, here is disk six of CP/M-86 files that I have found. All these have come from a "Public Domain" archive file found on the Defense Data Net.

The SEQ files on this disk can be read using "SEQ LIST CBM/ASC" pgm found on Kernaghans Utility disk.

This disk contains the continuation of the series of files downloaded from GASP, the beginnings of which are on DISK #5.

All files starting with "CLCIB86.LBR" are utility files for general use. I have included the DELBR and USQ pgms again as backups. Also included a Library Utility (LU86.CMD). This is supposed to be a useful pgm.

```
0 "cp/m-86        " 06 2c
3    "to-norm.sii.6"    seq
9    "directory.sii.6"  seq    This file
14   "guide.sii.6"      seq
2    "gasp13.dat"       seq    Data file for USER1.CMD
2    "gasp14.dat.1"     seq              /
2    "gasp15.dat.1"     seq             /
2    "gasp16.dat.1"     seq            /
2    "gasp20.dat.1"     seq    Data file for USER2.CMD
2    "gasp21.dat.1"     seq               /
2    "gasp22.dat.1"     seq              /
2    "gasp23.dat.1"     seq             /
2    "gasp24.dat.1"     seq            /
2    "gasp25.dat.1"     seq           /
2    "gasp26.dat.1"     seq          /
2    "gasp30.dat.1"     seq    Data file for USER3.CMD
2    "gasp31.dat.1"     seq               /
2    "gasp32.dat.1"     seq              /
2    "gasp33.dat.1"     seq             /
2    "gasp34.dat.1"     seq            /
2    "gasp35.dat.1"     seq           /
2    "gasp36.dat.1"     seq          /
```

```
289  "gaslibx.r86.1"    prg    Relocatable module file of GASLIBX.F86
10   "clcib86.lbr.1"    prg    CB86 commadn line interpertar
15   "crc.cmd.2"        prg    CRC checker in CP/M-86
12   "crc.com.2"        prg          /
76   "crc-64.aq6.1"     prg          /
140  "crcbuild.lbr.1"   prg    Builds catalog files- CP/M 80 and 86
52   "delbr.com.2"      prg    De-Library routine in CP/M-86
74   "lu86.cmd.1"       prg    Library Utility in CP/M-86
54   "put.aq6.1"        prg    Copies files between user areas
12   "put.cmd.1"        prg         /
68   "sq183.cmd.1"      prg    CP/M-86 file squeezer
108  "tab86.lbr.1"      prg    Creates tabs in code
103  "untab86.cmd.1"    prg    Removes tabs from code
11   "usq.cmd.2"        prg    CP/M-86 file unsqueezer
8    "usq.com.2"        prg    /
781  "86kermit.lbr.1"   prg    CP/M-86 version of Kermit terminal program
62   "find20b.a86.1"    seq    Locates any string in any part of a file
10   "find20b.cmd.1"    prg    /
59   "print10b.a86.1"   seq    Pages printer output with optional header
7    "print10b.cmd.1"   prg    /
39 blocks free.
```

CP/M 86 INFO & PROGRAMS #7 **CBUG #PR14** **#12116**

From: Col. John Wright

The SEQ files on this disk can be read using "SEQ LIST CBM/ASC" pgm found on Kernaghans Utility disk.

```
***************************************************************************
*                          CP/M-86    DISK #7                             *
***************************************************************************

1 "cp/m-86        " 07 2c

------  ----               ----  ----    ---    -----------
2    "to-norm.sii.7"      seq    1K
4    "directory.sii.7"    seq    1K
10   "guide.sii.7"        seq    2K
67   "8086.a86.1"         seq   17K    26 B3  8086 monitor
212  "du-v75a.a86.1"      seq   53K    27 BA  DU for CP/M 86
33   "du-v75a.cmd.1"      prg    8K    FD 83   /
26   "du-v75a.doc.1"      seq    7K    EC BE  /
12   "unera31.cmd"        prg    3K    C7 5C  Unerase for CP/M 86
76   "unera31.a86.1"      seq   19K    45 1D  /
15   "printer.doc.1"      seq    4K    B1 24  DOC file for following printer
                                              Programs
36   "epsprtr.a86.1"      seq    9K    65 1F  Sets Epson printer in CP/M 80
40   "epsprtr.asm.1"      prg   10K    BE 52   and CP/M 86
8    "epsprtr.cmd.1"      prg    2K    0F 0A   /
6    "epsprtr.com.1"      prg    2K    04 86  /
20   "oki-92.a86.1"       seq    5K    16 0F  Sets Okidata 92 printer in
```

```
------  ----               ----  ----    ---    -----------
5    "oki-92.cmd.1"       prg    1K    E1 4F   CPM/80 and CP/M 86
37   "setoki.asm.1"       seq    9K    F5 04   /
6    "setoki.com.1"       prg    2K    BC 7B  /
236  "sd-48b.a86.1"       seq   59K    A2 BB  Sorted directory utilty with
19   "sd-48b.cmd.1"       prg    5K    F9 5C   help file.
370  "vfiler11.a86.1"     seq   92K    2F 20  Rich Conn's screen oriented
35   "vfiler11.cmd.1"     prg    9K    C1 23   file manipulation utility
                                               translated for CP/M 86
82   "whatsnew.a86.1"     seq   21K    B8 75  Tells what has been added to
10   "whatsnew.cmd.1"     prg    3K    DD 0F   or deleted from disk
460  "c86.lbr.1"          prg  114K    32 07  8086 version of Small C
4    "c86.doc.1"          seq    1K    FD D1  /
78   "archive.a86.1"      seq   20K    94 5E  CP/M 86 translation of Archive
76   "archive.cmd.1"      prg   19K    48 52  /
32   "file-ext.a86.1"     seq    8K    14 D5  CP/M 86 file-extension program
9    "file-ext.cmd.1"     prg    3K    28 42  /
21 blocks free.
```

WACKS ASSORTMENT **CBUG #PR15** **#12050**

from: David Wacks

This was mentioned in the Summer 87 Library Lead editorial but omitted from the library. Since there was no commentary from Mr. Wacks with or on the disk, we know little about it. It has a very different menu program which employes windowing. You load it from "run-me.p" or from Shift Run. The directory program automatically alphabetizes the directory. Comments welcome!

```
1 "david wacks     " w1 2c    35 "yatzee-pro.g"      prg   20 "computer conv.p"  prg   24  "othello.g"          prg   8   "blackbook.s"     prg
23 "disk control"    prg     8  "yatzee.scrng"      prg   16 "camel.g"          prg   21  "cannon.g"           prg   38  "black book.p"    prg
11 "prog-lister.u"   prg     8  "yatzee.instr"      prg   8  "star plot.p"      prg   4   "hex/dec conv.p"     prg   13  "warfish ii.g"    prg
56 "fortune wheel.g" prg     8  "yatzee.disp"       prg   12 "warfish.g"        prg   462 "phone list.f"       rel   2   "ml input.p"      prg
29 "hangman.g"       prg     15 "chg un add.u"      prg   9  "poster maker.p"   prg   9   "xmission.p"         prg   1   "ml input.f"      prg
19 "racetrack.g"     prg     20 "cards.g"           prg   5  "rocket-print.p"   prg   8   "castle.drw.s"       prg   8   "run-me.s"        prg
9  "spelling.p"      prg     4  "b.o.c. symbol.p"   prg   10 "cat-print.p"      prg   8   "time.trap.s"        prg   1   "run-me.p"        prg
32 "arithmetic.p"    prg     13 "led-zeppelin.p"    prg   16 "story.ss2"        seq   2   "coming.attr.p"      prg   511 blocks free.
9  "digital clock.p" prg     24 "black-sabbath.p"   prg   51 "miser.g"          prg   462 "phone list.f.bak"   rel
```

<<U.S. Law is different. Under the fair use doctrine of the U.S. Copyright law, a user/owner is entitled to make as many copies of a program (or other intellectual property as desired SO LONG AS IT IS NOT SHARED OR PASSED ON (to anyone not similarily entitled to the same materials.) In a business, for example to buy one word processor and then distribute 10 copies amongst the 10 secretaries is un-lawful. Likewise to share a copy with your brother is illegal. But generally, to share the one original copy in the absence of another in use copy (your backup is always in the vault) is OK. After that it gets rather complex. US and Canadian law last I heard as regards PC usage are quite similar. Europe is, however another matter.>>

If it is ever necessary to change the SFD to 8050 format use the following program:

```
10 INPUT "DEVICE NUMBER",d :REM DEFAULT IS 8
20 OPEN15,D,15
30 PRINT#15,"M-W"+CHR$(172)+CHR$(16)+CHR$(1)+CHR$(1)
40 PRINT#15,"M-W"+CHR$(195)+CHR$(16)+CHR$(1)+CHR$(0)
50 PRINT#15,"U9":CLOSE15
```

The drive will then remain in 8050 format until reset

That is enough for this issue. My fingers are about to rebel against any more keyboard activities this month. Hope to have some more from PSL in the next issue.

SUPER* CORNER, May 87

edited by Jim Kennedy

8 Bit Superbase Programming Tips

When editing a Superbase program you can get back to the line before by pressing the backarrow key. This will save lots of cursor key activity especially on long program lines. Another tip is when you want to get to a line in the middle of a large program clear the screen by SHIFT CLR/HOME and then type PROG 2340, for example, to call up program lines starting with 2340 at the top of the screen. These two tips save lots of program development time.

Superbase: The Book - Errata Sheet

Below are some errors to correct if you have this really good book.

1. pg 30-31 missing words:
...output or other processing. You can copy, rename, or delete lists.
Memos
These are created with the memo option. They...

2. pg 154 line 20: File 'data'

3. pg 149 line 60: insert after file 'old' and before
select n : select a$:

4. pg 101-102 missing words:
...update the invoice file to make the new invoices show a status...

5. pg 136 bottom: 10 PRINT CHR$ (27);CHR$(14)... not
10 PRINT = CHR$...

6. pg 102 & 103: Line 80 appears twice.

7. pg 23 add following:
Illegal characters list = # - @ < > / & '' [ ] ( ) for both field names and contents.

8. pg 29: Several lines are repeated on pg 30.

9. pg 138: Program has CR$ = CHR$(13) missing.

10. pg 27 & 28: Renumber second guideline #3 and all subsequent ones.

Dates as Key Fields by Month and Day

Sometimes one wishes to store records by date but irrespective of the year for things such as birthdays. For example, one file with a key field of [lastname] could have birthdates in the usual Superbase format of 13Jun87. Suppose one wants to make another file with the records stored (keyed) in the order 12 Jan 45, 14 Jan 36, 23 Jan 65 etc and, of course, we wish the key to be unique to avoid the problems which duplicate keys lead to. Also assume the birthdate is a field titled [birthdate]. We must convert this date, which is stored as a number, to the new format and store various other bits of information in the new file.

```
10 file ''lnamelist''
20 select f:goto 100
30 ln$=[lastname]
100 x = 65:rem set for ASCII no for letter A
120 dt = [birthdate]:rem read in birthdate as stored in
      numerical form
130 convert dt,dt$:rem convert the date to string form
140 da$ = left$(dt$,2): rem extract day of month numbers
150 mo$ = mid$(dt$,3,3): rem extract month abrev
160 date dt$,m:rem find out which month by number ie xx=m
170 if m<10 then mn$=''0''+right$(str$(m),1):else mn$ =
      right$(str$(m),2)
180 yy$ = right$(dt$,2):yr$ = ''19''+yy$:rem extract year
      abrev number
190 file ''bdaylist'':rem call new file
200 ky$ = mn$+da$+yy$+chr$(x):rem build key string eg
      012254a for 22 Jan
210 select ky$: rem see if key already exists
220 pmat goto 300: rem if partial match then ok to store
230 nmat goto 300;rem if no match then ok to store
240 x=x+1:goto 200:if match then increment to 012254b and
      try again
300 clear:rem create blank record
310 calc [lname]=ln$;[day]=da$;[month]=mo$;[year]=
      yr$;[key]=ky$
320 store rem store the new record
330 file ''lnamelist'':select ln$:rem go back to same
      record in old file
340 select n:eof menu:rem move to next record
350 goto 30:rem repeat process for next record
```

Super* Corner, JunAug87

edited by Jim Kennedy

Superbase Key Field Size

This next item came from Jack Cohen when he experienced it whilst running ICPUG membership files. Bear in mind that our membership is continuing to grow so we have to keep finding ways of cramming more data onto the 1 megabyte floppies in our 8250 drives which store all your names, addresses, subs, membership numbers, etc.

Keep key fields as small as practicable especially when dealing with large databases. The larger the key field in a database the larger the B tree file is which contains the keys to the database. And the larger the B tree file the less room on the disk for database files.

Super* Corner, SepOct87

edited by Jim Kennedy

Inserting fields in Superbase 64

From Jonathan Barber comes some suggestions about inserting fields into existing Superbase record formats. Bear in mind that simply adding a new field in the middle of an existing format will result in all the contents of the fields from the new one onward shifting upwards and messing things up in royal fashion.

There are various methods of doing this, two of which are:

1. Put an extra field at the end of the format, rename fields as necessary, and shuffle the data into the right fields using 'batch'. This is tedious if you have a database which is large either in terms of number of records or of fields to be shifted.
2. Use the database to database transfer procedure. This can take a long time however.

There is, however, another method which Jonathan has used with success with Superbase 64 version 2.03. With a file of names, addresses and other data we wanted to insert a [telephone] field after [postcode] and before [type] which was the first of the remaining data fields.
First, a backup of the disk was made in case of accidents, and then the following was entered:

```
batch all [postcode] = [postcode] + chr$(96)
```

This writes an an additional field separator character into the [postcode] field of each record in the file, so that Superbase will read back an extra (blank) field between [postcode] and [type] in each record. The final stage is to format the file definition again, this time with the [telephone] field in its right place. It is important to do this immediately after the previous stage to prevent loss or corruption of data.

◆◆◆◆◆◆◆◆◆

MORE ON SUPERBASE

By: Bruce Faierson

From the number of calls that we get daily, it appears that there are a lot of people out there that do not use Superbase. This is amazing because it is a very powerful tool for development of either simple database activities or powerful applications such as accounting systems. It does require some study to use this program effectively. We will address the topics of using and understanding what a database is and eventually enter the realm of programming applications.

Superbase has been designed to operate at three different levels. The first level is called the menu level and is accessible by even the novice. The menu level gives you all of the most basic commands and the flexibility to do everything from record input to report generating. The menu commands are accessed by typing either the number or first letter of the command. If you press return while in menu 1 you will access menu 2.

The second level of access is at the command level. Up to two lines, of 80 characters each, are available at the command level. Any Superbase command or group of commands separated by colons are permissible. By using the command line, several instructions can be carried on without further intervention.

The third level of Superbase is the programming level. This level is the most exciting and frightening for the first time programmer and user of Superbase. The programmer can do everything from automate simple sorts and searches to develop sophisticated applications like accounting systems. Unlike most other languages, Superbase contains its own editor and error checking. The user can also program the screen display and use the more than forty additional commands that Superbase adds to the Basic language.

For those of you that may be a little apprehensive about working with Superbase, I will relate my own personal experience. When I bought my first B-128 system in late 1984, Superbase was one of the first programs that I purchased. Though I knew nothing of programming, Basic or otherwise, the information presented by Protecto on Superbase was fascinating.

I remember opening the manual and browsing through it. The language used was foreign to me at the time and when I got to the programming section I was absolutely petrified. I was using a computer to run accounting for my business and was not prepared to take on the role of a programmer to. It took me over a year to get the courage to investigate the mysterious art of Superbase programming. It was well worth the effort!

A little history of Superbase follows. The first Superbase was developed for the Pet 8096 and ported over to the B-128 and C-64. As with any product, Superbase went through many modifications and improvements. As we all know, no product is absolutely bug free and Precision Software did their best to upgrade Superbase. We have not heard of any problems with the current versions. If you are going to use Superbase it is highly recommended that you use the current version. Information on acquiring the most recent version of Superbase II is located at the end of this article.

A database is a collection of related information. A card file of phone numbers or a inventory list is a database so to speak. The advantage of a computer database system is that data is almost accessible instantly. If you had a manual inventory system you would have to search sequentially through your list to find the right item or code number. A good database program will search for your data in a random manner locating it much faster than you could ever do it yourself.

There are two types of databases that are important to our discussion. The first is called a relational database and the second is called semi-relational. A relational database is usually defined as one in which at least two related files, using selected fields, can be merged to form new files or data can be accessed from one file by relating a it to a field in your current file. While Superbase is not a relational database it has the capability of accessing information in one file by using the key field in the current file. This is very important when developing applications and is usually called semi-relational.

A computer database breaks down into several components. In Superbase, a database consists of between one and fifteen related files. I did say related, though you could have unrelated files within a Superbase database. It is not recommended that you have unrelated files in your database unless you are working on a hard disk. This special case will be covered later.

A file consists of a group of records that are related to each other by structure. The structure is determined by the developer of the database. As an example, the record format for an address database might contain locations for first name, last name, address, city, state, zip, phone number and other miscellaneous information. It is not imperative that all these locations be filled with data. It is of essence that the correct type of information is placed in each area.

A record can be broken down into a number of locations called fields. The field is the smallest component in the database. Every field in your record must be labeled and defined as a particular data type. A field in an address database might be labeled <lname> for last name. Superbase has eight types of data fields definitions. These are the key, text, numeric, forced, date, constant, result and calendar fields. Some of these will be defined and discussed later in this article.

In summary you have to define a database that will control your files. At least one file must be created to hold your records and a structure of fields has to be set up to make a record format. These are the many dependent elements in a database.

Superbase automatically sets up your basic data disk and discusses the start up procedures in Tutorial Level

One in the Superbase manual. Since it is covered so well there, it would be redundant to explain it again here. It is highly recommended that the reader carefully work through the tutorials in the manual and go over the reference section. Superbase-The Book is also a necessity for any level of Superbase user. This book is an excellent supplementary book, as it covers substantial material that the manual omits.

Once your data disk has been set up, you must create a database. It is recommended that you only have one database per disk. With the limits on disk space and a maximum of fifteen files per database, it does not make sense to have more than one per disk. The database name controls access to all of its' related files. It must be specified any time you want to access files and data within it. A name should be chosen such that it describes the relationships of the files that it is associated with.

Once you have created your database it is time to name the first file. The file will contain all the records for your particular application. If your list was a customer list, you would most certainly name the file <customer>. You are allowed fifteen files per database, so pick a name that reflects the data stored in it. If you work with a number of files, you will find it much easier to remember what is in them.

After the file has been designated, you will have the opportunity to design the format. This may be one of the most important aspects of database design. The format of your screen should not only be legible, but should also reflect all the data that is necessary. A very important factor in file format design is to determine how much record space you are going to need. The minimum record size is 128 bytes whether you use one or more fields. Appendix A discusses exactly how the record size is determined. If you need to put a lot of records on disk then design your record formats to not exceed 128 bytes. The system uses as many 128 byte units as it needs to store your record. For maximum storage efficiency, design your records to hold as close to a multiple of 128 bytes as you can.

Think carefully when you design the format. You may be very upset, after entering several hundred records, when you find you want to alter the structure. YOU MAY NOT ADD OR DELETE A FIELD IN THE MIDDLE OF THE RECORD FORMAT! You can only add or delete at the end of the record structure. If this problem occurs, do not despair, there are ways of getting around this limitation. We will cover this topic at a later time.

Upon naming a new file, you will default to the format option of Superbase. Superbase was one of the first database programs to have a built in entry screen generator. This means that you can design an attractive screen display with almost no effort. Most database programs of this era made you program the screen display. Superbase allows you up to four screens of display for a record format. The maximum system specifications are located in the technical appendix in the Superbase manual. The reader should browse through them to find out the system limitations pertaining to file and record development.

A screen format will include field names, descriptive text and possibly borders. A border can be painted on the screen by the use of the <esc b> combination. You can then pick the character to use for the border on the screen. The reference-format section of the manual details the use of fields types and other information on setting up your file format. In the limited space for this column, it is impossible to cover everything on such a powerful system. It is important that the reader cover all the material in the manual themselves.

The only field type we will cover here in depth is the KEY field, which is the most important field type. The KEY field must be defined before any record format can be saved. The maximum KEY size is 30 characters. It is recommended that the KEY be kept as small as possible for space considerations and access to records. A very important concept is that the KEY field is a text field. Numbers and characters can be entered together. The importance of this will be made clear, when we enter the programming language of Superbase.

The KEY field is the field by which a record is indexed. Indexing of a database is a very important concept as this is the only efficient method of accessing a single record. A record can be accessed almost instantaneously by the key field. On the other hand if you were looking for a specific match in a non-KEY field it would take substantially longer. It is unfortunate that Superbase can only index on one field in a file. There is a way around this one key limitation by linking files. The cost is as much storage space being used up by a second related file as was used in the first. If your files are not large, you have so much data you are exceeding the minimum 128 bytes or you have a hard disk this may pose little problem. We will discuss this in detail when we enter the programming environment.

Once data is stored in the key field, the field can not be reduced to less than the largest item already present. Therefore if your largest key is 10 characters then you may not make the key field less than 10 characters. You will also have a choice of allowing duplicate keys. A duplicate key is one that is identical to one already entered into the database. The limitation is that you may not directly select any item after the first one is entered. This means if your key was Smith and there were five Smiths, the only one you could access would be the first Smith. There is a slow method around this by either programming or other direct commands.

The result field deserves some looking at because its validity is questionable. A result field is a field whose value is dependent on a calculation derived from using the contents of other fields. The only logical reason to use a result field is if you do not program your database. If you do program, the result field just uses up extra space on the disk. By programming, you can calculate any value that is necessary for reporting. By using a result field you just add one extra field to your file that is not really necessary. Considering the limited disk space we are dealing with, conserving is a virtue. One other problem is that it appears that a result field is zeroed when selected. If you select a record and try to use the result field in a comparison it will not be valid.

A text field stores strings of characters, both alphabetic and numeric. Numeric fields store only numbers. A forced field is one in which it forces the user to enter data in a particular field before they can store the record. Very useful! A constant field is used for holding a value that will not change from record to record. A sales tax field would be an example of this. The last two fields the date and calendar fields are again questionable. While I have not found a use for either maybe someone else has. Rather than using a date field it seems more logical to use a numeric or text field. The date would be entered as 880102 to signify January 2nd, 1988. Use of this format makes it very easy to compare and select dates.

After the format has been designed and saved you will want to start entering records. The way to exit the formatting process is by pressing the keys <esc> <stop>. This writes the format to disk for future use. The Menu 1 default screen will be displayed after you are finished with the format screen. The menu 1 screen is determined by the fact that <1 enter> is the first selection choice on the screen.

To enter a record into a file, you must have the correct file name displayed on the screen. After creating a file and screen display, the default file for the entry mode will be the one just created. To enter a record press <1> for enter. Enter all the information in the correct fields and press return when you are finished with the last field on the screen. This final return will store the record. You may also press <shift><return> at any field in the record and you will be prompted to store the record.

Superbase will monitor the type of data you enter into a record format to verify you enter the proper type of data in each field. This means that Superbase will not allow you to enter, as an example, a non-numeric character in a numeric field. NEVER ENTER THE DREADED DOUBLE QUOTES < " > IN ANY FIELD OR I GUARANTEE YOU WILL LIVE LONG ENOUGH TO REGRET IT! Superbase does not prohibit you from entering a double quote in a KEY or TEXT field. So be careful and do not use it as shorthand for an inch. Although the /,*,-,?,< and > are legal to use in text and key fields they will invalidate some search techniques.

One final tip when you are using numeric keys is to be sure to make all your numbers the same length. Therefore use the highest number you could ever think of using and number your lowest number such as 1 a 00001 if using five places. If you do not use this method your indexing will be out of order. As an example if you had five numbers as keys, 1,2,12,3,123 your index order would be 1,12,123,2,3.

If you have entered all your records or enough to become curious about searching your database, then the <Select Menu> is your next stop. The select functions allow you to search and display any record from any data specified in your search. The options can be selected by number, first letter or by typing the command word. The manual covers the choices so we will focus on the two most important ones and comment on three others. The two most powerful commands are select key and select match.

Select Key allows you to find a record or its closest match virtually instantaneously. The biggest wait is for the floppy drive to get up to speed. The reason that access on the key field is so fast is because your whole file is indexed on the key. The index is stored in separate blocks with information on what file it represents and the location on the disk of the record. When using the key option there are three possible outcomes. A complete match of your key, a partial match of the key or no match at all. In any case without programming you will display the closest possible key to the one you entered. There are ways of programming to ignore any match but an exact match.

The Select Match option allows many ways of searching for data and allows you to choose any or all fields in the search. The only disadvantage of Select Match is that it is slow relative to an index search. On the other hand it is terribly easy to track down minute amounts of data using the match operation. Matching can occur through exact, sliding, partial and wildcard patterns. These features are extremely powerful and work with the FIND option. The match option can select all items with the desired characteristics. Press <m> to match the next record and use the <last> command to set up the database for the next match.

The other three select options to consider are the add, replace and delete functions. The add option allows the user to add a record based on an existing record. The replace option allows you to change information in an existing record and the delete function will delete a record. Note on earlier versions of Superbase it was worth exporting and importing your data files to insure the records were packed after deletions. Most database programs recommend this. This a good practice to get used to as it frees all space not in use at the time. It also may help to eliminate mismatches in your files.

The Find option is the most important option when dealing with smaller subsets of the database. This option allows you to set up a list of KEYS of records by using the select match options. The system displays the record format screen and the user selects the match criteria from page R-33 of the Superbase manual. To our knowledge there is no specified limit to options selected for the <find> command.

After the criteria has been selected, the system stores all the keys that match to the default list. This list is called <hlist>. Every time you select the find command the default <hlist> will be overwritten. If you desire to change the name from <hlist> you must specify this by using the command line and typing <find "somename">. The concept is if you keep your <hlist> updated, you will save time by not having to search your database again for the same information. This principle is sound if you are not changing and specifying a lot of different match criteria.

If you are using different criteria all the time, the method used to find your data should vary. If your data search will use most of a large database then you should not use the find command. Use the command lines or program in this situation. The find command has to search the database and store all the KEYS selected to the <hlist>, before you can output the data to screen or printer. This can be agonizingly slow on a one time search. Using programming with the select next, for displaying data and printing , is totally inefficient when locating a small number of records in a large database. Only experience can tell which method to use when searching for and outputting data.

The Output command will direct output to the screen on default. The output can be redirected by typing <print> or <display> at menu 1 or menu 2. The options for output are all the records or the records from a list. The data may be displayed either across or down as specified. You may also specify the fields desired with square brackets surrounding them. When the output is across there will be a space between fields and when it is down there will be a field on each line.

When outputting data, there are several methods of modifying your output. The first is to add descriptive text to your command line. If you add text with double quotes surrounding it like "The sum is", it will be displayed or printed. A field output can be truncated by typing a <&> in front of the field, &[field]. If the &8 is used it will display only the first eight characters. A numeric field can be truncated for one decimal point by using &2,1 in front of the field. Finally the display may be put anywhere on the screen with the <@20,10> before the field. This would put the display at column 20 and 15 lines down.

The output command can also be used to create a file readable by a word processor or used to export a file selectively. Now that Superoffice is available it really negates the use of merging from an output file with Superscript. If you wanted to export your file, picking only certain fields, then output is the way to do it. You must specify the sequential file between double quotes to export to. The form of command could be <output all to "names" [lname][fname][address][city]>. The output from the file or hlist can not be acted on, nor can a report format be created with it.

Well that does it for this issue of the Escape. In the next issue we will cover all the rest of the menu options not covered in this issue. More importantly we will start working with the Superbase programming language. This is the real power of Superbase, the fact that you can set up complete systems that will operate with little user intervention.

If you do not own Superbase 2 there are two methods of obtaining it. You may buy Superbase I from Northwest

Music Center, Inc. for $9.95 and trade it in to Progressive Peripherals for a discount on the purchase of Superbase II. Superoffice, which is distributed by Northwest Music Center, Inc., has the most current version of Superbase II, 2.08 and the most current version of Superscript II integrated.

Though you need a 256k machine to operate both programs together, Superbase will be the default program upon loading on a B-128. Therefore you can use Superbase 2.08, in Superoffice, without upgrading your B-128 and both of them if you upgrade in the future. Effectively, double the bang for the buck.

Bruce Faierson, Northwest Music Inc.
539 N. Wolf Rd., Wheeling, Il. 60090
312 520 2540

◆◆◆◆◆◆◆◆◆

A SECOND LOOK AT SUPERSCRIPT III FOR THE B'S

By: Anthony Liversidge

Edited by: Barbara Spurlock

Editors note: Some remarks in this article have been changed, others deleted. The basic content of the text remains intact.

Although (Liversidge) sees the upgrade of Superscript as a fix of SuperScript for the C-!28, at second glance, he finds gold...a program of nearly total flexibility which contains overwhelming advantages for owners of bigger machines, if they can spare the few hours needed to tame it. You have to invest time designing and installing your own command structure, but the result in that respect can be your version of ideal. All function keys can be reclaimed and defined with almost any direct command. A total of 20 Functions keys and 99 macro-definable keys mean that nearly all SuperScript II facilities can be imitated and a great range of new commands added.

There is only one temporary stumbling block. When installing your command structure, the program must be handled with care - rebooted after any set-up error exclusively from cold starts for computer, if not drive and computer both, and the default file (the prepared macro commands and printer designations) should be loaded only with safe, direct alphabetic commands (i.e. not pyramided Function-Esc-macro or macro-macro except in rare cases) - or at an unpredictable juncture a strange, as yet unidentified bug or bugs will trash some of your Function keys, or inscribe some garbage on your precious text, or even send the cursor on a trip to nowhere or some faraway line, removing all control and requiring a cold start.

Error handling is otherwise excellent, however, and the Function key trashing problem seems to occur only when over-ambitious command sequences are tried. Once properly set up it seems to handle other kinds of errors very well, including the one that freezes SuperScript II (hasty move of a range at the end of a long file).

WONDERFUL

*Expands text space substantially or even enormously: In a 128, you get an additional 288 line bank, and the normal screen expands from 714 lines to 809 lines. In a 256, you get the same expansion from 714 to 809 lines for your three banks, plus the extra 288 line bank.

In a MegaB, a 128 expanded to 1024 bytes with Anderson's board, you get 8 banks in SuperScript II, rather than one, whereas if you use SuperScript III you get 14 banks, including the smaller bank! In other words, whereas SuperScript II will give you 10,700 lines of on line text space, SuperScript III gives you 17,332 lines, i.e., about forty per cent more to use, or to lose if your power supply is untowardly interrupted.

*Major macro facility: You can create up to 99 special commands with Esc plus one or another of nearly all the keys on the keyboard. All function keys can be programmed as well. Slews of built-in Ctrl key commands can be installed on Esc-letter macros, or on the function keys, which can read the main Control key as chr$(28) and the Esc key as chr$(27). You can build your own individual total command structure with the two layers of command, Function keys and macros, which can be changed as you go along, if you wish, as far as the Esc-macro structure is concerned. Function keys can't be changed mid-stream, however, by reading through to keyboard macros, unless you studiously avoid making the macros anything but CTRL key commands (i.e. not Esc / commands). If you try it, they will probably corrupt the Function keys.

You can build your own mnemonic Esc commands, e.g., Esc w to replace their poorly allocated built in Commands, e.g., Control e, for the function 'cursor-to-next-word.' All the function keys can be instructed. Only eight are live to start with, with mostly redundant commands, but any command can be given them because they can recognise the Esc key - which means that function keys can instruct the Esc commands, which can in their turn instruct the Control Key commands.

<<Due to CBUG's inability to type set the UP ARROW symbol it is substituted below as Q >>.

The exact limits of the command structure can only be drawn by experimentation. For example, one strange phenomenon I have found is that when F10 and F19 are defined as Control w and Control W, and w is defined as a macro Esc-Qe, and W is defined as a macro Ess-Qw, i.e., Next word and Previous word, then while running the program if you change the macros on w or W to a Backslash disk function, e.g., /df, the Function keys revert to the original definitions after one or two operations!! Weird.

*The method for selecting files for loading, or scratching from disk, has a convenient means by which the directory is displayed in tabular form and a bar cursor can be used to select the correct file for the operation.

PRETTY

*No self-destruct noises from the drives when loading. Copy protection is a matter of honor with SuperScript III. (On the other hand, SuperScript II can be loaded without noise if the correct intro program is used).

*Much clearer representation of tab positions, and basic tabs are provided at start-up.

*Search and replace is improved, giving you instance by instance opportunity to yea or nay exchanges, etc.

*The para marker doesn't wipe out text on the right when you are not in the insert mode.

*Page numbers are displayed when printing to screen, even without page numbered headings in place in text.

*Erase-all is safer because, like other dangerous commands, now you have to agree to proceed.

*The directory is nice and keeps still instead of sliding up rather unreliably as in SuperScript II. It can appear in tabular form for selection of file to load, or scratch, with moving bar cursor. Picking a file name to load is easy, you can run the cursor over the list of names in the directory.

*You don't have to remember how to do slightly recherche commands like Underlining and Superscript - the menu does it for you. The menu makes switching from SuperScript II relatively painless.

*Caps lock is Shift lock which doesn't give peculiar underline join of words instead of spaces if you depress the key by mistake.

*Suitable if you want to get a C-128, since the program was written for the C-128, and shares exactly the same manual as SuperScript for the 128.

*Has soft hyphens and hard spaces.

*Cleverly positions a para mark for you at end of last word when you RETURN after last word from any position on the line, and moves cursor down a line without blanking rest of line.

*SuperScript II command structure can be almost completely imitated, so the move to SuperScript III can be made without erasing the reflexes you have built with SuperScript II.

*Range block moves do not result in a rude remark (Error - Cursor in range) if you try to move a block upwards a lower number of lines that the block takes up. It does the move without complaint.

*Can move great blocks of text in large files without bumping up ~~against~~ limit of size of text area.

*You can transpose letters.

*When it loads a block or additional file, the cursor is placed at the beginning of the added block, not at the end as annoyingly with SuperScript II.

*You can use commas in headings and footings without problems

*Unlike SuperScript II the defined search string renmains uncorrupted when switching banks.

*The Lord be praised, you can load III files into II or vice versa without real problems, except you have to cancel the last two lines of gobbledegook which get attached to SuperScript III files. Also - see below.

DISAPPOINTING

*Does not translate SuperScript II files to SuperScript III without some minor, but laborious, fixing of the few differences: Commas in headings change to up arrows; certain format commands-rb, pr, pp, pg, page number sign, and possibly others-change.

*Watch out! Unlike SuperScript II, it doesn't insert when you simply load from disk while in insert mode. It wipes out following text. You have to use the actual disk-insert command.

*Many of the Control commands are redundant or poorly assigned to letters which have no meaning; often both. However, they can easily be assigned to Esc commands with more rational alphabetic assignations.

*Goodbye to the curious but convenient quotation mark system of quickly entering file names in SuperScript II disk operations. SuperScript III gives you an alternative, labeling the file in the bank and reading that automatically when saving, and the name can be quickly changed, but that's all. The loss of the complete system is sad, but it is a relief in one way - it frees up the use of quotations marks in text without that SuperScript II fear of mucking up that file name selection system in mutli-file banks.

*You can't file a ranged block onto disk without scratching any file of the same name on the disk first.

*You can't remove the menu from the screen, so you lose two screen lines from SuperScript II. More unfortunately for B-256 users, perhaps, with their beautiful IBM quality screens that they might wish to coddle as much as possible to avoid burning and burnout, you can't blank the screen to save your phosphor as you can with SuperScript II (by inserting *fp0 at the start of a file) with one touch of F1.

DISSATISFACTORY

*Printers become a bit of a tangle to sort out. They won't necessarily work straight at first unless you know what you are doing, and there isn't an easy menu. The 4023 does not work properly in its entirety in CBM mode. (It freezes and starts flashing when it encounters Enhanced). The C-Itoh built Commodore 6400 in Diablo mode won't follow all of the special formatting instructions such as Superscript either. The poor guys trying to use the RS232 port are reportedly out of luck. Changing printers if two are on line is a pain, since the operation wipes out text in the particular bank on the screen.

*The Spellchecker is still the world's worst. For a start, it works very inefficiently compared with SuperScript II, but that hardly matters when the main dictionary (not on the back of the disc, as labeled, but on the front) is just as poorly done as in SuperScript II - includes countless 'words' (e.g., 'abel'), which are garbage and thousands of mis-spellings! (e.g., 'dumfound'), and excludes innumerable words in everyday use. Even if you build your own enormous user dictionary, this doesn't solve the problem because you cannot correct the basic dictionary. The Spellchecker is best forgotten!

*The initial setup of the function keys as provided is useless and redundant, with only eight of them live! They can be revived and assigned very easily, however. (Be brave and rewrite the ss en 30ju86 file as explained by Jim Bogart in CBUG Escape, or see the SuperScript II manual-totally omitted from the SuperScript III manual). The slew of Control keys looks poorly assigned alphabetically and just as redundant, until you realise that the macros use them and can reassign them to more alphabetically suitable keys.

*The cursor goes berserk at unpredictable moments following loading the default file and use of a macro command if you have done certain, as yet unidentified, things including such as making the backslash or the = sign macro keys in the default file, which won't accept them as operative (though they can be defined later). No recovery without power down cold start rebooting, including drives!!! Cold start booting and conservative default file definition of macro commands seems to avoid the crazy cursor problem fairly consistently in my experience, however. The problem seems to occur on booting up, or not at all.

*Function keys, especially F1, F2 and F3, get corrupted for as yet unidentified reasons spontaneously while program is running. The reason appears to be mainly if you define macros overambitiously, i.e., too long or too complex. May be associated with disk drive instruction errors, e.g., use of 1: or 0: in filing (SuperScript III does not recognise them).

It appears to be safest not to include "Control" alone as a macro (i.e.,/) on a key in the default file, and not to put a macro on the backslash key, or indeed on any non-alphabetic key except the numerals. (If you make the = key a macro for set command (/sc) as I did but do no longer, it must be done outside the default key loading i.e. define /sc as a macro on the = key AFTER loading defaults. Even then it occasionally corrupts F1 and even F2, so probably best avoided.) Keys also suddenly corrupted (unlike SuperScript II) if you have two machines on one set of drives and access disk simultaneously by mistake. Basically, it seems that any looping or pyramiding of macros onto Function keys, or macros onto macros, is at fault. But who really knows? Only time will tell. Find your own solution, when experimenting, I haven't yet defended completely against the F Key threat. (For example, loading ⓆnⓆm (move text and cursor down a line after breaking text apart) onto p as a macro corrupts F1, for no known reason at all.) But the Function key definitions and macro-commands listed below are good, i.e., armor plated against the worms.

*It still takes forever to move a block of text in a long file-go make tea-even longer than SuperScript II, it seems, though that may be an illusion because the text does not open up a gap until the last moment. At least the program doesn't freeze with carelessness in such range moves, as SuperScript II does with large files rearranged towards the end.

*The 0: an 1: method of addressing individual drives does not apply any more, evidently because C-128 owners are assumed to have only a single drive. The program annoyingly addresses only one drive at a time, except when loading when it will search both drives on an 8050. This easily leads to filing a file on an unintended drive, especially since the drive addressing system is not perfectly reliable.

*Very, very boringly the directory cannot be loaded into edit mode any more. This is another reason to keep SuperScript II around on your second B, if you have one, in order to make printed directories for the covers your old disks, and a reference file of old disk directories, etc. Otherwise you have to go back to slug it out in basic, whence you can print out and also translate directories into SuperScript form if you know how.

FUNCTION KEYS: So far these are my best suggestions for revised keys. F1, F11, F10, F20 are the same as SuperScript II. F3 and F4 allow immediate search of a disk directory to get alphabetic matching directory for files - e.g. w* will select all files beginning with w if you can't recall file name.

The F4 and F14 keys are three way useful. One tap will give you set up command for selection of files alphabetically from Disk (i.e. a* Return will list all files beginning with a on the disk), or list all files (add an * and Return) on that disk, or simply quickly list Disk names and space available (Delete the : and Return) on both disks

(Delete : and Return). F3 and F4 will also to change the drive addressed, otherwise the function of F5.

The Control key is moved to F18 from F1 for two reasons: so that one button video output is available as before with SuperScript II, and also to remove it from on-stream corruption during a work period. With a full macro command set you shouldn't need it that much, but it is awfully inconvenient if some over-ambitious macro-set up results in corruption of the control key, which is the danger if it is left on either F1 F2 or F3. You then lose all Control commands until you reboot. F8 is Go to end of text, and F18 (Control) tapped three times is Go to beginning of text.

F13 and F14 retain the tabular, cursor selection directory facility available from direct Control-menu commands for Erase File and Insert File which disappears if you make them macro commands.

1	Output-to-video	K pv
2	Search which string	K ssf
3	Find 1 Files alphabetically	K duo $1:
4	Find 0 Files alphabetically	K duo $0:
5	Choose drive	K dud
6	Name Document	K dn
7	Go to line number	K gn
8	End of text	K ge
9	Next screen	K gd
10	Next word	K w (with w macro'd as Qe)
11	Reoutput to video	K pr
12	Pause ps:	K lp
13	Erase document from disk	K due Esc
14	Insert file from disk table dir	K di Esc
15	Set tab	K ts
16	Clear tab	K tr
17	Comment *nb:	Esc * nb:
18	Control key	K
19	Previous screen	K gu
20	Previous word	K W (with W macro'd as Qw)

DEFAULT FILE: The following default file commands work OK. I imitate SuperScript II as far as possible. There is a wide array of possibilities. Beware, however, of loops and pyramids - the program does not like using any macro for "set command" (i.e./sc on any key), or use of the backslash key as a Control Key macro (i.e loading / key with /), or loading / as a macro on any key, or loading any macro on top of another. However, the use of the asterisk * to make reverse asterisk macro is OK. I mimic SuperScript II as far as possible e.g. Esc r is the range block comand as in SuperScript II, and replace existing file is Esc (upper case) F. Making screen up and down F9 and F 19 speeds things up and frees Esc D for delete.
a=/ar - block append;
c=Qf change case of word;
d=/eb draw a block to delete
e= nothing yet;
f=/df file existing document;
g= h= nothing;
i=/di document insert at cursor;
j= k= nothing yet;
l=/dl load a file from either drive;
m=/am move the block to here;
n=/dn document
o= nothing
r=/ab draw a range block;
s=/due scratch doc from disk
t= u= v= w= y= nothing;
w=Qe cursor to next word;
x=/am move block
z= nothing;
A=/gc10Qm go to bank A: B=/gc11Qm same thing for B;
C=/gc12Qm; D=/gc13Qm; E=/gc14Qm;
F= /dr replace file on disk
G= H= nothing;
I=/si insert toggle (shouldn't be too accessible);
J to Q = nothing;
R=/at draw a column range to move bodily;
S to Z= nothing;
!= @= #= $= %= Q= &= nothing;
*=Q1 sets command marker;
don't use (for macro
don't use) for macro
-=/el remove line;
+=/ai add a line;
=nothing;
;=/duo$0: select specific files from drive 0 for directory display;
:=/duo$1: select specific files from drive 1 for directory display:
?=/duo$Qm tell me which disk is in which drive and how much left;
0=/gc go to which bank
1=/gc1Qm go to bank 1; 2=/gc2Qm go to bank 2; 3 to 9 same thing go to (which) bank;
don't put / on a macro

◆◆◆◆◆◆◆◆◆

UPDATED SUPERSCRIPT II VERSION OF FINANCIAL SPREADSHEET TEMPLATE

By: Fred J. Petersen

Disclaimer: Author certifies that to the best of his knowledge all materials incorporated in this work are in the public domain. Anyone having information to the contrary please notify author and the Chicago B128 Users' Group (CBUG) at once!

Reservation: Author reserves sole and exclusive ownership of this entire work or any part thereof, but extends the privilege of free use to any and all members of CBUG. NO REPRODUCTION, DISTRIBUTION or SALE of this work shall be made outside the CBUG Organization.

Subject: Financial Spreadsheet

A satisfactory yet simple method for keeping a record of all receipts and disbursements for a small business is to use a "spreadsheet".

This enables one to be aware constantly of all monies received and spent with notation as to the source of the receipts and the reason for the disbursements. The business for which this method of accounting was developed is a "services only" business. It has no tangible material to sell; however, the system would also adapt to a tangible product oriented business.

As can be noted from this recording, Superscript II with a Text Width of 180 columns is being used. Also required is a line printer which will accomodate 8-1/2 x 14 inch paper and a text pitch setting of 15 characters per inch. With this "hardware" the format for the spreadsheet can be created. The sheets are best created for one year at a time with summary sheets for the "end of year" figures. Examples of the first month's sheets and the year end sheets will be given. They are named "jandoc86r+" (for january 1986 receipts), "jandoc86d+" (for january 1986 check disbursements), "jandoc86pc+" (for january 1986 petty cash disbursements),. "sumdoc86r+" (for the year end monthly and total summary of all receipts for the year), "sumdoc86d+" (for the year end monthly and total summary of all check disbursements for the year), "sumdoc86pc+" (for the year end monthly and total summary of all petty cash disbursements for the year) and "sumdoc86t+" (for the year end monthly and total summary of the combined check and petty cash disbursements for the year.) The "+", as you are aware, maintains all tab settings. Subsequent monthly sheets for the year should be identified as "febdoc, mardoc, aprdoc, maydoc, jundoc, juldoc, augdoc, sepdoc, octdoc, novdoc and decdoc. Also needed is a monthly bank reconciliation statement. This has been included as an example and, for the month of January 1986 is referred to as "janbkdocrec86+". Additional monthly statements should be referred to as "febbkdocrec86+, etc.

IMPORTANT

For the above version of this program, use the example year 1986 when calling for the sample spreadsheets; e.g. 'jandoc86r+', etc.

****

The entire program can be modified for use with a printer able to accomodate only 8-1/2 x 11 inch paper and a normal 10 character pitch. This would, of course, require a reduction in the number of columnar headings.

For this modified version, use the example year 1987 when calling for the sample spreadsheets; e.g. 'jandoc87r+', etc.

The columnar headings used for the example check and petty cash disbursement sheets (1986 version) are suited to the particular business for which this system was designed. They should be changed to fit the needs of the business for which use this program is intended.

Armed with this explanation you should now proceed to "format" the spreadsheets for the first year. To begin, for the 1986 version, you must be in the Superscript II word processer mode and must have set the Text Width to 180 columns. Now load "jandoc86r+" to see the format for the first spreadsheet. Thereafter, to see the examples of the remaining spreadsheets, load jandoc86d+", etc., etc., etc.

While this expanded (1986) version is preferable, printer limitation may dictate that you used the modified (1987) version. If so, begin in the Superscript II word processer mode, but set the text width to the default value of 80. Then proceed to load "jandoc87r+" to see the format of the first spreadsheet. Thereafter, to see the examples of the remaining spreadsheets, load "jandoc87d+", etc., etc., etc.

As you should know from having worked with Superscript II in the past, all monthly and summary row and column totals are automatically generated by the '-' (row) '+' (column) and '*' (both column and row) operators. However, the monthly totals shown on the various summary sheets are being manually transferred from the individual monthly sheets. Help is sought for a method to transfer these figures automatically, perhaps using Superbase which is compatible with the Superscript II mode.

<<The underlying templates are released in this issue of the CBUG Library as CBUG #78 stock #12045.>>

Fred J. Petersen, P.O.Box 7501
1000 Oaks, CA 91359-7501
(tel:805/492-0066)

◆◆◆◆◆◆◆◆◆

SOLVING A PROBLEM CAUSED BY A SS-II BUG
..........(While avoiding work!)

By: Ted Lacy

Keywords: SS-II bug, page-number limits

I wish to report a verification of the page numbering bug mentioned in the Winter/Summer CBUG, page 6. I haven't written any longish projects lately so I didn't think it would affect me. I just noted that it could be a problem for some people and never thought any more about it. Then a writer friend asked me to type her novel, putting it on disc so that I could help her edit it and produce the final copies needed by the agent and publisher. Things went very smoothly until -- you guessed it -- page 255. On that page video output starts acting silly, and efforts to set the page number to be greater than 254 results in "Type C Format Error" -- which means "Number too large for command". You would think that the programming engineers would have reserved more than one byte to hold the page number value! Two bytes would have allowed the paging of an entire encyclopedia. I really wonder that the memory requirements were THAT tight.

I consider this to be a serious bug, created inadvertently by development people who were, naturally, more engineer types rather than writer types. Actually, I believe they made excellent choices in developing SS II. I LOVE Superscript II. But what am I going to do about MY problem?

This particular novel takes place in Scotland, England, Spain, and Germany so there are lots of words with umlauts and other diacritical marks to be added. I have been unable to find any special printwheels for my Olympia Compact RO, so I had to dust off my antique IBM Selectric II and locate the symbol element for it.

A PARTIAL SOLUTION.

Since I had to use the typewriter on some pages anyway, I resigned myself to typing the page number on all pages from 255 on -- and knowing that my friend is a wordy writer, she could go on for another 600 or 800 or more manuscript pages! She has done it before! Now I'm the kind of guy who likes to do some things -- and anything else is WORK! I love to type and format text on my B128, even if it is someone else's text. Let me tell you that handling each sheet and typing a page number 400 to 600 times is not my idea of fun! So I went to bed that night muttering and grumbling, wondering how I got myself into so much WORK. Before I went to sleep I got this idea: I don't have to type in the whole number. So I wrote a note to myself, that after page 254 set the page number to 55. Then all I'd have to do is type a 2 in front. So I went to sleep feeling that I'd cut down the workload a little anyway.

A BETTER IDEA.

I woke up the next morning with a brilliant idea. It was so simple, I felt dumb for not thinking of it before, causing myself all that worry and frustration. My very best inventions and creative writing have been motivated by avoiding what I perceive as work. [See pages 125 & 126 of your SS II manual.

All I had to do is this: At the end of page 254 I inserted a p#55 command and changed the header to include a "2" just before the page number. Let the <u>header</u> do all that work for me!

<u>Pages 1-254</u>: (set p#xxx at the top of every file)
*hd3,Gardner,Rain II,#

<u>Pages 255-299</u>: (set p#xx starting with 55)
*hd3,Gardner,Rain II,2#

<u>Pages 300-399</u>: (set p#xx starting with 00)
*hd3,Gardner, Rain II,3#
And so on........

I hope that my experience can save some of you time and aggravation in case you run into this problem and might not think of this simple solution. That's what CBUG is all about, isn't it? People helping people.

Ted Lacy, 680 Dickinson Ave.
Ben Lomond, CA 95005

◆◆◆◆◆◆◆◆◆

INTRODUCTION -- CPU

By: John Berezinski

Information appearing in the following articles is as I understand it. If I find that my understanding of a particular item is wrong or incomplete, I will publish updates through CBUG.

Because of time limitations, the bibliography I promised will be available on a later CBUG disk. The only direct quote of any other publications comes from the information licensed to CBUG by Commodore for it's use.

Through a greater understanding of a tool comes the abillity to use it both effeciently and elegantly. With this in mind, in the following articles, I hope to share the workings and the potentials of the B128 computer.

Much of the information shared here comes from the information given to CBUG under it's licencing agreement with Commodore. Other sources come from various books and courses. These sources will be outlined in a bibliography with comments on quality and usefulness.

Nomenclature

Because of the nature of computers, it is sometimes difficult to define terms with terms other than computer terms. But if you read through the article more than once, most of the terms will become clearer.

One set of terms you will encounter concern communication between circuits. This set includes the words wire, line, signal, bit and bus. They all try to communicate the same idea but apply to different perspectives. A wire is a physical connection between two chips. A line is about the same as a wire except I chose this term when discussing connections within a single chip where actual wires are seldom found. Signal applies to the electical state of the wires or lines. There are two states in a digital circuit and each of these states can be discribed in two ways. The first is to state the actual voltage level, either +5 volts or +0volts. These two voltage levels are valid in the B128. The other way is to state them as logic levels. +5 volts becomes a logic 1 level and +0 volts will represent a logic 0 level. When a chip recieve an electrical signal from a wire or set of wires and responds to the level of the signal, it is said to have recieved data. Each line or signal can be discribed as a bit, having a logical level of 0 or 1. If eight lines are used to communicate a set of signals, it is an eight bit bus. A bus is the name given to a wire or wires shared between two or more electrical devices. An example of a bus is the wiring in your kitchen. It is possible to plug several devices into the same wire and use them as needed. They share the same power without running seperate wires from the electric company for each appliance.

Another interchangable set of terms are the name, buffer and register. A register is an area used to store needed information. This area can be within the cpu, in a memory chip, or in an interface chip. It's function is simpy to remember. A buffer acts the same. Sometimes however, the term buffer is applied whenever the chip or circuit reading the information is not the same as the chip that placed it there.

An example.

The disk drive reads data off of the disk and places it in a buffer. An inteface chip places that data on the communication bus. Another chip takes the data from the bus and puts it in a buffer in the B128. When the cpu needs that data, it then reads it from the buffer. RAM memory caan be considered one huge buffer.

Think of a buffer as the neighborhood mailbox and a register as an individual's mailbox. The contents of neighborhood box usually does not concern most of the neighborhood as long as it gets delivered. Whereas the owner of the private mailbox is usually very concerned about the content.

6509 structure

The B128 is controlled by a microprocessor which goes by the number 6509. It is a member of a family of microprocessors known as the 65xx series (sixty five hundred series).

It is packaged in a forty pin dip, dual inline package and requires a single 5 volt power source. The 65xx series recognizes 56 documented and several undocumented machine code commands.

The 65xx series also includes interface chips which are electrically matched for easy system design. They are also bus compatible with an older series, the 68xx.

Both the 65xx and 68xx are known as eight bit microprocessors because they communicate internally and externally on eight bit (wire) bus sets.

Internally the 6509 cpu can be divided into three types of circuits. Buffers and registers, busses, and function circuits.

The two main buffers are the address and the data buffers and as the name implies each handles a specific catagory of information. They buffer (hold) the signals (information) intended for circuits either within the cpu or those without until those circuits are ready to recieve them. Think of them as a staging areas.

When a specific circuit needs information, it is transfered from a buffer or alternate circuit through a bus to a register associated with the specifc circuit. There the information remains, immediatly accessable to the circuit that requested it. Some of the registers are used for memory indexing purposes. Others point to the programs being executed by the computer. There is one for keeping track of products of the alu and one hold information needed to carry out commands

Connected to these buffers and registers are the Address, Data, and control busses. These busses interconnect various registers, buffers, and logic circuits. Many of the lines that comprise these busses extend put from the cpu into the the circuitry of the B128. The data and address busses pass through the appropriate buffers. However, several of the control lines have direct access to other chips or circuits.

Finally there are three main function circuits responsible for making decisions and altering information. They are tied to each other and the various registers by the busses. These three circuits are the instruction decoder, the control logic, and the arithmatic logic unit (alu).

Buffers and registers

The cpu uses it's two buffers to communicate with the rest of the circuits in the B128. Using the buffers as a staging area allows the cpu to execute various chores while the computer responds to the data in the buffers.

The address buffer is in reality a set of three buffers. The buffer responsible for actual communication between the cpu and the B128 is a 16 bit tristate buffer. The tristate buffer is capable of disconnecting the busses within the cpu form those without. This is necessary when another chip needs to control the system address bus. This buffer then splits into two eight bit buffers. This is done because not all of the cpu circuits require all sixteen bits of address data.

The other buffer is the data buffer. It is eight bits wide and is responsible for holding data either needed by the cpu circuits or data generated by the cpu and destined for other chips. As far as I know, it does not have tristate abilities.

Registers

There are several registers in the 6509. Some are used to store and manipulate data, both within the cpu and in memory. Others either control the actual operation of the cpu or the of the other chips in the system.

Bank registers

The B128 was designed to handle up to 1 meg of memory through the scheme of banking rather than direct addressing. This means that rather than simply addressing a byte of memory between adresses $0 to $fffff (1048575), the cpu is aware of one bank of 64k bytes of memory at a time. In order to access an alternate bank (there are 16 possible in the B128), the bank execution registers must be altered. The contents of the bank execution registers are reflected in memory adresses $00 and $01 in all banks and the registers can be altered by storing the desired value in the propriate memory location. $00, the execution bank ˬister, contains the number of the bank which the cpu is obtaining instructions from and acting on. $01, the bank indirection register, contains the number of the bank which certain commands can look at or manipulate. When one of these commands is encountered, the cpu is temporally connected to the bank pointed to by indirection register and then reset to the value of execution register when done.

The Program Counter

The program counter is a dual eight bit register (sixteen bit), whose function is to point to the next byte of data needed in the execution of a program. The byte addressed may be either an instruction or simply a needed byte of information.

How the cpu can recognize the difference between a byte of data that is a command and one that is just data is simple, it can't. The cpu goes through a cycle of read, evaluate, and execute. It starts by reading data from the byte of memory pointed to by the program counter, incrementing the program counter in the proccess. This byte of data will be assumed to be a command automaticly and evaluated. If it is a valid command, it is executed, with further incrementation of the PC if more data is needed. Upon completion of the read, evaluate, and execute cycle, the program counter should be pointed to a new valid command.

If it is not a command, it is still executed, the results usually being disaster.

The program counter registers are tied to both the data and the address busses within the cpu.

Instruction register

The instuction register holds the data retrieved when the cpu executed the instruction read cycle. This register is tied to the instruction decoder circuit and the data bus.

Input Data Latch

The input data latch is responsible for transfering information between the address and the data busses under certain conditions. It is called a latch because it can latch (hold) on to a specific piece of data until told to let go.

The Stack and Stack Pointer

The stack is a very strategic area of your computer. It is used by the cpu to keep track of where it is in the program it is executing when it encounters certain commands or the cpu recieves an interrupt signal. It is also used by many programs for temporary storage or complex banking schemes.

The stack is one of those system areas determined by the wiring of the 6509. It is located in bank 15 at memory addresses $f0100 to $f01ff (bank15 #256-#511) and can store 256 bytes of data. This memory area literaly becomes an extension of the cpu registers. When a command or operation acts on the stack, it does not matter which bank the cpu is obtaining instructions from, a stack command will always address the stack in bank15, $0100-$01ff. This means information can be stored on the stack while a program is being executed in one bank and retrieved when program execution switches to another bank.

Think about the stack as a pile of papers on a desk. If you put a piece of paper on top of the stack, it remains on top until you take it off or put another on top of it. One rule of the stack is that normally only the top byte is available. If you want the next byte down, you must remove the byte on top first. To make things more confusing, take your stack of papers and suspend them upside down from the ceiling. This is how the stack works. When a byte of information is placed on the stack, it is stored at address $01ff. The next byte is stored at $01fe.

The cpu keeps track of the stack address location where it must store the NEXT byte of data by using the stack pointer (SP) register. The stack pointer is another register internal to the 6509 and always points to an empty memory location. It is only eight bits wide and can only adress memory from $0 to $00ff (255), but the 6509 is wired to automaticaly add $0100 to the stack pointer to adress the stack at the proper memory location. Also the stack can only hold 256 bytes and if more are put on, the stack pointer loops around and stores the new bytes where the old ones were, thus writing over the original ones.

Status Register

The status register is a register of flags. A flag is a term usually applied when a byte or bit of an adress or register is used to record the occurance of an event. The data stored in the flag will not be directly related to the event, but only to its occurance. An example of a flag in real life, is the flag on rural mailboxes. The owner of the mail box sets the flag down and when the mail comes, the mailman set the flag up so the owner knows that there is mail. Upon retrieving the mail, the owner resets the flag for the next day. The flag will not tell you what the mail is, only that it has or has not come.

More on the status register. The status register is an eight bit register with each bit having the following assignment.

```
Bit 0 is the carry flag.        1= carry set
                                0= carry cleared
```
Affected by the result of arithmatic operations. Tells when the results of a math operation was too big or too small to be represented by an eight bit register.

```
Bit 1 is the equal flag.    1= true (zero or equal)
                            0= false (not zero or not equal)
```
Tells whether the results of a math operation was zero. The compare operation is actually a subtraction operation within the alu.

```
Bit 2 is the irq inhibit flag. 1= ignore interrupt request
                               0= permit interrupt request
```
Controls the cpu's ability to acknowledge a certain control signal.

```
Bit 3 is the decimal or binary 1= do math in binary coded
                                    decimal,bcd
                               0= do math in straight binary
```

```
Bit 4 is the break flag         1= brk command detected.
```
Signifies that the cpu has read a brk instruction in the program and has generated a false interrupt. A special program is then executed.

Bit 5 is not used.

```
Bit 6 is overflow.              1= overflow condition
                                0= no overflow
```
Another flag for monitoring the results of math results. This one is used when working with negative numbers.

```
Bit 7 is negative.    1= negative number or number over #127
                      0= positive number or zero.
```
Used in some kinds of math to designate a byte of data as being a negative number.

General Registers

The a (accumulator), x, and y registers are three general registers used for data manipulation. They have both shared and unique abilities in the area of data manipulation. Other than altering flags, they do not directly control the operation of the cpu.

The accumulator is connected to the data bus and the alu. It's function is to hold data needed by the alu or the results of an alu chore. Often it is used as a buffer for data transfer between memory registers.

The x and y registers are two index registers. While often used like the accumulator for simple data transfer, their specialty is the modification (indexing) of data used for memory addressing. Each has unique abilities and greatly expand the versitility of the 6509 cpu. Although tied into the addressing functions, these registers have direct access only to the data bus.

Busses

The 6509 cpu has a bus system within itself, connecting the various storage registers with different processors circuits. This bus system then extends itself out through the chip pins, allowing access and control of the B128 circuitry.

This bus system can be divided into subsystems, the power bus and the information busses.

The first, the power bus simply supplies power to all the transisters which make up the cpu and the various other chips in the B128. The rest of the busses use that power to communicate information by having one of two voltage values.

The first of the three information busses, the Address Bus, is used by the cpu to activate and inform the various other chips or memory locations in the computer that data is needed or available. The sixteen lines allow the the cpu to identify 65536 different locations. Within the cpu this bus is is treated as two eight bit (line) busses, one handling the lower half of the address data and the other handling the upper half. These sub-bussés connect the various circuits responsible for the manipulation of address data.

Generally only the cpu can decide what data can be placed on the Address Bus, however there is an exception.

This occurs when a coprocessor board, such as the 8088, or the 6545 video chip requires the use of the B128 address bus. Because only one chip can have control of this bus, the high impedience mode of tri-state address buffer is actiivated upon reception of the appropriate control signals.

Another information bus is the Data Bus. This bus is used to transfer data between the circuits or chips addressed by the cpu and the cpu. Under the supervision of the Address and the Control Busses, either the cpu or any of the various support chips may place information of the Data Bus.

The 6509 treats all address locations as memory. This means the cpu generates the same timing and voltage levels for each address connected to the bus. This signal consistancy also applies to the data bus. If the data signals sent must be modified for use by the circuit addressed, interface chips must be placed at that address. Examples would be the keyboard, the monitor, or a printers who's signal voltage levels are different from the B128. These chips then recieve the data and modify the signal levels to the correct voltage before sending it to the appropriate circuit. These chips will often also handle the problem of signal timing differences betweeen the cpu and the desired circuit.

The third bus, the Control Bus, is used for sending signals(data) both to and from the cpu. However each line is dedicated to sending the signal in only one direction. On this bus, data involving the status of the cpu and other chips is communicated. There are several lines in the control bus, some are;

Clocks 1 & 2 Two clock lines used internally for for the timing of instruction execution and register access. Externally the clock signals are used for the synchronization and bus sharing by all B128 system chips. The clock signals are generated externally and sent to the cpu.

The next three lines control program execution by the B128. They are generated externally and cause cpu to execute programs whose addresses are stored at the top of the execution bank memory.

Reset Reset occurs upon power up or when the reset button on the back of the B128 is pushed. It is an edge sensitive signal. While the signal is at zero, all communication with the cpu is ignored. When the signal on this line changes from low to high, the cpu begins execution of a program stored within it's own circuits. Upon detecting the signal transition, the cpu reads data from memory locations $fffc-$fffd and stores them in the program counter. The cpu then begins executing the program located at that address.

NMI Non Maskable Interrupt - This is another edge sensitive line whose signal is generted external to the cpu. It is capable of overiding everything but the Reset signal. Upon recieving the NMI signal, the cpu completes the command it was currently executing. It then stores the contents of the program counter and the processor status register on the stack. Finally it loads the program counter with the data found at $fffa-$fffb and executes the program addressed by the program counter. In the B128, this signal is not utilized.

IRQ Interrupt Request - The third of the overide lines available on the Control Bus is state sensitive rather than edge meaning when the signal is at +0 volts, the cpu responds to it. This signal is generated by an interface chip which checks a variety of sources for a reason to generate this signal. The source which usually causes the generation of this signal is a clock signal generated by the AC power. When triggered, the cpu proceeds to execute steps similar to those of the NMI except now it loads the program counter with the data found at $fffe-$ffff.

W/R Read/Write is a signal generated by the cpu and tells the various chips whether data is being written to or from the chip designated by the Address Bus.

SYNC Coordinates the alternate sharing of the Address and Data Busses between the cpu and the video chip. Also involved in dynamic memory refresh. Generated by the cpu to inform other chips that the Address Bus is being used by the cpu.

RDY A complement to the Sync. Signal is generated by another chip to let the cpu know that it still needs the Address Bus. This is used by the memory refresh circuit. It is also used in combination with an IRQ signal by the coprocessor board to take control of the system busses.

AEC Address Enable Control. An external control of the tri state address buffer. This signal is hard wired high and ignored.

SO Set overflow. Bringing the voltage on this pin to ground causes the overflow flag to be set. Available but not used.

P0-P3 There is an additional set of lines that fall in a catagory are technically part of the Address Bus. These are BP0-3 and are responsible controling access of the different banks of memory and support chips. The B128 is capable of supporting 16 banks of these chips. The signals for these lines are generated through two registers in the cpu and are interpreted by a series of support chips. Unlike the rest of the address bus, these lines do not pass through a tristate buffer. Any isolation from the B128 bus system must be accomplished by the support chips.

Pipelining

Retreival of instructions and information is a somewhat perculiar process in the 6509 cpu. It involves a process called pipelining.

Because the 6509 uses buffers on the Address and Data Busses, it is capable of interpreting and executing one instructon while fetching the next one. Data being executed internally is seperated from data appearing on the system busses through the buffers.

The advantage of this is that the cpu can execute instructions as fast as it can shovel them in.

The exception to this is the write command because it is putting information back on those busses.

Function circuits.

The function circuits are the decision makers of the cpu and the computer. They control the busses and registers.

The instruction decoder is the primary decision maker. It's job is the interpret the byte of data known as an instruction. As it retrieves this data off the data bus, it determines what registers and busses are needed to execute the instruction. Many of the signals on the control bus are generated by the instruction decoder. These signals in turn activate data transfer between the busses and registers or even the alu.

Tied to the instruction decoder are control ciruits. They are responsible for the timing of the different steps in the execution of an instruction. There are also circuits for making sure the instruction decoder executes reset or interrupt procedures when those signals are recieved. Others are responsible for keeping the rest of the compute informed of the actions of the cpu.

Finally, there is the arihmatic logic unit. It is in reality a pretty stupid circuit. It can add and it can shift data. Even subtraction is treated as a special way of adding. But with the correct sequerce of instructions, almost any kind of math can be executed (imitated).

The information on the function circuits is limited primarily because their actions are so isolated and buffered. All control of the functions occur through the buffers and busses, so most of the explanations went into those areas. I may include a small database on this article disk on instructions.

Regards

This concludes my general discussion of the 6509 cpu. My intent was to touch on and define many of the terms encountered in computer literature as they apply to the B128.

This document evolved from a set of decriptive paragraphs in an article on the IRQ routine. They became as large as the rest of the article, so I decided to let them stand on their own.

Again, I hope to include a bibliography of the books and sources I have found most helpful.

◆◆◆◆◆◆◆◆◆

INTERRUPT REQUEST ROUTINE part 1

By: John Berezinski

The Interrupt Request routine is the primary communication program between the B128 and you. An understanding of it's actions can often help in creating much more versitile programs. In the following articles, I will step through the machine language program, examining the various chips and routines used. Included in these articles will be copies of the actual source code used in the development of the operating system. These are under a strict copy right licencing agreement from Commodore. Please do not make these articles available to non CBUG members.

This first article is a general outline of the IRQ routine. Consider it a roadmap for finding your way through the detailed explaination.

The IRQ is a routine triggered by a hardware generated signal. This signal in turn is usually generated from the AC power running your computer. AC current alternates direction in a tightly controlled cycle. This makes it ideal for a clock source. In the United States, and Canada ?, this cycling occurs sixty times a second. In Europe, it is 50 cycles per second. By altering the voltage levels of this cycling current, the compute obtains a clock signal without an execesive ammount of software or hardware overhead.

After buffering this signal to an acceptable level, it is fed to a pin on the cpu. While the voltage level on the pin is at 5 volts, the cpu carries on normal activities. However when this voltage goes to 0 volts, the cpu begins executing a special program. The first few instructions executed are actually encoded in the microprocessor.

Upon recieving the irq trigger, the cpu completes the current instruction it is executing, incrementing the program counter to the next command in the program. It then stores the high byte of the program counter on the stack, followed by the low byte. Next it stores the contents of the processor status register on the stack. The registers most important to returning the cpu to normal operation are safely tucked away.

The cpu now loads the program counter data found at memory address $fffe(low byte) and $ffff(highbyte). With the cpu instructions completed, the cpu begins executing a program addressed by the program counter.

This program begins by storing the accumulator, the x register, and the y register on the stack. Using a copy of the stack pointer, the cpu retrieves a copy of the status register data stored on the stack. This data is then tested for a certain condition.

The condition is the setting of the brk flag. Brk is a false interrupt generated by a software command. It is used for program development. If this command is detected, irq program execution will be rerouted to the monitor program. The monitor program allows direct access to most of the cpu's registers.

However, usually the irq is real and the normal execution continues. One more important register, the indirection register, is protected on the stack. Next the decimal flag is cleared ensuring that true binary math is peformed.

With the proper conditions set, the program now checks with the interface chip responsible for communicating interrupt requests to the cpu. This chip monitors several sources for irq signals and other than the software brk command all interrupt requests must come through this chip.

If the data register of this chip contains zero, no interrupt, the program branches to the exit routine. This routine restores all protected registers and restarts execution of the program which was interrupted.

A byte of data greater than zero signals a true interrupt. Each bit of the byte of data retrieved is capable of pinpointing one source of interrupt and the program begins testing each bit. This is done by comparing the byte from the interface chip with a byte representative of a specific interrupt signal. The use of compare and branch commands permits the examining of a byte of data with damaging it's contents.

The first signal source checked is the RS232 interface chip. If interrupt was from this source, a program for handling this chip will be executed. Upon completion of the program, the main program will be restarted via the exit routine.

The next interrupt to be checked is the interprocessor interrupt. This interrupt is generated by the 8088 coprocessor board available for some of the computers. Again the appropriate routine is executed. When control is returned to the B128 cpu, it is done through the exit routine.

If no source for the iterrrupt has been found, the next bit is tested. This bit monitors another interface chip. While this chip can generate an interrupt, the code supplied simply ignores it and drops through to the exit routine.

Next source tested is the ieee 448 bus, the chip communicating with the diskdrive and printer. Again ignored and exited. The code for this bus is handled by the various basic and kernal commands.

The final bit to be tested monitors the buffered AC signal. This bit is not actually tested. If execution has gotten this far in the program, then the only source left is the AC signal.

The AC interrupt is responsible for executing the keyboard read. This occurs 50 or 60 times a second, far faster than anybody can type.

Execution of the keyboard read is done by a jump to scan subroutine. Here the routine begins by reseting several registers used to hold the result of the scan. Next the interface chip responsible for monitoring(reading) the key switches is set to acknowledge all input lines.

The keyboard is set up on a 16 by 6 wire grid. This grid is controled and monitored by the interface chip. The sixteen lines can carry a signal to the key switches and the other six recieve the signal as modified. By comparing the output signal to the input signal, the current key being pressed can be deduced.

The first read of the keyboard is done to determine if any keys are pressed. The information returned by this read is too vague to establish is a specific key is pressed.

The program tests the results for the presence of a key. If none is found, the keyboard scan subroutine is exited.

With the detection of a key, the program executes a more exacting key scan. This done by activating only one of output lines of the interface chip at a time. With a single line active, the keyboard grid is tested. If a key connected to the active line is pressed, one of the six input lines will be changed.

The content or state of these lines after the first scan is stored in a buffer for special testing later.

The routine tests each of these lines, keeping track of which line and also the total number of tests. If the first line revealed no key, each suceeding line is tested. If all six lines returned empty, the next output line is activated and the keyboard is tested again. The six lines are tested again. Remember the program is keeping track of total tests, so the second scan starts with a count of six. Using this method of accumulation, the program can determine which key is pressed. Each output line is strobed until a key is identified.

A word on priority. Priority is when one event takes precedence over another. The way the keyboard is scanned, the first key noticed will be the one aknowledged. If you hold the a key and the d key down at the same time, only the a key will be noticed because it is the first key checked.

Because of the way the first single line scan is handled, the shift and control keys are excluded from the priority scheme.

When a key is detected, the routine branches to the next section. Here the test count is stored and also used to get the ascii value of the key pressed from a reference table. These values represented unshifted keys.

The copy of the first key scan is tested at this time. This copy reveals whether the shift or control keys were pressed. If one of these wass detected or the graphics mode was set, the appropriate ascii value is retrieved from a table, again using the total key test count.

The resulting ascii key value, retrieved from the aplicable table, is tested. The test is for the function keys. If the key pressed was not a function key, the keyboard scan is exited.

If it represents a function key, a subroutine, responsible for retrieving the information stored under that key, is executed. Upon returning from the subroutine, the keyboard scan routine is exited.

Upon exiting the keyboard scan routine, the irq routine continues.
The next job performed is the checking of the stop key. The stop key is monitored by a seperate interface circuit. The irq routine reads a register in this circuit and records the result in a memory buffer.

The responsibility for using the various pieces of information the irq has retrieved belongs the operating system program.

Next the irq resets the chip responsible for the tape drive motor. There may be no tape drive, but there is some code.

Finally, the data protected on the stack is retrieved and restored to the correct registers. The program counter is loaded with the address of the next command to execute in the main program. And the program continues execution.

This concludes the outline of the irq routine. A much more detailed explanation of the interrupt routine is covered in subsequent articles. With this as a starter, you should be able to follow them more easily.

INTERRUPT REQUEST ROUTINE part 2

By: John Berezinski

Details, Details

Here is presented a detailed account of the interrupt request program. Information in this article is built upon the foundation laid by the articles "1&&cpu" and "2&&irq". The first describes the B128's microprocessor and the second gives an outline of the irq routine. They are important for understanding the following article.

When I first wrote this article, it was intended as a machine language primer. So there are extensive explainations of commands along with instructions on how to interpret the simple disassembly. I have since added the source in a second column.

The Interrupt Routine

The interrupt request is a hardware initiated program responsible for communication between the cpu and the outside world.

The first thing the cpu does on receiving an IRQ pulse, is to finish the instruction it is currently executing. It then increments the program counter to point to the next instruction and precedes to store it on the stack.

Because the PC is a sixteen bit register and the stack is an eight bit register, the cpu divids the data in the PC into two eight bit numbers. It stores these number on the stack, the high half first, then the lower half, decrementing the stack pointer as it does.

Once the cpu has stored onto the stack the address of the next operation of the program currently running in the main memory and has decremented the stack pointer register twice, once for each half of the program counter data, it stores the Status Register on the stack and decrements the stack pointer again.

After the status register has been stored on the stack, the microprocesser is ready to retrieve the address of the software portion of the IRQ routine. Retrieving the data stored at $fffe, it places it in the lower eignt bits of the program counter. It then retrieves the data at $ffff and stores it in the higher eight bits of the PC. The cpu now begins executing the program at the address in the program counter, which in the B128 will be $fbd6.

Below is a copy of the software IRQ routine disassembled. It will be shown a section at a time to give me a chance to explain what is going on.

```
simple assembly               source code
$fbd6 48        pha          nirq pha ;save registers
$fbd7 8a        txa           txa
$fbd8 48        pha           pha
$fbd9 98        tya           tya
$fbda 48        pha           pha
$fbdb ba        tsx           tsx ;check for brk
$fbdc bd 04 01  lda $0104,x   lda $0104,x
$fbdf 29 10     and #$10      and #$10
$fbe1 d0 03     bne $fbe6     bne brkirq ;yes
$fbe3 6c 00 03  jmp ($0300  puls jmp (cinv)
$fbe6 6c 02 03  jmp ($0302) brkirq jmp (cbinv)
```

The first set of numbers indicate that the program starts at $fbd6. The command is one byte long is represented by the hex number #$48 and the nmemonic PHA. PHA means push the accumulator on the stack. This command would make a copy of the data found in the accumulator, store it on the stack and decrement the stack pointer one. The number in the accumulator is not destroyed by this command but only replicated.

The next command is found at address $fbd7, represented by the hex number #$8a and the nmemonic TXA. TXA means transfer the data found in the x register to the accumulator. The data previously found in the accumulator will be written over. The x register is another general register.

At the next memory address, we again find the PHA command. The cpu takes the data which is now in the accumulator, previously in the x register, puts it on the stack and decrements the stack pointer.

TYA is the next command read by the cpu and stands for transfer the y register to the accumulator. Again the previous accumulator contents are lost.

Again we have a PHA command, storing the accumulator and decrementing the stack pointer. If you noticed, what the program has done is stored a copy of the contents of the a,x, and y registers on the stack. This means that we are free to use these registers for other things.

Next command, located at $fbdb is TSX. Transfer stack pointer to x register. This command allows us to examine the stack pointer without affecting th pointer itself.

LDA $0104,X is a much more complex command. First, it is three bytes long, bd 04 01.

The command continues execution, loading the accumulator with the data found at $0104,X. This is called an indexed command because the cpu has read in the address specified into a storage address register and then indexes (increments) it x more (where x is between 0 and 255) and performs the command operation on that address. So the accumulator is loaded with the number found at address $0104+x, x being equal to the stack pointer because we used the TSX command earlier.

A possible picture of the stack as created by our program so far.

contents	address	
aa	01ff	other data
23	01fe	high byte of pc
ff	01fd	low byte of pc
77	01fc	copy of status register
10	01fb	copu of accumulator
a0	01fa	copy of x register
bd	01f9	copy of y register
??	01f8	no current data in this location

If the stack were in this condition, the stack pointer would contain #$f8, the location of the next available stack location. The actual stack locations are reletive to the contents of the stack pointer when the IRQ routine was requested. If the stack pointer had contained #$eb when the IRQ was triggered, then the type of data found at $01fe to $01f9 would have been found at $01eb to $01e6.

The command tells the cpu to get the data found at $0104 + x, which in our example is #$f8, so the accumulator retrieves data from $01fc ($0104+#$f8). $01fc contains a copy of the status register placed on the stack earlier. So this command allows us to examine what the contents of the status register was when the IRQ routine was triggered.

The indexed load is one of the tricks of getting around the normal stack access rules..

The program counter is now pointing to $fbdf and the command AND #$10. AND #$10 says to AND the number found in the accumulator with the hex number #$10 and leave the result in the accumultor.

In the symbols of our assembler, if a number is preceded by a $, it signifies an address in hexidecimal. If it is preceded by a #$, it signifies a byte of data (number) in hexidecimal.

What AND does is perform a binary math operation. The rules for AND are:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Only when ones appear in the same bit location of each byte being compared will it appear in the answer. This is known as masking, like when you tape off a window when painting, only the unprotected area can be painted. In this case the zeros are the tape.

So our AND command loads the storage data register with #$10 and uses it as a mask on the copy of status register in the accumulator ,blanking out all flags except bit 4, the brk flag. The result of this mask is left in the accumulator. If it was set, the resulting number will still show it, but if it was cleared, the resulting number will be zero. And it will set the zero flag in the status register.

An example with no brk flagged; %00010000 (the mask) and %10000111 (copy of status register) equals %00000000

The AND #$10 command was two byte command and the program counter has incremented accordingly.

The next command, at $fbe1, is bne $fbe9. Bne is a two byte command and means branch if not equal. It reads the equal(zero) flag and if not set, it will branch to the address shown. Now it seems that this should be a three byte command, however that is an illusion done by the assembler. What the command actually tells the cpu to do is test the flags and when the test is true, branch so many bytes back or foward in the program from where the program counter shows it to be. In the 6509, there is a limit of 127 bytes foward or 128 bytes back. Again we are seeing the eight bit rule, the designers of the 65xx chips allotted eight bits for this command. Why only 127 bytes foward and not 128?.

To understand this and also what the negative flag looks for, we need to look at the binary representation of numbers. 0 equals %0000 0000 and 255 equals %1111 1111. These are the minimum and maximum that can be represented by eight bits. But the solution to our understanding is not in the ends but in the middle. Note that 127 equals %0111 1111 and 128 equals %1000 0000. Notice that the high bit has gone from 0 to 1. First this means that anytime a number is over 127, this high bit will be 1 and anytime the number is not, it will be zero. That almost sounds like a flag. So if on special occasions, we declare that a binary number represents a negative number, we can use the high bit as a flag. In fact whether we do so or not, when the cpu manipulates a byte of data, it will look at the high bit and set the negative flag in the status register accordingly. It is then up to the program to use or ignore that information.

When the cpu reads a branch command, it reads the command byte, increments the program counter. It then executes the command which informs it that there is another byte in the command. This byte is retrieved and the program counter is incremented again. The cpu now tests the status register. If it finds the branch command to be false,it continues command execution at the next command. Remember BNE is false when the equal flag is set and BEQ is false if the equal flag is cleared.

If it finds the branch command to be true, it alters the program counter. The data retrieved is the number of bytes the program counter must be adjusted to be pointed to the new routine. If the high bit of this number was 0, the number will be added to the program counter, incrementing it. However, if the high byte had been 1, the neg. flag in the status register will be set and the program counter will be decremented by the number specified. Finally, having completed the command, execution continues at the new address.

Because we are looking at the normal IRQ, we will assume that the cpu did not find the flag set and incremented the program counter to $fbe3. JMP($0300) is call an indirect jump. What it tells the microprocessor to do is load the program counter with the data found at addresses $0300 and $0301 (low byte, high byte) and continue execution of program from the address placed in the program counter.

The advantage of an indirect jump has to do with Read Only Memory and Random Access Memory. Since rom is permanent and cannot be modified, the microprocessor will always read the same instructions when executing a program. If you modified the hardware on your computer or needed to do something really different, you would not be able to tell the computer without going through some extreme measures. But if you could insert an instruction in your permanent memory to look for instructions or pointers to instructions in temporary memory (ram), you could then easly alter ram to cause the computer to do the things you need it to do.

This is in fact what the indirect jump does. The addresses at $0300, $0301 are ram addresses and on power up of the computer will be loaded with the numbers #$e9 in low byte, and #$fb in high byte. Upon encountering the indirect jump, the program counter is loaded with this address ($fbe9) and continues execution there.

```
$fbe9 a5 01     lda $01        yirq lda i6509 ;save indirect
                                 segment #
$fbeb 48        pha            pha
$fbec d8        cld            cld ;clear dec to prevent
                                 further problems
$fbed ad 07 de lda $de07       lda tpi1+air
$fbf0 d0 03     bne $fbf5      bne irq000 :handle priority
                                 irq's
$fbf2 4c a2 fc jmp $fca2       jmp prendn
```

The first command, found at $fbe9, LDA $01 tells the cpu to load the accumulator with the data found at address $01. $01, the indirection register, is the register which contains the bank address which the indirect indexed commands act on.

The next command, PHA pushes the data on the stack freeing up $01 for use by the IRQ routine.

CLD, clear decimal flag, clears the flag, bit 3, that instructs the cpu as to the type of binary math it is to perform. Normally each half of a byte can represent #$0f hex. and #$01 + #$09 = #$0a. However when the decimal flag is set, the cpu performs binary coded decimal math (bcd). Each half of a byte can only represent #$09 and #$01 + #$09 = #$10, so whereas a normal hex. number #$ff can equal 255, in bcd, #$99 equals 99 and is the highest number a byte can represent. CLD, then assures that any math done by the IRQ routine is done in proper binary math.

Next, the accumulator is loaded with data found at $de07. $de07 is called the Active Interrupt Register(AIR). This address is actually a register found in the 6525 Tri Port Interface chip #1 (tpi#1) responsible for interrupts. Note that interface chips are designed to act as buffers between two sometimes incompatible electronic systems, such as the cpu and the keyboard or the data bus and a printer. The 6525 chip is capable of being programed. In the B128, one of it's ports, port C, has been programed and wired to recieve an interrupt signal from one of five sources. After it recieves in interrupt signal, it stores the source in the active interrupt register and brings one of the lines (bits) of port C low(+0 volts). This is the line tied to the IRQ pin of the 6509 cpu which triggered the routine we are now examining.

The AIR is capable of recording interrupts from any one of five sources. And these are;

IRQ0 PWR a buffered version of the 50/60 hz signal generated by the electric company. If you live in the United States of America, the standard is 60 cycles, in Europe its 50 cycles, so the electric company literaly becomes the IRQ clock.

IRQ1 status register an interrupt generated by the ieee 448 bus, our disk drives and our Commodore printers.

IRQ2 CIAIRQ alarm from 6526 CIA complex interface chip. Alarm can be triggered by time of day clock.

IRQ3 IP this is a line found on the inter-processor connector inside the B128 and would be activated by an alternate microprocessor.

IRQ4 ACIA the interrupt line of the rs232 serial port.

When the accumulator reads data from the active interrupt register ($de07), it is reset to zero and care must be taken to protect the data retrieved.

One of the qualities of a load command is that it will affect the equal(zero) flag. If the data loaded is zero, then the zero flag is set and if the data is not zero, then the zero flag is cleared.

The next command is a branch routine again. If the data retrieved was not zero, the program counter will be incremented to be $fbf5 and the program will continue from there. However if the data was zero, the program executes the next command at $fbf2 which is jmp $fca2, where it restores the protected registers and exits the IRQ routine.

When the cpu recieved a interrupt request and began executing the IRQ routine, checked the 6525 tri port interface chip for confirmation of the IRQ and since all interrupt requests must originate through the tri port #1 chip, the software has been instructed to ignore any interrupt requests not acknowledged in the Active Interrupt Register($de07) of tpi #1. Having found no confirmation, the program restores registers and leaves the routine.

However, since there was most likely a true IRQ, the program would have found the data at $de07 to be other than zero and would have branched to $fbf5.

Starting at $fbf5, the program will do a series of compares (cmp#) and branch accordingly. The advantage of CMP over AND is that while affecting status register flags, it will not alter the value of the accumulator.

As I step through these compares, I will assume that the interrupt was called by the PWR clock and that the keyboard is to be read.

```
$fbf5 c9 10     cmp #$10    irq000 cmp #$10 ;find irq source
$fbf7 f0 03     beq $fbfc   beq irq002 ;6551
$fbf9 4c 5b fc  jmp $fc5b   jmp irq100
                              ;6551 interrupt handler
$fbfc ad 01 dd  lda $dd01   irq002 lda acia+srsn :find irq
                              source
```

The cmp command has two subtypes, cmp#$ and cmp$. The first compares the accumulator with a byte of data. The second compares the accumulator with the data found at the address specified in the command.

So the program is taking the data retrieved from the active interrupt register and compares it with #$10 or binary %00010000. If the cmp is found to be equal, it signifies that line 4 of port c of the TPI #1 chip recieved the IRQ. The compare command guarantees that an interrupt will be recognized if it comes from only one source. If the active interrupt register had returned %0001100, both line 4 and line 3 recieving IRQs, then they would be ignored.

BEQ $fbfc is another branch command having the same rules as a BNE except that the cpu will obey it only if the equal(zero) flag had been set true. If the branch command finds that the compare was true, it will branch, increase the program counter, to a routine which will service the 6551 Asyncronous Communications Interface Adaptor (ACIA) chip. This chip cares for the RS232 communications between a printer or modem and the cpu.

Again, let's assume that the compare was false. The beq command is ignored and the next command is read. Jmp $fc5b is an absolute jump, it affects no flags and simply sets the program counter to $fc5b where the program continues.

```
$fc5b c9 08     cmp #$08       irq100 cmp #$08 ;cneck if
                                 inter-process irq
$fc5d d0 0a     bne $fc69       bne irq110 ;no
$fc5f ad 0d db lda $db0d        lda ipcia+icr ;clear irq
                                  contition
$fc62 58        cli             cli ;this irq can be
                                  interrupted
$fc63 20 48 fd jsr $fd48        jsr ipserv ;do the request
$fc66 4c 9f fc jmp $fc9f        jmp irq900 ;done!
```

The cpu takes up execution at $fc5b where it again does a compare. In this case cmp #$08 (%00001000) is now examining the data in the accumulator for an interrupt on line 3. Remember that compares do not disturb the accumulator, so the original copy of the active interrupt register is still in the accumulator. IRQ3 comes from the two 60pin plugs inside the B128 and are intended for an alterate cpu(8088 or Z80).

The BNE at $fc5d is read, incrementing the program counter and executed. If the IRQ has not come from this source, line 3 of portC of the tpi#1 chip, the program counter in incremented by #$0a (#10). ($fc5f + #$0a = $fc69) where the program continues.

```
$fc69 58        cli            irq110 cli ;all other irq's may
                                 be interrupted
$fc6a c9 04     cmp#$04        cmp #$04 ;check if 6526
$fc6c d0 0c     bne $fc7a      bne irq200 ;no
```

```
$fc6e ad 0d dc lda $dc0d    lda cia+irc ;get active
                              interrupts...
```

CLear interrupt inhibit, CLI, and Set interrupt inhibit, SEI, are two commands used to manipulate bit 2 of the status register. This bit controls the cpu ability to acknowledge an interrupt signal. If it is set, the cpu will ignore most interrupts. CLI clears the flag and allows the cpu to handle interrupts. There will be times when an interrupt may be time consuming or even disasterous to the actions of a program. At a time like that the SEI command would be used. However once set, without very special programming, the cpu will never acknowledge an interrupt and no communications can occur with the interface chips. This is known as lockup and the only remedy is to shut off the computer.

Back to $fc69, the CLI command ensures that the cpu will respond to any further interrupts. While this will allow recursion of the interrupt routine, according to the information I've read on the 6525 interface chip, it will not accept or generate another IRQ signal until it is reset.

Again there is a cmp# , CMP#$04 (%00000100), and a branch if not equal(zero) which checks for an interrupt from line 2. Line 2 is the interrupt from the 6526 Complex Interface Chip (CIA). This chip controls the interface with the cassette port(what cassette port?) and the user port.

If the accumulator was not equal to #$04, the program counter is incremented to $fc7a and the program continues.

```
$fc7a c9 02     cmp #$02       irq200 cmp $02 ;check for
                                 ieee srq
$fc7c d0 03     bne $fc81      bne irq300
                                 ; need code
$fc7e 4c 9f fc jmp $fc9f      jmp irq900 ;...dump interrupt
```

Line 1 is interrupt source for the ieee 448 bus (i triple e) and cmp#$02 (%00000010) checks the copy of the active interrupt register in the accumulator for a match. If true, the branch command is ignored and $fc7e, the jmp $fc9f command is executed. The routine continues at $fc9f. This is the section of the routine which restores the a,x and y registers and exits the IRQ routine. The IRQ routine does not handle a request from the IEEE. All IEEE handling is directly handled by basic and kernal routines.

Having checked all sources of interrupt, except the keyboard, and having found either no interrupts or multiple IRQs, the branch if not equal incrememts the program counter to $fc81. Since the program decided that none of the other IRQs applied, it will service the keyboard.

```
$fc81 20 13 e0 jsr $e013      irq300 jsr key ;scan the
                                keyboard
$fc84 20 79 f9 jsr $f979      jsr udtim ;set stopkey flag
```

There are two ways to execute a subroutine and return to specified section of a program. One is an IRQ and the other is a jump to subroutine (JSR). The first can only be acomplished by an electronic signal or the use of the brk command to generate a false irq. The other is a standard command. When a subroutine is finished there must be a command which informs the cpu to return to the execution of the original program. For the IRQ, it is the return from interrupt (RTI) and for the JSR, it is the return from subroutine (RTS).

How the cpu remembers where to return to when it encounters one of these commands varies with each command. When the IRQ was triggered the cpu finish the software command it was executing, it then executed the interrupt request, pushing the high byte of the program counter, the low byte of the program counter and Status Register on to the stack. The end of the software portion of the IRQ is signaled by RTI. The cpu pulls a byte off the stack and restores the status register. Next it pulls the low byte of the program counter off the stack and restores that. Finally, it pulls the high byte of the program counter off the stack and restores that. The cpu then continues with the program it was executing when interrupted.

However, the actions a cpu takes when it encounters a JSR are different. When the cpu reads the command pointed to by the program counter, it stores it in the instruction register and increments the program counter. Evaluating the instruction causes it to fetch the next two bytes and increment the program counter. However because of the availability of a data latch within the cpu and the technique of pipelining, the cpu actually saves the program counter before it's final index. The instruction is completed with the transfer of the new address to the PC.

Notice two things. One, the Status Register was not saved. Two, the return addess was pointing at the end of the current command when saved and not the beginning of the next command. As far as the cpu is concerned, the JSR command is not completed until a RTS is encountered.

When the cpu has executed the subroutine called and encounters a RTS, it pulls the low byte and high byte of the program counter off the stack and restores them to the program counter. The cpu now considers the JSR to be complete and increments the program counter to the next command and continues.

With a basic understanding of a JSR, we'll now examine the next command JSR $e013. The current program counter is memorized onto the stack and the program counter is set to $e013 and the program continues.

```
$e013 4c 65 e8 jmp $e865     jkey jmp key
```

This is an absolute jump. The program counter is set to the new address and the program continues. This may seem like a usless step, but it's invention has a history. When a programmer creates a program, He or She usually does not know how long, what kind of redundancy, or type of fixes have to be made to have a quality program. As a result, routines may change location in memory as the program grows or is corrected. To avoid having to readjust all jumps, the programer creates a jump table. This an area containing the locations of important routines. If a routine needs to be moved, the new address is stored in the jump table. If a program looks for the address of the routine needed in jump table, then the only address that needs correcting is the one in the jump table rather than each reference in the program.

The current jump is in one such jump table located at $e000-$e024. The program then continues at $e865, the keyboard scan routine.

INTERRUPT REQUEST ROUTINE part 3

By: John Berezinski

Continuing at the keyboard scan routine.

```
$e865 10 ff    ldy #$ff      ldy #$ff ;say no keys pressed
$e867 84 e0    sty $e0       sty modkey
$e869 84 e1    sty $e1       sty norkey
$e86b c8       iny           iny ;init base kybd index = 0
$e86c 8c 01 df sty $df01     sty tpi2+pb ;allow all output
                               lines
$e86f 8c 00 df sty $df00     sty tpi2+pa
$e872 20 1e e9 jsr $e91e     jsr getkey ;get keybd input
```

The first command, LDY #$ff, loads the y register with #$ff. Another way of looking at it is that all bits in the y register are turned on, %11111111 (#$ff). Thinking this way becomes important when the cpu begins to deal with interface chips. Next the y register, containing #$ff, is stored at $e0(control or shift) and $e1(any key). These are adresses designated to contain the keyboard key currently being pressed. At this point, the program is blanking them in preperation of a keyboard scan and update.

Having reset $e0 and $e1, the key flags, the program increments the y register, INY. Since the y register contained %11111111 and this is the largest number eight bits can represent, incrementing will cause the register to recycle and now hold %00000000. Various flags in the status register will be affected, but are ignored by the program.

Next, having incremented the y register to %00000000 (#$00), the program stores the contents of the y register at

$df00 and $df01 and begins the general keyboard scan.

$df00 (Port Register A, PRA) and $df01 (Port Register B, PRB) are adresses of registers located on the 6525 TPI #2. This chip interfaces with the switches which are your keyboard. All lines of PRA and PRB are set to low (zero volts) by the store, STY, commands in preparation of a keyboard scan. They have also been programed as output ports, ie, they send a signal or voltage level to be read elsewhere. The signal sent is going to be a logical zero.

Important, so far the IRQ program has assumed that both TPI chips have been properly programed for it's use. This was performed by a routine executed when power was applied to the B128. These chips can be reprogramed with software commands, but require much care.

Again a JSR command is executed. If you are following the stack activity, you should be aware that the IRQ pushed three byte on the stack. The IRQ software then pushed three more byte on the stack, copies of a, x, and y registers. Then a JSR pushed two more bytes on. And again a JSR has been encountered, pushing two more bytes on the stack. So far the IRQ has pushed 10 bytes on the stack, ahd while there should be quite a bit of stack space left, there is always a danger from poor programing of overloading the stack. This condition will also cause a crash, so a programers should always be aware of their use of the stack.

Program execution continues at $e91e where the actual read of keyboard data occurs.

```
                                 getkey
$e91e ad 02 df lda $df02      lda tpi2+pc ;debounce keyboard
                                input
$e921 cd 02 df cmp $df02      cmp tpi2+pc
$e924 d0 f8    bne $e91e      bne getkey
$e926 60       rts            rts
```

$df02 is Port Register C (PRC) of TPI chip #2. PRC has been programed as an input port from the keyboard. It is the port which will recieve the signals put out by PRA and PRB through the keyboard. To understand how the keyboard is scanned, the wiring needs to be explained. Six lines connecting the lower six input pins of PRC to the keyboard are also connected to a buffered +5 volt source(logical 1). When PRC is read with NO keys pressed, it returns a certain value. When a key is pressed, a line from PRA or PRB is connected to a line from PRC. The low output from PRA or PRB would then brings a high line of PRC low(logical 0) resulting a different value in PRC.

This is what the software is looking for when it reads PRC (LDA $df02). The routine then compares the accumulator with PRC ($df02) and if it is not the same (BNE $e91e), it reads $df02 again. If the routine finds the contents of $df02 to be the same twice, it then returns to the routine which called this one.

RTS pulls a byte off the stack and stores it in the low half of the program counter. It then pulls the next byte off the stack and stores it in the high half of the program counter. For each pull, the stack pointer was incremented one. The cpu now considers the JSR command done and it increments the program counter and continues execution there.

```
$e875 29 3f    and #$3f       and #$3f ;check if any inputs
$e877 49 3f    eor #$3f       eor #$3f
$e879 d0 03    bne $e87e      bne *+5 ;hop over long branch
$e87b 4c f3 e8 jmp $e8f3      jmp nulxit ;exit if none
```

The data retrieved is prepared for testing by first ANDing it with #$3f (%00111111). Since the keyboard uses only the lower six lines of PRC, the upper two bits are masked with zeros. Next, an eor #$3f is performed. EOR is a binary manipulation command. What it does is compares the bits of two numbers. It followes these rules:

0 eor 0 = 0
0 eor 1 = 1
1 eor 0 = 1
1 eor 1 = 0

Exclusive means that only one of the two bit being compared can be 1. If both are 1, they exclude each other and the command returns a 0.

```
Suppose the keyboard scan returned    %01111011
First an AND #$3f is performed        %00111111
Leaving in the accumulator            %00111011

Then the EOR is executed              %00111111
Leaving in the accumulator            %00000100
```

If the keyboard scan had found no keys pressed, it would have returned %01111111 (#$7f) and the execution of this section of the routine would have returned %00000000 or zero. The BNE command would have been ignored and the JMP $e8f3 command executed.

There are three exit routines for the keyboard scan routine. They occur in consecutive order and are built upon each other. The first performs some buffer preparation before going on to the next one. This one in turn performs other operations before going to the last one. The third exit routine is the true exit and is always valid.

By exiting through the first address ($e8f3), the last key buffer is blanked before returning to the main IRQ program.

However, we need to backtrack and find out what would have happened if a key had been pressed.

If the original keyboard read ($e91e) had returned with a key, the program would have executed the AND, the EOR and branched to $e87e. Here the program begins to scan the keyboard one column at a time, starting from the left.

```
$e87e a9 ff    lda #$ff      lda #$ff
$e880 8d 00 df sta $df00     sta tpi2+pa ;allow only output
                               line 0
$e883 0a       asl           asl a
$e884 8d 01 df sta $df01     sta tpi2+pb
$e887 20 1e e9 jsr $e91e     jsr getkey ;get input from
                               line 0
```

It continues by storing #$ff at $df00(PRA), bringing all output lines high. Next it performs an ASL, arithmatic shift left. Again, an illustration is the easiest explaination.

```
SR carry bit.  data byte
      0         00101101
asl
      0         01011010
asl
      0         10110100
asl
      1         01101000
asl
      0         11010000
```

ASL shifts the bits in the byte being manipulated left. If the left most bit, high bit, was 0, the carry flag in the SR was replaced with a 0. If it had been 1, the carry bit(flag) would have been replaced with a 1. Also the right most bit, low bit, was replaced with a 0.

ASL with no suffix directly affects the accumulator, whereas ASL followed by an address will manipulate the byte specified in memory. In our case, the accumulator, containing #$ff(%11111111), becomes #$fe(%11111110) with the carry set.

The new accumulator results are stored at $df01(PRB). This causes the output port (PRB) to have it's lowest line (bit 0) low or +0 volts and the other seven lines high, +5 volts. And since PRC, our input port, is normally kept high, it will only acknowledge an input if one of it's lines is brought low. So at this point in the routine only the lowest line of PRB can affect PRC. Or we can say that all lines of PRA and all but the lowest line of PRB have been masked.

Next ,the keyboard read is repeated. This time only the first column of the keyboard is tested by the read subroutine. Because of the speed of execution of the cpu, the program is assuming that the key being pressed is the same one that it detected on the first test read.

```
$e88a 48       pha             pha ;save shift & control bits
$e88b 85 e0    sta $e0         sta modkey ;shift keys are down
```

PHA pushes the accumulator which contains result of the key read onto the stack, protecting it. It is also stored at $e0, the shiftkey buffer.

Because of the values stored at PRA and PRB, PRC recognizes the following keyboard values.

```
%01111110 f1 function key
%01111101 esc key
%01111011 tab key
%01110111 not used
%01101111 shift keys
%01011111 ctrl key
%01111111 any other keys.
```

It can also return any combination of the above keys.

Having preserved the accumulator on the stack and also at $e0, the program continues to process the data retrieved from the read of the first keyboard column.

```
$e88d 09 30    ora #$30        ora #$30 ;mask them by setting
                                   bits
$e88f d0 03    bne $e894       bne line01
                                 linelp
$e891 20 1e e9 jsr $e91e       jsr getkey ;get line inputs
                                 line01
$e894 a2 05    ldx #$05        ldx #$05 ;loop for 6 input
lines
                                 kyloop
$e896 4a       lsr             lsr a ;check line
$e897 90 10    bcc $e8a9       bcc havkey ;skip ahead if have
                                      input
```

In this next section of code there are some very percul ier programing techniques.

At $e88d, we find the accumulator ORed with #$30 (%00110000). The OR rules are as follows;

0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1

Whenever a 1 appears in the same bit location of either number, the result bit will be 1. The result of this is that a number ORed with #$30 will always be #$30 or more. In this case, the OR command masks the bits connected to shift and ctrl keys by setting them to 1. Notice you mask with zeros by using AND and mask with ones by using OR. The next branch command will be true because of the OR command. Having executed the branch command, the program skips the JSR $e91e, the keyboard read. What the programer has done is included in this routine a command which will be needed later, the keyboard read. For now though, the branch command has been added to force the cpu to skip over the keyboard read subroutine. Later, a branch or jump will direct the cpu back to $e891 and the keyboard read subroutine will be included in the execution of this section of the program.

Next, at $e894, the x register is loaded with #$05

Then a LSR is performed on the accumulator. LSR, logical shift right, is the compliment of ASL. It act in the following way.

```
data byte    SR carry bit
10011010         0
lsr
01001101         0
lsr
00100110         1
lsr
00010011         0
```

LSR shifts the bits of a byte right. The left most bit is replaced with zero. And the right most bit, the lowest bit, is shifted into the status register carry flag.

To understand what happens next, we need to review the data in the accumulator. After executing OR#$30, the accumulator will contain a value between %01 110000 (#$70) and %01 111111 (#$7f). The state of the upper two bits applies only to the B128 low boy. See hilo boys. The LSR then shifts the data in the accumulator one bit to the right into the carry flag register.

The next command is BCC $e8a9, branch if carry is cleared. If the bit shifted into carry has been set to zero because of a key being pressed, the branch command will execute. The routine advances to the next section of the IRQ routine taking with it the value of the key stored in the y registers. If more than one key had been pressed, resulting in more than one bit being set to zero, the first zero bit encountered would trigger the branch and any subsequent zeros (keys) are ignored. This is priority and this is also why the shift and the ctrl keys were masked with the OR#$30. Without the mask, the shift and ctrl keys would have priority over most of the keyboard keys and shifted characters would not be recognized.

If a 1 had been shifted into the carry register, no key, the branch would be ignored and the program continues at.

```
$e899 c8       iny             iny ;inc keydecode count
$e89a ca       dex             dex
$e89b 10 f9    bpl $e896       bpl kyloop
```

Back at $e86b, y had been incremented to zero. Now it is incremented again, INY. Next, DEX, decrease the x register to #$04

INX, INY, DEX, DEY, INC(increment memory), DEC(decrement memory) all have the ability to affect the equal flag, the carry flag, and the negative flag, if they were the last operation executed.

So the next command BPL, branch on plus, tests the negative flag affected by the last command, DEX. If the X register is still not negative, the negative flag will still be cleared and the routine branches back to $e896(LSR). Zero is considered a positive number because the cpu looks for the high bit of the data being tested to be one and in zero (%00000000), its zero.

The LSR is performed again and the carry flag is loaded with the next bit from the accumulator. If the carry flag is cleared, indicating a key, the program branches to the next section with Y now incremented one. If the carry is set, no key, BCC is ignored, Y is incremented again, x is decremented again and the BPL command tests again, branching back to $e896 if x is not negative. This routine continues to loop until LSR deposits a zero bit in the carry flag or x is decremented past zero to #$ff and the negative flag is set.

Remember that the shift key and ctrl key are masked and will not trigger the carry flag, thus are ignored. So the first column scan can only return no key, f1, ESC or tab. Information on the shift and ctrl keys has already been stored in $e0.

If the routine has looped through and tested the six lines of PRC, represented by the data in the accumulator and has not encountered a key, x will now contain #$ff and the BPL command is ignored, the routine executing the next section of the routine.

```
$e89d 38       sec             sec
$e89e 2e 01 df rol $df01       rol tpi2+pb ;rotate to activate
                                 next
$e8a1 2e 00 df rol $df00       rol tpi2+pa ; - output line
$e8a4 b0 eb    bcs $e891       bcs linelp ;loop untill all
                                 lines done
```

First, SEC, make sure the carry flag is set to one.

Then perform ROL on memory location $df01. ROL, roll (rotate) left, is just like ASL, only different. Example;

```
SR carry bit.  data byte
     0          10011101
rol
     1          0011~~0~~10
rol
```

```
      0         01110101
rol
      0         11101010
rol
      1         11010100
```

When ASL is executed the left most bit is shifted into the carry flag and the right most bit is filled with zero. With ROL, again, the left most bit is shifted into the carry flag, only now, the value that had been in the carry flag before the shift is transfered to the right most bit. If you executed ROL nine times in a row, you would end up with the same data byte as when you started.

Like ASL, ROL can act on the accumulator or a byte in memory. In this case, the program is rotating data at memory location $df01, the TPI#2 data register port B. Because PRB contains #$fe (%11111110), the carry flag remains set. Also because the carry was set at the beginning of execution, the low bit, right most, is replaced with a one. PRB now has #$fd (%11111101).

With the carry flag still set because the left most bit (high bit) of $df01 was one when it was ROLed, the program now performs a ROL on $df00, PRA. Because PRA contained #$ff (%11111111), a one is rotated out and a one is rotated in and the result is still %11111111.

The carry flag is then tested and since it is set, the routine branches back to $e891.

The program repeats the keyboard scan routine with these differences.
1. The results of the read of the first column of the keyboard was followed by masking commands for the shift and ctrl keys. Now because the reentry to the keyboard scan routine occurs through $e891, the masking commands are not included and all keys in a column will be recognized. This is also the JSR to the read routine skipped over the first time.
2. The lowest line of PRB has been brought high and the second line in now low allowing the routine to read the second column of the keyboard.
3. X is reset to #$05, however y is also at #$05 and will continue to increment each time a bit is examined.
4. If the program continues to scan each column of the keyboard and increments $df01 to #$7f (%01111111) before encountering a key, when the ROLs are executed, they wil affect $df01 and $df00 this way. The zero has migrated to the highest bit (left most) and with ROL will be placed in the carry flag, clearing it. But because the carry had been set by the SEC command before the ROL command was executed, a one is again placed in the lowest bit (right most). $df01 now contains #$ff (%11111111). The program executes ROL on $df00 with carry flag now cleared. $df00 will contain #$fe (%11111110) and the migrate to the left as the program continues loop back through the keyboard scan section.

If at any time a zero is encountered in the data from PRC, representing a key, the routine branches to the next section with y containing the key value obtained by keeping count of the number of bits tested.

On the other hand, if the program has scanned all sixteen lines of PRA and PRB and has not encountered a key, when the ROL $df00 command is executed, the zero which has been migrating across the bits of PRA and PRB will finaly rotate into the carry flag register. The BCS at $e8a4 is ignored and the program continues at $e8a6.

```
$e8a6 68       pla          pla ;clear shift/control byte
$e8a7 90 4a    bcc $e8f3    bcc nulxit ;exit if no key
```

Pull a byte of data off the stack, this is a copy of the unmasked data retrieved from the scan of the first column of the keyboard, store it in the accumulator and increment the stack pointer. In this case the PLA is executed to clean up the stack. Because the ROL commands have shifted the zero bit all the way through $df00 and $df01 and back into the carry flag, this branch command will always be true when it is encountered.

In this case, the branch may have been used because it is two bytes long, whereas a JMP command, which could have worked just as well, is three bytes long. This may seem like a lot of planning just to save one byte of memory but when memory was more expencive, every byte helped.

The branch routine again uses the exit routine that clears last key buffer before exiting the keyboard scan routine.

However if the keyboard scan and test section had encountered a key, the BCC at $e897 would have sent program execution to this section.

```
                                   havkey
$e8a9 84 e1       sty $e1          sty norkey ;have a normal
                                     keypress
$e8ab be 29 ea ldx $ea29,y         ldx normtb,y
$e8ae 68          pla              pla ;get shift/control byte
$e8af 0a          asl              asl a
$e8b0 0a          asl              asl a
$e8b1 0a          asl              asl a
```

First store Y, the number of bits tested before finding a key, at $e1. Next load X with data found at $xxxx, where $xxxx = $ea29 + the value stored in the Y register. This is an indexed load command. $ea29 is the begining of a table of numbers representing the ASCII value of the keys. The A key is in the second column and the fourth row of the keyboard. It would be the tenth bit tested, however the routine would yeild Y equal to 9, because the routine counts the first key as 0. The data at memory location $ea29 + #$09 ($ea32) is #$41, the ASCII value of the letter A. Note that the physical location of the keys does not always match the electrical organization of the columns and rows.

Having loaded x with a value out of the ASCII key table, the routine then retrieves the result of the first column scan off the stack with a PLA.

PLA was also executed when the alternate condition of 'no keys' detected. In that earlier case the data retrieved by PLA was discarded. It is very important to clean up the stack when you have used it. Even if you find that the data you have stored on it is of no use to you, take it off or the cpu may use it for part of an adress upon encountering a RTS or RTI or misuse it in some other way.

The status of the first column scan is again in the accumulator and will now be tested for shift or ctrl keys. Illustration is again the best explaination of the next three commands. Start by looking back at the chart of bit values returned by the keys in the first column. A variation of one of these values will be now in the accumulator.

When the three ASLs are performed, they will treat the data retrieved in the following manner. xxxxxx represent the unknown values of the lower six bits.

```
carry     acc.
  ?       %01xxxxxx  data from first column scan.
$e8af     asl
  0       %1xxxxxx0
$e8b0     asl
  1       %xxxxxx00
$e8b1     asl
  x       %xxxxx000
```

The accumulator has been shifted left 3 bits and various flags will be affected according to it's actual contents. The bit containing the status of the ctrl key will be shifted into the carry flag. The bit representing the shift key will be in the high bit position of the accumulator.

First a quick sumation of the following section of the IRQ program. The program is going to test a series of flags for specified character set. These character sets are ctrl, shift, graphic mode, or lower case. Upon determining which character set is desired, the routine will fetch the ASCII character code for the key pressed from the appropriate key value table. Table addresses are;

$ea29 - $ea88	unshifted alpha numeric.
$ea89 - $eae8	shifted alpha numeric.
$eae9 - $eb48	graphic symbols on the front of the keys.
$eb49 - $eba8	control-character.

```
$e8b2 90 0e      bcc $e8c2      bcc doctl ;skip ahead if
                                  control depressed
$e8b4 30 0f      bmi $e8c5      bmi havasc ;skip ahead if not
                                  shifted - have ascii
$e8b6 be 89 ea ldx $ea89,y      ldx shfttb,y ;assume shifted
                                  textual
$e8b9 a5 cc      lda $cc        lda grmode ;test text or
                                  graphic mode
$e8bb f0 08      beq $e8c5      beq havasc ;have key if text
                                  mode
$e8bd be 49 eb ldx $eae9,y      ldx shftgr,y ;get shifted
                                  graphic
$e8c0 d0 03      bne $e8c5      bne havasc ;go process ascii
                                  key

                               doctl
$e8c2 be 49 eb ldx $eb49,y      ldx ctltbl,y ;get pet-ascii
                                  char for this key
```

First the carry flag is tested with a BCC command. If the ctrl key had been pressed, the BCC would be true and the routine will continue at $e8c2. At $e8c2, another indexed load would retrieve a value for the key pressed with the ctrl key from the ctrl key table. The way this section of the routine is written, ctrl has priority over the shift key.

However if ctrl has not been detected, the routine tests the high bit for the shift key with the BMI command. If the shift key was not pressed, a one will be in the high bit of the accumulator, the BMI command will be true and the routine will continue at $e8c5. Since the ASCII value for unshifted keys was loaded in the x register before any flags were tested, back at $e8ab, the routine can branch to the next section of the program at $e8c5

If the high bit was zero, the routine continues at $e8b6. The x register is again loaded from a table using an indirect load command. This table will contain the ASCII code of shifted characters. Next the accumulator is loaded with the value of memory location of $cc. $cc is a buffer which will contain information on whether graphic characters or shifted characters are desired. This buffer would have been set after a previous IRQ. If this buffer contains zero, the branch will direct execution to $e8c5. But if it contains a number other than zero, an indexed load will again be executed. This time loading the x register with the PETascii graphic codes stored in the table at $eae9. What's PETscii?

Early in the history of data communication, a standard binary code was developed to represent letters, numbers, and various commands and signals needed to communicate with data terminals. This code was called ASCII and used seven bits or 128 numbers to represent the data. It is still the standard used in most communications between computer systems or programs.

However, since many computers use at least eight bits in a byte, another 128 numbers are available for code representation. Also many coded commands are no longer needed because of the advance of computer design. PETscii was devised by COMMODORE to take advantage of these unused code numbers to represent graphic character useful in games and graphs.

Again the zero flag is tested with a BNE and the program continues at $e8c5. Since the indexed load from the graphics table will always contain a number other than zero, this branch will always execute.

The instruction at $e8c2 is the load from the ctrl key table explained earlier.

Having retrieved the appropiate code, the routine will always end up at $e8c5.

```
                             havasc
$e8c5 e0 ff     cpx #$ff      cpx #$ff
$e8c7 f0 2c     beq $e8f5     beq keyxit ;exit if null pet-
                                ascii
$e8c9 e0 e0     cpx #$e0      cpx #$e0 ;check if function key
$e8cb 90 09     bcc $e8d6     bcc notfun ;skip - not a
                                function key
```

The program now tests the result of the key tables fetch for specific keys. The first test is for #$ff, the petscii null character, representing no character. Normally I suspect this character will not be found unless some data was misread. If this character is found, the BEQ sends execution to $e8f5 where the key number stored in a buffer and this section of the routine is exited. This is the middle of the three exit routines.

If null was not found the routine checks to see if the PETsci value returned was less than #$e0. If true, execution continues at $e8d6. If the value is greater or equal to #$e0, the routine executions a section of program which tries to determine which function key was pressed.

Remember that when used in conjunction with a compare command, BCS and BCC are not quite equal but opposite. When the cpu performs a compare, it actually performs a subtraction operation, automaticly setting the carry flag before doing the compare. When a larger number is compared to or subtracted from a smaller number, a one is borrowed from the carry and the carry is cleared. And when a smaller number is compared to or subtracted from a larger number, the carry is left set. However, one more condition can occur, the two numbers being compared can be equal. Because two equal numbers can be subtracted from each other without borrowing, the carry remains set. BCS will be a true command after a compare of two equal numbers. So BCC can be thought of a branch if less than command and BCS would be a branch if greater or equal to command.

I have not studied the disassembly of the function key routine, information on that code may be released as an addendum at a later time. So for now, let's assume that the key pressed is not a function key and the BCC command is executed. Execution continues at $e8d6.

```
                            notfun
$e8d6 8a        txa         txa ;get pet-ascii code
$e8d7 c4 cd     cpy $cd     cpy lstx ;check if same key as
                              last time through
$e8d9 f0 27     beq $e902   beq dorpt ;skip ahead if so
```

A copy of the x register is copied in the accumulator, freeing the x register for other uses. Next the y register, still containing the key number, is compared to the data in $CD, the last key buffer. If they are the same, the routine branches to the key repeat timing at $e902.

```
                            dorpt ;do repeat
$e902 c6 d8     dec $d8      dec delay ;dec initial delay
                               count
$e904 10 f1     bpl $e8f7    bpl keyxt2 ;exit if was not
                               zero-still first delay
$e906 e6 d8     inc $d8      inc delay ; else reset count
                               to zero
                             ;  check if secondary count
                               down to zero
$e908 c6 d7     dec $d7      dec rptcnt ;dec repeat btwn
                               keys
$e90a 10 eb     bpl $e8f7    bpl keyxt2 ;exit if was not
                               zero - still on delay
$e90c e6 d7     inc $d7      inc rptcnt ;reset back to zero
                             ;  time to repeat - check if
                               key queue empty
$e90e a6 d1     ldx $d1      ldx ndx ;get kybd queue size
$e9a0 d0 e5     bne $e8f7    bne keyxt2 ;exit if kybd queue
                               not empty
```

Repeat timing is accomplished by essentially wasting time manipulating two buffers $d7 and $d8. When this section of code is initially entered from the BEQ command, $d7 will contain #03 and $d8 will contain #19, set either by the cold start routine or the previous IRQ execution. Upon encountering a repeat key the first time, $d8 is decremented and the flags are checked for a positive number. If true, remember zero is also positive, BPL $e8f7 sends execution through the keyboard scan exit without recording the key in the keyboard que. When $d8 is decremented below zero, BPL is ignored and $e906 is executed, bringing $d8 back to zero. This means that if a key is held down, BPL at $e904 will be true nineteen times in a row. Since the IRQ occurs 50 or 60 times a second, depending where you live, the delay will equal the number of times BPL tests the flags * 1/number of IRQ executed per second. Delay(USA) = #$19/(1/60) = .31667 second delay.

$d7 is then treated in the same way. Now when the keyboard scan detects a key still being held down, it branches to $e902, where it decrements $d8, it tests the results and upon finding $d8 now negative increments it back to zero. It tnen decrements $d7 and tests again for a positive number. As long as $d7 stays positive after being decremented, the routine will continue to exit without storing the key in the que. Delay2(USA) = #$03/(1/60) = .05 seconds. This gives a total delay of .36667

Note in Europe, where the IRQ occurs fifty times a second, this initial delay is .46 seconds long. This may be the source of the key bounce problem in Superscript in the United States. Superscript expects the initial key delay to be longer than it actually is in the USA.

When $d7 is decremented below zero, BPL is ingnored and the x register is loaded with data from $d1, number of keys stored in keyboard que. If any keys are already stored in the que, the BNE at $e9a0 will force an exit from the keyboard scan without saving the key in the que.

Remember this whole routine is only called when the current key value equals the last key value. It's function is to slow the fetch cycle to a speed usable by the human body.

The keyboard que acts a lot like the stack and $d1 like the stack pointer, only it counts up instead of down. When $d1 contains zero, it indicates that there are no keys in the que thus acting as a counter. Now if you transfer it's contents to the x register and use an indexed store, very similar to the indexed loads used the retrieve the ascii values for the key pressed, it ($d1) acts as a pointer to the next available que location. When $d1 contains zero, x will contain zero and the indexed store will be address to the first available location. When basic is functioning or Superscript is running, they are constantly scanning the key buffer and resetting the pointer to zero. This allows a key to be held and repeated.

```
                                savkey
$e912 9d ab 03 sta $03ab,x      sta keyd,x ;store pet-ascii in
                                  kybd buffer
$e915 e8       inx              inx
$e916 86 d1    stx $d1          stx ndx
$e918 a2 03    ldx #$03         ldx #3
$e91a 86 d7    stx $d7          stx rptcnt ;reset delay btwn
                                  keys
$e91c d0 d7    bne $e8f5        bne keyxit
```

If the keyboard queue pointer ($d1) was set to zero, then the accumulator is stored at $03ab + the value in the x register, ($03ab + #$00 = $03ab). Way back at $e8d6, the contents of the x register, containing the ascii value of the key pressed was transfered to the accumulator, so that is what is saved in the queue. x is incremented and stored in queue pointer.

Next x is loaded with #$03 and stored at $d7, the delay between keys. At $e91c, we encounter a BNE which will always be true because of the load at $e918 and execution continues back at $e8f5, the keyxit.

INTERRUPT REQUEST ROUTINE part 4

By: John Berezinski

If the key repeat test fails, the program continues exectution at $e8db

```
$e8db a2 18    ldx #$13         ldx #19
$e8dd 86 d8    stx $d8          stx delay ;reset initial delay
                                  count
$e8df a6 d1    ldx $d1          ldx ndx ;get key-in queue size
$e8e1 e0 09    cpx #$09         cpx #keymax ;check if queue
                                  full
$e8e3 f0 e0    beq $e8f3        beq nulxit ;exit if yes
$e8e5 c0 59    cpy #$59         cpy #dblzer ;check if keypad
                                  - 00
$e8e7 d0 29    bne $e912        bne savkey ;go save key-in if
                                  not
$e8e9 e0 08    cpx #$08         cpx #keymax-1 ;check if room
                                  for two
$e8eb f0 06    beq $e8f3        beq nulxit ;exit if not
$e8ed e9 ab 03 sta $03ab,x      sta keyd,x ;save first zero
$e8f0 e8       inx              inx ;update queue size
$e8f1 d0 1f    bne $e912        bne savkey ;always
```

First, using the x register to move data, it resets the key delay($d8), to #$13 (#19). Next, transfering the contents of $d1, the queue size counter, into the x register, it compares the data with #$09, size of full queue and if equal, it exits keyboard scan, setting $cd ,last key buffer, to #$ff, null.

If queue is not full then the y register, containing the key number, is checked for a match with #$59, the double zero key on the keypad. If there is no match, BNE $e912 at $e8e7 sends execution to $e912.

On the other hand, if a match is found, then the x register, containing the size of the queue, is again tested. If x is found to contain #$08, there will not be enough room in the queue and BEQ will force an exit, setting $cd to null along the way.

If enough room is found, then STA $03ab,x is executed. Here the x register is used as a pointer to the first opening in the queue, by adding it to $03ab when the STA is executed. The accumulator contains the ascii value for a single zero. X is then incremented. The BNE, again always true, sends execution to $e912 where another STA $03ab,x is executed, x now pointing to the next available queue location.

$e912 saves key value in queue, increments queue pointer (x register) and stores it in que size buffer, and resets $d7, delay between keys, to #$03. The routine branches back to where the keyboard scan is exited.

```
                              nulxit
$e8f3 a0 ff    ldy#$ff         ldy#$ff
                              keyxit
$e8f5 84 cd    sty $cd         sty lstx ;save last key number
                              keyxt2
$e8f7 a2 7f    ldx #$7f       ldx #$7f
$e8f9 8e 00 df stx $df00      stx tpi2+pa ;reset output
                                 lines to allow
$e8fc a2 ff    ldx #$ff       ldx #$ff ;- stop key input
$eife 8e 01 df stx $df01      stx tpi2+pb
$e901 60       rts             rts
```

Above is the exit routines are the three possible exits.

The first is at $e8f3, where y is loaded with #$ff, the number representing null or no character found. If no character was found or because of the current conditions of various buffers, the results of the key scan needed to be ignored, the exit routine was entered here.

The next exit point is at $e8f5. If a valid key number has been recovered and meets all requirement for use, the routine enters here. Both execution from the first and the second entries store y in the last key buffer. The third exit is at $e8f7. Exit will occur here only if it is important that $CD is not disturbed, such as when a key is being held down and the delay loop is executing. Again each of the other exits will pass through this command.

Now the actual working of the exit routine. Load the x register with #$7f (%0111 1111). Store it at $df00 (TPI#2 portA). Next load the x register with #$ff (%1111 1111) and store it at $df011 (TPI#2 portB). These four commands set up the TPI ports to scan the stop key.

Finally RTS pulls a byte off the stack, storing it in the low byte of a Program Counter. Again RTS pulls a byte off the stack, storing it in the high byte of the PC. With RTS completed, the PC is incremented and execution continues at location pointed to by the PC.

```
$fc84 20 79 f9 jsr $f979      jsr udtim ;set stopkey flag
```

Again a JSR is executed, pushing first #$fc and then #$86 onto the stack and setting the PC to $f979, where execution continues.

```
                             udtim
$f979 ad 02 df lda $df02      lda tpi2+pc ;check keyboard
$f97c 4a       lsr            lsr a
$f97d b0 12    bcs $f991      bcs udexit ;no stop key
$f97f a9 fe    lda #$fe       lda #$fe ;check for shift
$f981 8d 01 df sta $df01      sta tpi2+pb
$f984 a9 10    lda #$10       lda #$10
$f986 2d 02 df and $df02      and tpi2+pc
$f989 d0 01    bne $f98c      bne udttt ;no shift key
$f98b 38       sec            sec ;shift key mark
                             udttt
$f98c a9 ff    lda #$ff       lda #$ff ;clear
$f98e 8d 01 df sta $df01      sta tpi2+pb
                             udexit
$f991 2a       rol            rol a ;move bit 0 back
$f992 85 a9    sta $a9        sta stkey
$f994 60       rts            rts
```

Having setup TPI#2 to check for the stop key, lda $df02 proceeds to read portC. LSR (logical shift right) shifts the lowest bit of the data from port C into the carry flag. If the stop key is pressed then the low bit would have been zero and the carry will now be cleared. If the stop key is not pressed then the BCS at $f97d will be true and execution will continue at $f991 where the accumulator is rolled (shifted) right, negating the LSR performed earlier and the result is then stored at $a9, a buffer set aside for remembering the results of this scan. Finally the routine is exited with a RTS.

However if the stop key has been detected, #$fe is stored $df01, portB, allowing the shift keys to be scanned again. The accumulator is loaded with #$10 (%0001 0000) and port C is read, the results being ANDed with #$10. This will mask all bits except bit 4 (remember first bit is bit zero). If the shift key is down, the result of the masking will yeild a zero, the BNE will be ignored and carry flag set. If shift is not pressed, the bit representing the state of the shift key will be one and the results of the mask will yeild #$10. The BNE then skips the SEC.

In either case, the accumulator is loaded #$ff and stored at $df01, setting portB high. Next the rol is performed, rotating the carry into the high bit of the accumulator. If the shift and stop were down together, then the accumulator will now contain #$ff (%1111 1111). If only the stop was pressed then a zero was rolled into the high bit and the accumulator will contain #$7f (%0111 1111). The results are stored are stored at $a9. The RTS causes the PC to be reloaded with the original address and incremented upon completion of the retrieval and execution resumes at $fc87.

```
$fc87 ad 01 de lda $de01      lda tpi1+pb ;get cass switch
$fc8a 10 09    bpl $fc95      bpl irq310 ;switch is down
$fc8c a0 00    ldy #$00       ldy #0 ;flag motor off
$fc8e 8c 75 03 sty $0375      sty cas1
$fc91 09 40    ora #$40       ora $40 ;turn motor off
$fc93 d0 07    bne $fc9c      bye irq320 ;jump
                             irq310
$fc95 ac 75 03 ldy $0375      ldy cas1 ;test for flag on
$fc98 d0 05    bne $fc9f      bne irq900 ;yew computer
                                control leave alone
$fc9a 29 bf    and #$bf       and #$ff-$40 ;turn motor on
                             irq320
$fc9c 8d 01 d3 sta $de01      sta tpi1+pb ;store mods into
                                port
                             irq900
$fc9f 8d 07 de sta $de07      sta tpi1+air;pop the interrupt
```

Having completed the keyboard scan, the routine now checks the line which would connect the tape player through line 7 of TPI#1 portB. LDA $de01 reads portB and BPL tests the high bit of the data retrieved. This will always be high unless you have wired some experiment to the cassette port, since there is no programing currently in the B128 to handle a cassette. BPL will be false and ignored. The y register is loaded with zero and stored at $0375, the cassette motor flag.

Next the accumulator is ORed with #$40 and the BNE is executed, continuing executiion at $fc9c, where the results are stored $de01. Remember that OR can be used as a 1's mask and the results will be %x1xx xxxx, where x can be either 0 or 1. Bit six of TPI#1 portB, where the one ended up, would control the cassette motor, which would run if bit six is low. So with bit six high, the cassette motor would be off.

Finally the accumulator is stored at $de07 which is the Active Interrupt Register of TPI#1. What this does is reset the TPI#1 chip to recieve an interrupt again. From what I can find out, what is writen to this register is not important but that the act of writing to AIR resets the chip.

```
                        prendn
$fca2 68       pla       pla ;restore registers
$fca3 86 01    sta $01   sta i6509
                        prend
$fca5 68       pla       pla ;entry point for register only
$fca6 a8       tay       tay
$fca7 68       pla       pla
$fca8 aa       tax       tax
$fca9 68       pla       pla
                        panic
$fcaa 40       rti       rti ;come here if no new nmi vector
```

The microprocessor is now ready to leave the IRQ. First PLA pulls a byte of data off the stack, incrementing the stack pointer. This is a copy of $01, the indirect bank register and is stored once again at $01. Again PLA, pulls a byte off the stack, incrementing the stack pointer. This byte of data is transfered to the y register with the TAY, transfer Accumulator to Y register. Again, PLA pulls a byte off the stack, incrementing the stack pointer. This byte is transfered to the x register, TAX. The final PLA pulls a byte off the stack and leaves it in the accumulator.

RTI triggers a hardware routine, inherent to the 65xx microprocessors. Upon reading an RTI, a byte of data is pulled off the stack and stored in the status register and the stack pointer incremented. Next another byte is pulled off the stack and the stack pointer incremented. This byte is stored in the low byte of the program counter. The final byte is pulled off of the stack and stored in the high byte of the program counter, incrementing the stack pointer.

The status register, the stack pointer, and program counter are now identical to what they were when the IRQ began execution.

This concludes the main description of the IRQ routine.

ABOUT HILOBOYS

By: John Berezinski

BE AWARE. According to schematics of the B128 Highboy and Lowboy computers, the TPI chip #2 is not wired the same. The highest two lines (bits) of port c are used to control the video output of the 6545 video output chip.

In the low boy, PRC line 7 is wired to ground. And bit 6 is wired with a link. If the computer was sold in the United States(NTSC), the link was cut and bit 6 is maintained at +5 volts. If sold in Europe, this link was left alone and this bit (pin 32) is grounded.

In the Highboy, both of these lines, pin32 and pin31, are wired high. It is important to keep this in mind, as it will affect the value of the data retrieved from $df02. Because of this, the software IRQ routine in the Lowboy may be different from the routine in the Highboy. The routine I am disassembling appears to have originated in a Lowboy.

WHAT IS AN ASSEMBLER

By: John Berezinski

This is an explaination of what an assembler is. The information stored in the computer exists as switches set on or off. The setting of these switches can be represented by numbers (see Hexprimer). When the cpu reads these switches, it sees them as switches and act on them. But we see them as representing a piece of data or a command. We have already established the use of numbers to represent the state of the switches, but if you had to remember each command as a number, it could still become very confusing.

Enter the assembler. An assembler is a program which allows you to type in a word which represents a command and the assembler then recognizes the word, replaces it with a number and places it in memory where you desire. There are two basic types of assemblers. The simple assembler has a nmemonic for each command which the cpu can understand and that is all. A nmemonic is a symbol which resembles the command, ie. LDA means load the accumulator register and STA would mean store the accumulator register.

The other type of assembler is called the symbolic assembler. It allows you to define words to mean a command or group of commands. It is almost as if you are creating a language. An example would be;

```
*=$0400 ; where to place the following routine in memory
STORE lda $0100
      sta $0200
      inc $0100
      bne store
```

This is first written as a basic program or as a file in a word processing program and is called the source code. Even the note on the first line can be included. This allows you edit and annotate the program. It is then compiled by the symbolic assembler as machine code also called object code. This example would be stored at adress $0400 and be called STORE. If you were writing a large program, having created this routine, you could access from anywhere in the program simply by using a command like JSR STORE. When the program was compiled, the compiler would translate the symbolic routine into actual machine code at $0400 and then would replace the word STORE, when found elsewhere in the program, with the actual address $0400.

Note that both assemblers act as interpreters between you and the computer. Now with the simple assembler, the interpretation usually occurs immediately after you hit the return key, whereas the symbolic assembler would be run after you have written the symbolic program. It would then create the object code program where you specified.

The code we will look at was disassembled using simple assembly nmemonic words. The first number appearing on a line is the memory adress where the command is located. Next the number representing the command found at that memory adress is shown. Finally, the nmemonic word which represents the command. Multiple numbers on a line will be explained shortly.

◆◆◆◆◆◆◆◆◆

DEPROTECTION/COPY UTILITIES

By: Norm Deltzke

A few comments appear to be in order regarding the various copy and deprotection devices available thru CBUG and applicable to the B128. This is not a comprehensive discussion as there are numerous copy utilities in the library ranging from Copy-All B128 on Liz Deal's various disks, to simpler variations on the original Butterfield unit to unit copy alls. There is even one by Dale Finkey on Swan's Utilities which will allow copying of more than one file by the same name to a destination disk by a trick of imposing a prefix on all file names from each source disk used. A noteworthy copy program from Goceliak which was not well touted which will put a serial number prefix on each sequential file and a linking command at the end of each file to allow full disk search & replace and/or linked printing of an entire disk of Superscript/seq files. This last mention (by Goceliak) is SS AUTOLINK which appears on CBUG M45.

To the main issue: The original Superscript II and Superbase I programs could not be copied by conventional means. The solution provided for this was Knights Copy All. A special copy program which makes an exact copy of Superscript inclusive of all errors. Since Precision Software was duplicating on 8250 drives, special provisions had to be made to reconstruct BAM and DIRECTORY to certain of the 8050 standards. Knight's program is loaded into the computer and immediately re-loaded by the computer into the 8050. You can then disconnect the IEEE cable from the drive and load as many 8050's as you wish should you have a duplicating project. While the original intent of Knight's program was to allow people to make backups of Superscript and Superbase for their own use under the Federal Copyright Law's fair use provisions, this program is ideal for making backups of un-protected software as well -- this is how CBUG does its duplication. Once loaded, you insert the sorce disk in drive 0 then as many destination disks as you need (in sequence) in drive one. It takes 10 minutes to make a backup with Knight's disk. So far as is known, Knights will copy anything available for the B-128 however it does not work on protected software using the 8050 and other computers such as the 8032.

Then came Casey's Scrubber. A program which will allow you to copy AND decopy-protect Superscript II. It will add Liz's fix to the resultling Superscript copy if you have a copy of Liz's program. Casey's scrubber does a partial deprotection job on Superbase as well.

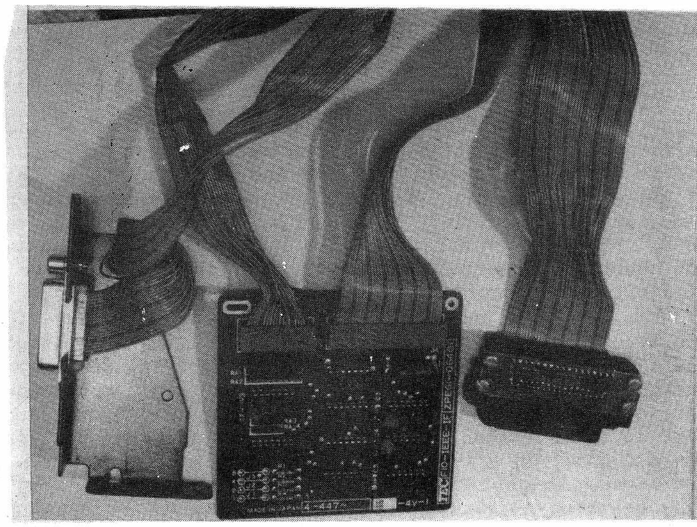
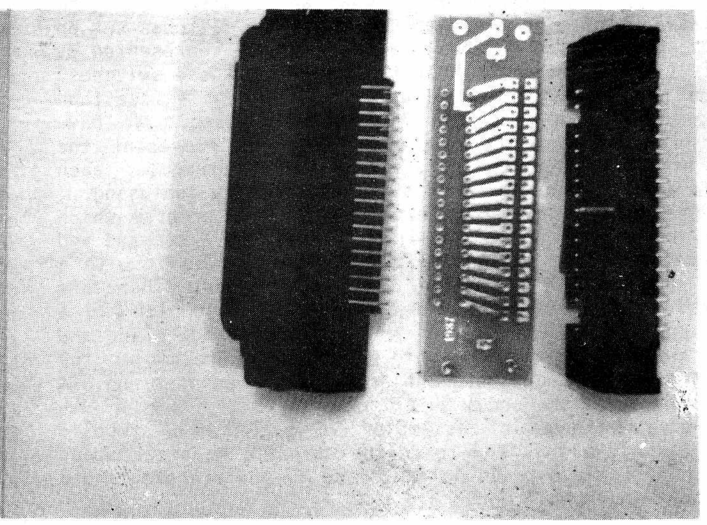
Superscript III B128 and Superbase II B128 sold separately thru commercial channels are not copy protected. Versions for other machines are copy protected. The programs, however remain encripted.

Then comes Superoffice, which is Superscript II (modified) and Superbase II combined and usuable concurently in a B128 with 256 or more memory. While the master disk provided to N.W. Music is not copy protected, it is in 8250 format which effectively prohibits duplication on an 8050 even though there is only one side used. Schwarzbauer's deprotection program de-copy protects and decrypts the program. The resulting disk is in 8050 format and can be copied with the standard backup or duplicate commands. Knight's copy all will also copy Superoffice using a 8050, however the result is still partly 8250 coded and can not be routinely copied. If you have only 128K, Superoffice will load only SUPERBASE II. (See article by Bruce Faierson, this issue.) Interestingly, Schwarzbauer's program if run on a 128K only machine will only copy & deprotect the Superbase II program.

Schwarzbauer's Deprotection Disk is being released in this issue -- check the library section for more detail.

This information is based on making various brief duplication and loading tests. This is NOT adequate testing but given the history of Knight's and the extensive use Schwarzbauer has given the deprotected copies, we are likely OK. It is known that a subtle alteration to Schwarzbauer's program will produce a copy which will occasionally crash typically about 1 hour into usage. (A good reason not to tamper with published software without the author's assistance.) Caveat Emptor.

	Order #	
Knights Copy All	12204	$20.00
CBUG #28 Casey's Scrubber	12504	19.00
CBUG #74 Schwarzbauer's Deprotection Disk	12007	15.00

IEEE to Centronics Adaptor Board
order #11221 $35.00

Connector type changer (comes assembled)
order #11236 $15.00

USE THE 6400 IEEE CONVERTER WITH OTHER PRINTERS

A couple of issues back Warren Kernaghan suggested that the 6400 converter board was a superior method of IEEE to Centronics interfacing. So why not rewire the adaptor board's flat cable to a standard 36 pin Centronics connector? Easy said, not so easy to do even for fairly good technicians. So, CBUG has made up a little circuit board and rounded up the necessary mating connectors to produce a type changer adaptor. Plug the type changer into the Centronics header on the IEEE adaptor, and the other side of the type changer into your Centronics ported printer. Walla, instant Centronics with all the added features Mr. Kernaghan wrote about.

For about a year, CBUG has been offering the IEEE internal converter designed for the CBM 6400 printer. It is a 4" square board with two long flat cables, one of which has a standard IEEE 488 connector, and the other has a 34 pin dual row 1/10" header to connect with the logic board in the 6400. Unfortunately the header with not mate with standard Contronics connectors. The circuit board is double sided plated thru, the connectors prime quality with gold plated contacts.

These adaptors are believed to work with most common printers, though they will not work with my large Daisywriter 2000. They do work with the Star and Cannon printers and many others which now follow the standards. If by chance the adaptor does not work with your printer, we'll refund the full purchase price.

The adaptors have jumper positions to select device numbers other than 4. They are not enclosed so you will want to mount or tape them securely out of the way or install them in a protective box. In some cases, the connector on the printer uses bailing clamps to keep the connectors latched together -- so you may need to use a short Centnronics extension cord compatable with the bailing clamps which are readily available locally or via mail order (see Matos's article).

REQUEST FOR REVIEWERS

By: Marilyn Gardner

In one of the first issues of the ESCAPE, a coupon was included for potential reviewers of library disks. A number of people filled out this coupon, but were not contacted to review programs. THIS WILL CHANGE. I have taken over the job of librarian, and I do not intend to do all the work myself. I just don't have the time nor the expertise. However, the list of reviewers has been lost, so I have to ask you to sign up again if you wish to review programs. Be assured that you WILL be contacted if you sign up again. Fill out the coupon included in this issue and mail it to me. You will soon receive a set of programs to review.

We need all kinds of reviewers. You do not need to be a programmer to be a reviewer, although programmers ARE needed to look at some kinds of programs. Let me know your interests and I will try to match programs to those interests. I do ask that you complete your review in a reasonable time, say a month, or else let me know of your progress, if you cannot complete your review in that time. Let's get these CBUG programs sorted out and classified and find out what is good and what isn't so we'll be able to purchase programs wisely.

- -

CBUG NEWS - - - - FOR LOCALS ONLY

Chicago Area B128/256 Users   January, 1988

CBUG EAST MEETING SCHEDULE

January 24 -- Programming by Committee -- Payroll

Ed Rhyner has written a payroll program to specifications set by Dan Schoger at a previous meeting. The program is fully operational and has been tested extensively by Ed. That's what some computer people call "alpha testing." At the January meeting, we will do what some call "beta testing," in which other users will see what they can do that may cause the program to do something unexpected! We will also take suggestions for making the program more "user friendly" and make changes needed during the meeting. See the development of a program! Watch the programmers try to think on their feet! Come join us in January!

CBUG East meets at 2:00 pm, on the fourth Sunday of each month except December, although occasional rescheduling is necessary. In case of rescheduling, all on the local mailing list will be notified by one of these mailings. There are no dues for these meetings. No one even checks to see if you are a member of CBUG. There is a sign up sheet, to make sure that you will get these local mailings. We meet at Bethlehem Lutheran Church, Wesley Avenue and Greenwood Street, in Evanston. Greenwood Street is one block north of Dempster. Wesley Avenue is a block west of Asbury (a continuation of Chicago's Western Avenue). Enter on the north side of the building. Parking on the street and behind the church. If you need more details, call Marilyn Gardner at 866-9159.

APPLICATION FOR REVIEWER STATUS

Name

Address

City State Zip

Phone Best Time to Call

I am interested in the the following types of programs:

My experience with the B-128 is as follows (include what programs you've used, whether you can program and if so, whether in BASIC or assembly language):

Mail to: Marilyn Gardner, 163Ø Madison Street, Evanston, Illinois 6Ø2Ø2

- -

CBUG WEST MEETING SCHEDULE

YOU CAN GET CBUG LIBRARY DISKS FREE OR FOR $1 above the Library's cost

Come join the growing group at CBUG West where we're evaluating which CBUG library disks may be considered as CBUG EASYWARE. Show up and take home a disk for review and it's yours FREE. Also, anyone who attends any of the CBUG West meetings can obtain any of the disks reviewed that night for $1.ØØ (or $1 plus the author's royalty for royalty disks).

Anyone can review a disk. Don't worry. The less you know - the better. We don't want experts to decide what disks are usable by us non-experts! Just grab a disk, follow the instructions on its label (if any), plug it in, and see if it makes sense to you. We'll help you, and we need your help.

CBUG West meets on the second Monday of each month at the First Congregational Church, 5th and Main Street, West Dundee, IL. Main street is State Route 72 (Higgins Road). 5th is about 6 blocks west of the Fox River and 2 blocks east of Illinois 31. Meeting time is 7:3Ø PM. Please disregard the Fox Valley Commodore Users Group meeting and go straight down to the basement where CBUG meets promptly at 7:3Ø.

For information including meeting contents contact Warren Swan at (312) 665-1514 6 to 9 PM (please no later). For weather cancellation queries, contact either Herb Gross (312) 695-1316, or Warren Swan.

---- PREVIOUS RELEASES ORDER FORM

Please repeat your name and zip code per chance your order sheets are separated:

Shipping Name: ____________________ Shipping Zip Code: ______________

Description	Quantity	Stock #	Price	Extension
RR1 Norm's Utility v1.2		12862	9.00	
CBUG #3 Swan's Utility #1		12881	14.00	
CBUG #6 CBUG/TPUG #1		12913	9.00	
CBUG #47 dFile database pgm--Available ONLY to US members		11856	16.00	
CBUG #51 JCL Work Shop & Assembler -- 2 disk set		11894	29.00	
CBUG #7 Northrup's **Superbase** Applications . . .		12932	9.00	
CBUG #13 **Superbase** tutorial pgms & Leighfield aids texts		12787	9.00	
CBUG #31 **Superbase** Corner & Hints		12538	9.00	
CBUG #15 Friendfam (**Superbase** application pgm) . .		12716	14.00	
CBUG #33 Medical Accounting (**Superbase** application)		12540	9.00	
CBUG #49 Medical Finance #2 (**Superbase** application)		11875	9.00	
CBUG #62 Super Church (**Superbase** application)		11697	9.00	
CBUG #71 Precision Church Accounting(**Superbase** application)		11875	9.00	
CBUG #8 Sermons		12946	9.00	
CBUG #9 CABS GL pro forma #1		12951	9.00	
CBUG #11 **Terminal** Pgms w/ BTerm		12257	14.00	
CBUG #11a **Terminal** Pgms w/o BTerm		12261	9.00	
CBUG #12 Scott's B-Mon		12984	14.00	
CBUG #16 Swan's Basic Course		12773	19.00	
CBUG #60 Liz Deal's Took Kit (Utilities)(current upgrade)		11659	16.00	
CBUG #67*** B128 Kernal/editor Source Code		11747	9.00	
CBUG #68*** Basic Source Code B128 & Others . . .		11752	9.00	
CBUG #69 Basic Source Code under study (Brezinski)		11766	9.00	
CBUG #66 The NEW 8432 Emulator v.g & More . . .		11733	11.00	
CBUG #18 Games and Education		12792	10.00	
CBUG #19 Old BUG texts and programs		12805	9.00	
CBUG M20 CBUG Utilities etc #2		12768	9.00	
CBUG M45 CBUG Utilities & Misc. #3 (mislabeled #2 fall 87)		12837	9.00	
CBUG M54 CBUG Misc. #4		11930	9.00	
CBUG #58 Dittinger's Utilities + ...		11925	12.00	
CBUG #21 Retail News Distribution pgm		12699	9.00	
CBUG #22 Math Education Programs		12701	9.00	
CBUG #23 Bible Games		12735	15.00	
CBUG #8 Sermons 1		12946	9.00	
CBUG #64 Sermons 2		11714	9.00	
CBUG #65 Sermons 3		11728	9.00	
CBUG #63 The New King James New Testament (on 2 disks)		11709	30.00	
CBUG #24 8432 Emulator Disassembled		12720	9.00	
CBUG #56 Harrison's Assembler, revised. 5.5 v8 . .		11959	35.00	
CBUG #27 Goceliaks Gold Mine - disk utilities/engineering		12492	9.00	
CBUG #57 Goceliak Strikes Again		11963	9.00	
CBUG #28 Casey's Scrubber		12504	19.00	
CBUG #29 CBUG TPUG P1 & P2		12519	9.00	
CBUG #32 Kernaghan's Utilities v3		11536	10.00	
CBUG #36 London Sampler		12561	9.00	
CBUG #37 SUPERPRINT		12576	19.00	
CBUG #40 Public Domain Math A		11771	10.00	
CBUG #41 Public Domain English A		11785	10.00	
CBUG #42 Public Domain GHBT		11790	10.00	
CBUG #43 Public Domain Science A		11803	10.00	
CBUG #44 Public Domain Science B		11818	10.00	
SET of 5 Public Domain CBUG #40 thr #44 inclusive		12822	45.00	
Physical Exam for the 1541		12223	35.00	
Physical Exam for the 4040		12238	35.00	
Physical Exam for the 1571		12242	35.00	
CBUG #48 CBM Diagnostics adapted for the B128		11860	9.00	
CBUG #M55 ML Programming Information		11944	9.00	
PR1 Pre Release #1		12824	9.00	
PR2 Pre Release #2		12839	9.00	
PR3 Pre Release #3		12843	9.00	
PR4 Pre Release #4		12749	9.00	
PR5 Pre Release #5		12542	9.00	
PR6p Pre Release #6 partial		12557	6.00	
PR8 CPM 86 Info & Programs #1		11611	9.00	
PR9 CPM 86 Info & Programs #2		11625	9.00	
PR10 CPM 86 Info & Programs #3		11630	9.00	
9060/9090 Service Manual - Photocopy (allow extra week)		12295	25.00	
SFD-1001 Schematics by photocopy & stat reductions		12308	5.00	
CBUG #10 Fall 1985 ESCAPE and prior files - disk . .		12965	9.00	
CBUG #25 Winter/Spring 1986 ESCAPE print files - disk		12665	9.00	
CBUG #26 Jan. 1986 Telecom issue and CBUG #25 overflow		12651	9.00	
CBUG #38 Summer part 1 1986 ESCAPE print files - disk		12580	9.00	
CBUG #52 Summer part 2 1986 ESCAPE print files - disk		11906	9.00	
CBUG #53 Fall 1986 ESCAPE print files - disk		11911	9.00	
CBUG #59 Winter/Spring 1987 ESCAPE print files - disk		11978	9.00	
CBUG #70 Summer 1987 ESCAPE print files - disk		11606	9.00	
NOTE: Fall '85 ESCAPE and Jan '86 Telecom issues are out of print. Substitute print file disks.				
Winter/Spring 1986 ESCAPE, copy of publication		12449	6.00	
Summer 1986 ESCAPE Part 1, copy of publication . .		12468	4.00	
Summer 1986 ESCAPE Part 2, copy of publication		12473	3.00	
Fall 1986 ESCAPE, copy of publication		12346	5.00	
Winter Spring 1987 ESCAPE, copy of publication		12608	7.00	
Summer 1987 ESCAPE, copy of publication		11555	4.00	

TOTAL THIS PAGE ---- Extend to main order form ____________

*** A CBM/CBUG non-disclosure agreement must accompany orders for these items

CABS/INFO DESIGNS PAYROLL REWORKED and REALLY REAPAIRED

In the spring of 1986 CBUG received the blessings of both Commodore Business Machines, Inc and Info Designs, Inc. to improve upon the CABS/Info Designs accounting suite for the B128. Over the year Protecto Enterprizes was selling the B128 system and software, several upgrades to the suite were provided to Protecto. The net result was that most later customers received the most current version, others did not and often customers did not try to use the programs for years.

CBUG is also been authorized to provide copies of the CABS Inventory program to members who can prove they have purchased or possess Order Entry which requires the Inventory program to operate fully (if at all). Inventory has been out of print for some time whereas all other programs in latest version are available from Northwest Music (see advertising).

The agreements with CBM and INFO allowed CBUG to provide the most recently upgraded versions for a nominal cost thru the CBUG Library upon proof of ownership under an appropriate user agreement with CBUG. At the same time CBUG received permission and source code allowing for repair of the remaining faults in the suite without further involvement of CBM or INFO. Finally this effort is bearing it's first fruit!!

There are two principal causes of the errors in the suite, most of which are mere irritants once you've learned to avoid or circumvent. However, two major problems remained. Certain very minor math errors ocurred on rare occasion, usually in General Ledger. This was due to the use of Pet Speed to compile the programs. This failing of Pet Speed while well known is not widely known. In any case not generally considered a "fatal error". Payroll in specific had some errors that were near fatal.

Consequently, Payroll was the first program addressed in the repair effort and is the first off the "line". The upgraded version has also compiled with a privately owned compiler which is free of the Pet Speed math error. The new version provides the major following corrections and improvements and according to the revision team is operable as originally represented and documented -- at last! Leave it to our beavers to do what no one else could!

** Roll over period adjusted to pay date;
** Fixes "New Year Bug" which caused Year End rollovers prematurely;
** Separate password on Maintain Employees to prevent computer operator from accessing employee statistics such as salary;
** Raises the number of allowable pay periods from 4 to 5 per month;
** Affects ONLY disk çA. of the Payroll pair of program disks.

To acquire any of the upgraded versions including the repaired Payroll, you need to send a self addressed paid envelope (SASE) to CBUG with a note in the lower left hand corner of the envelope saying "CABS update contract". We will send you back the manditory agreement forms and the order form for the materials. This offer also applies to a one time update of Superscript and Superbase. (yup we do just a bit of negotiating in behalf of our members).

Please be advised that the contract sets forth that the upgrades so acquired are without any warrantee or representation of workability by any of the parties -- CBUG, CBM, or INFO.

CBUG has been extensively using the CABS suite excepting Payroll which is not needed for 2.5 years. It keeps all books, writes checks, tallies everything including unthinkables. It even tattles on itself in many cases. I think it is a fine suite. The minor problems are a very small (albeit initially as frusting as can be) price to pay for the benefits gained.

CBUG will continue to try and perfect things, but few of us can afford to demand what we purchase in any area conform to our perceived ideals!

Good luck! AND, if you have a talent that can be of value to CBUG members, be sure and let us know loud and clear. The real "cost" of our group is the time of dedicated members. They deserve, each and every one, a note of thanks from every one of our membership. Yup, a bit of work and a couple of bucks in postage -- Thanks is more important than greenbacks to most all of them! TODAY, BETTER YESTERDAY!!