ISSN # 0748-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 24 May—June, 1986 $3.00 U.S.

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Production Assistant
Judie Overbeek

Circulation
Donna Carlson

Contributing Editors
C.Thomas Hilton
Donald Howes
Jerry Houston
Bill Kibler
Rick Lehrbaum
Peter Ruber

The Computer Journal® is a bimonthly magazine for those who interface, build, and apply microcomputers.

***Subscription rate**—$14 for one year (6 issues), or $24 for two years (12 issues) in the U.S., $22 for one year in Canada and Mexico, and $24 (surface) for one year in other countries. All funds must be in US dollars on a US bank.*

***Advertising rates**—available upon request.*

***Change of address**—please send your old label and new address.*

***Bulletin Board**—Our bulletin board will be on line 24 hours a day at 1200 baud, and the number is (406) 752-1038.*

Postmaster: Send address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbis Falls, Montana, 59912.

Address all editorial, advertising and subscription inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

Editor's Page

Our BBS is Here

TCJ's BBS

AMPRO has agreed to supply the equipment for our BBS system, and we expect to have it in trial operation by the time you recieve this magazine. We acquired a separate private line (not easy in our rural area) for the board so that it can be in operation 24 hours a day. It will initially be operating on 1200 baud only (8-bits, 1 stop bit, no parity).

We want to encourage you to use the board to trade tips and information, and to ask for help; but the messages should be restricted to computer related topics. Program uploads will be limited to those of interest to our readers. There is no need to duplicate the common public domain programs which are already on all the other boards. No PIRATE or illegal software please!

The contents of the board will depand on feedback from you, the reader, but some of the areas we have in mind include Program Listings for the TCJ articles, a small core of utilities (such as NULU), AMPRO support, ZTIME-I support, Periphco PROM Programmer support, Morrow support, NEW-DOS support, Turbo Pascal, C Language, etc.

The phone number for the TCJ BBS is (406) 752-1038, and if we aren't on-line the first time you call, keep on trying!

The TCJ Orphanage

Now that Morrow has filed for Chapter 11 bankruptcy, there are a lot of Morrow CP/M system owners without a source for user support (I couldn't even get support while they were in business). We are going to open TCJ's pages and BBS for support of the Morrow Decision One S-100 and their Micro Decision systems. Pass the word at user group meetings and post a notice on BBS's, because we need your help in establishing the Morrow User Support group. To start things off, I have two Decision One's with DJDMA disk controllers which only read hard sector disks, and I need someone to download the latest ROM to a disk file so that I can burn a new ROM and read soft sector disks.

Osborne has been operating under Chapter 11, and their creditors have pulled the plug and shut them down. FOG (First Osborne Group) is doing a good job of low technical level support, but we are willing to provide higher level support if the demand is there.

I understand that Vector has shut down, and we have been getting requests for Superbrain support. These are all good possibilities if we get feedback from our readers. What we really need is a coordinator for each area. This is your chance to share your knowledge and help others! Contact us with your ideas and suggestions.

Chip News

If you have older equipment containing chips not used in current production, you'd better stock up on spares because many IC vendors are dropping products which are not in volume useage. They are advising their OEM accounts to make a "Lifetime Buy" on certain chips which will be discontinued. Keep track of the status on chips you use, and stock up before they are all gone.

Zilog will be producing a HD64180 compatible chip in the next quarter. Hitachi and Zilog experts say that the growth of the Z80 software base will be in embedded computers and peripheral control, while MS-DOS continues to expand in the commercial, office, and personal computer areas. They also state the the 8-bit processor market will continue to grow, and that about 80% of all 8, 16, and 32-bit microprocessors are of the 8-bit variety. Some sources estimate that 8-bit microprocessors account for about 60% of all microprocessor revenues, and that the combination of the 8080, 8085, and Z80 account for more than 50 percent of those 8-bit revenues. See the April 28,

(Continued on page 36)

Letters From Our Readers

NEW-DOS & Public Domain

Allow me to congratulate all of you for the magnificent form in which you are producing this very useful magazine.

I am particularly taken with the kind of articles that in tutorial form show step by step how to modify, or introduce reforms to the systems already in existence. For example "NEW-DOS" by Thomas Hilton, what a load, that really excites me! And then not content as yet he points to the tools to work with it, what a Brave! This is the greatest idea and then all under the same roof in **The Computer Journal**, and the disk is our's for a meager $10.

. I do believe that knowledge is of Public Domain; not the castle of a few, because it enriches humankind, and should be available to all who want it, without regards, you gentlemen qualify here.

I want to mention Jerry Houston's "INDEXER" custom made for any one who reads magazines. You read my mind, thanks!

Rick Lehrbaum's "SCSI INTERFACE" series is just in time to understand what is going on, very useful, very informative.

H.C.

AMPRO, CP/M BIOS & Disk Formats

The January-February issue has a lot of good things that are right down my alley since I am interested in AMPRO Little Board stuff and probably will be interested in the SCSI some time in the future. At the moment, I have a very specific set of things I want to know about and would appreciate it if one of your fine CP/M experts (or you yourself) could manage to enlighten us on some of the arcane knowledge required to modify a CP/M BIOS to read/write disks according to some specific set of requirements.

It might help if I got specific. I have a Morrow MD-2 that came to me with single-sided drives and a CP/M that was configured for single-sided drives. I replaced the single-sided drives with double-sided Shugart SA-455 drives (Igave up on the REMEX as impossible—at least for me). Then I proceeded to muck around with the Disk Parameter Block in the BIOS. I managed to make some progress toward getting double-sided operation, but never did get the problem mastered. I took the coward's way out and acquired a later version of the Morrow CP/M operating system BIOS that supported double-sided drive operation.

Now, this is fine for solving the immediate problems but I want to know how o deal with the DPB, what Morrow calls the MTAB (Motor Table?) and the Boot loader. It sounds as though I know what I'm talking about but I'm really at that dangerous stage where I know roughly where my problem is but don't have a clear idea of how to solve it.

Most things are written either at a beginner's level where all the important details are left out or glossed over, or at such an advanced level that it is assumed you know a lot of stuff you really don't know. Bill Kibler's piece is a good example of the latter. I am sure he is a whiz at his subject but he lost me early on. At the end, he says, "Many little things must be kept in mind, however." It's these "little things" that I want explained! What we need is somebody that can take us by the hand and explain how to tailor the CP/M BIOS (or whatever) in some detail but also giving enough of the theory of why you are doing what you are doing so that you can adapt what you are being taught to several different situations.

I really want help in understanding the situation but I also have some practical reasons for wanting this understanding. As I mentioned earlier, I am the owner of an AMPRO Little Board in addition to my Morrow. Now, I can make disks that are readable on the Little Board on my Morrow and vice versa, but I might also want to put in quad density drives on both machines, or I might want to configure **both** of my machines to use the AMPRO format. Or, perhaps even more usefully in the long run, I might also want to put in a SCSI interface on one or both machines.

I suspect that you will eventually cover what I am asking for if you keep on in the direction you are going in. What I would find helpful is if you might encourage one or the other of your current authors to focus on this disk configuration problem early on. Rick Lehrbaum is going to be busy explaining about the SCSI interface but maybe he, or someone else from AMPRO, could take time out to give us a good rundown on CP/M disk formatting. Incidentally, I have a number of good books on CP/M (Andy Johnson-Laird, Dave Cortesi, etc.) and none of them really get down to cases on the disk drive configuring problem.

Another part of this subject, and I rather suspect quite a bit more complicated, is just how the programs like "Uniform"• that allow reading and writing in multiple formats are written. I conjecture that they might actually use the primitive commands directed to the disk controller chip itself. It would be interesting to know about that, but first let's have some dope on the CP/M treatment of disk formats.

F.O.

Editor's note:

Your questions are very timely, because I want to revise the BIOS on two systems so that the AMPRO is the normal disk format. Since AMPRO is the major supplier of Z-80 boards I want to use their format as the standard and have all the systems read and write to the same disks. I agree that the books don't give the details we need—see my editors column for information on what we are doing about this.

How about you CP/M experts providing feedback on this in either letters, articles, or on our BBS?

A New Reader

I just received my first copies of TCJ and wonder where this magazine was hiding all my life. I think that this is going to be one of those magazines that will drive me crazy waiting for the next issue to arrive. A great source of knowledge for those of us being dwarfed by big blue. A must!

R.S.

An Original Subscriber

With great pleasure, I am enclosing my check for $24 to renew my subscription to your fine publication for two more years. Frankly, when you first started, I was afraid that "The Computer Hacker" wouldn't last, but I'm very happy to be the proud possessor of ALL copies of your magazine.

It is too bad that the little man in the derby hat seems to have won the war, but it just shows that millions of dollars and the IBM name can sell mediocrity, and force it on us as "Industry Standard."

My primary machine is an ALTOS 5-15D running CP/M, but I have fixed up an AMPRO Little Board to run with it using a scheme similar to Tom Hilton's. I got the LB primarily to free myself from the limitations of the 80 track double sided drives on the Altos. I was delighted to see the last issue and welcome Tom's column.

Again, thank's for a good publication, and good luck for the future.

S.P.

Editor's Note

While the IBM-PC is technically mediocre, it is a useful appliance to use with the large amount of low cost software developed for it—and there probably wouldn't be this much software available for ONE system if the IBM-PC hadn't appeared. We don't HAVE to use the PC if we don't want to, and there is one significant benefit for those who don't like the PC. Now that the PC is King, the prices on CP/M systems have tumbled, and we can pick up great systems at dirt cheap prices.

Reader Feedback

Just a few quick comments from a new subscriber.

A) The magazine is terrific.

B) You are absolutely correct about requiring source code with your software

C) I am very interested in bus information.

D) Tom Hilton's NEW-DOS series is excellent.

E) Your new "C" and "Z" columns are of no interest to me; I prefer Pascal and assembly languages (+ NEW-DOS).

L.L.

THE HERMIT'S MAIL

French Wordstar

Dear Thomas;

I am a French Professor with a few "how to" questions. I use my computer with SMARTKEY to define the keyboard in the European style. How can I create a submit file to run SMARTKEY, load a keyboard definition file, and enter Wordstar with the ^OJ option disabled?

Robert W.
Lynchburg VA.

Well Robert, you pose a problem that is easy in speech, but more difficult in its solution. I will have to assume your keyboard definition program is smart enough to load its own definition document, once it has control of the system. This sequence should not present too much of a problem. It is the Wordstar entry that gives me pause, and it is here where we will begin.

The first thing to consider is what operating system would be the best for you. As I know you use a CP/M machine this narrows the field a great deal. I doubt if Hermit DOS would serve you. In your case standard CP/M should do the job.

(Continued on page 33)

Selecting and Building A System

by Bill Kibler

Over the past several months, I have talked about system integration from many different viewpoints. Since then I have started on and completed several projects. These recent changes and some other new writers to COMPUTER JOURNAL, have brought new information to light. This article reviews some changes to the more mature, and thereby more economical to build and use systems.

There are three type of systems to be built, they are components, kits, and un-kits. Like any list of items these are general groupings and overlap, all based on your skill level and resources. Let's look at the current most popular category first, the un-kits.

UN-KITS

Several companies have used the term un-kits in their sales literature to describe a "you do the final assembly" type of construction. When I bought my Heathkit Z100, I was rather surprised to find that the largest part of the construction was bolting things together. They did leave one card (the disc controller) to be soldered together, but the rest had only one solder joint to be made. The current PC clones, are also un-kits as most units require only mounting and connecting of internal components to get one running. Several single board computers like Ampro, Big Board I and II, Xerox 820 I and II, and others can be un-kits too. Let me show you how easy this is by reviewing my latest un-kit project for packet radio, a Xerox 820-I.

I must admit that my heart still belongs to the Z80 systems, and there are many single boards to choose from. Probably the best buy at present is an Ampro board, but these have one drawback for me. I love 8 inch drives, and still do all my work on them, all five systems to be exact. Having previously built both a Big Board I and a II (Xerox I & II are same design), using a Xerox for packet would be easy. It would require only plugging in of wires, and can run both 8 or 5 inch drives, making it an ideal system for my current needs. With boards costing $50, it is easy to see why other enthusiasts have chosen the Xerox as their system. Which is why I decided to buy one and see just what was happening on packet radio.

For packet radio you need one standard Xerox 820-I and two rather simple interface boards. The interface boards are covered quite well in other publications so I will stay as close to the putting it together stuff as possible. This stuffing is pretty much how it went. I had an old terminal which had bit the dust (used some rather hard to find devices) and used it for the box. I added one used but newer supply for both the board and monitor. The keyboard was parallel and had only to be soldered to a new DB25 connector to work. The only major work involved the soldering of the DB37 connector for the disk drives. Xerox really blew it, as this rather non-common connector does not fit either of the disk standards. I chose to solder mine to a standard 5 inch drive cable as I have a 5 inch to 8 inch adapter board. For now I use 8 inch drives, but later may change to 5's if I decide to upgrade the system.

Turning on the power produced the Xerox prompt, and using a BB-I boot disk, got me the CP/M A> message. I have since gotten 820 disks from fellow club members and now have full documentation and software. What made this job even more simple was a cook book listing of pins and their use. I got this list from an AMRAD newsletter. These people are doing packet work and produce a good monthly newsletter (POBOX 6148, McLean VA 22106-6148), which reviewed assembling the 820. I have provided my version of pin outs as appendix A to this article. I am having some troubles with the packet software and interface, but the Xerox 820-I works quite fine, especially when you consider it only took three short nights to put it together (4 to 6 hours).

There are Xerox 820-II's, which have a separate disk controller card (will also do double density), and more ROM space. I have been given one that failed, and will be turning it into a ROM-based Forth unit (without disk drives). Due to the price and their straight forward design, these units can make great dedicated controllers. This is of course why I have started messing with Forth, to build single board controller. Micro Cornicopia has a BB-I disk (#18) with IFORTH.COM. This is a monitor replacment and Forth system in one. I had to reassemble the 820 monitor code (a shortened version as described in the .DOC file), but now have a running FIG Forth system on a 820-II.

These systems were quite easy to put together, mainly because I had good documents and lots of past experience. The single boards can make easy work of system generation, even for the inexperienced computerist. PC clones are good examples of systems where no previous experience is needed, just good documents. Our club president showed slides of how his system went together in one evening, pointing out a few strange things that IBM did. It absolutely amazes me that the clones are so close to IBM's that they even reproduce the mistakes perfectly. For the novice assembler, this holding to true blue means that almost any parts from any vendor will work if you follow their instructions. If I can talk a fellow club member into ghost writting his research, I hope to have a report on just how compatible the clones are. He has bought every one made (some 30 to 40 units) and tried every combination possible. He says that all but one ran IBM stuff out of the box. That one worked after changing the ROM. His words are "only dealer support is the difference between them", which means being able to take it back should it fail in the first 90 days (any new item can fail, even Big Blue).

In building un-kits very little experience and skill is needed, especially with the clones. One advantage of the clones is the basic mother board requirement that it must run all boards that orignial IBM PC's can. This requirement has been lacking in other systems before they were standardized. The S-100 system went through many years of non-compatible products before the standard was created. Let's look now at just what problems those non standard years produced.

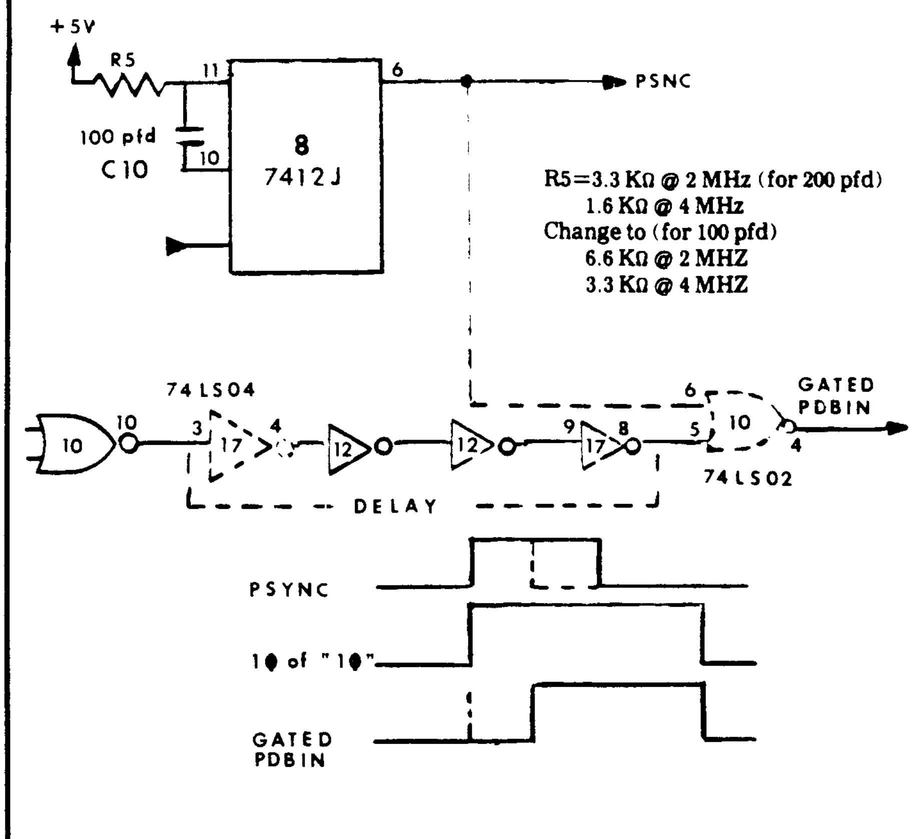
KITS

I put kits into a broad range of system types. Some kits are not much different than the un-kits, it is just the basic pc boards have not been stuffed and soldered. I have mentioned that I considered my Z100 as a un-kit, so I actually do not consider soldering one board as much of a construction project. However most Heathkits should be considered more of a kit than an un-kit. The real dividing line comes in how much adjusting or system integration work is needed. The un-kits will not require any fancy tools or skills to assemble. All parts will have been pretested and adjusted, which has not been done with kits. This definition means people without some knowledge of what they are doing should not attempt the construction. I put Heathkit in the un-kit becuse their adjustments can be accomplished without previous experience.

Using the skills level as a deciding line doesn't leave too many commercial products in this category. When the micro revolution started ten years ago, all systems were kits. The 68000 cards that developed from the articles in COMPUTER JOURNAL (sold by Intellicomp), are about as close to a kit as you can get. I say this becuse their system integration will be without a cook book of directions. The S-100 products are about the only kits left in which good skills will be needed. In my last big article I spoke of a friend who wanted to learn more by putting a set of used old S-100 cards together. I considered this a kit project and will show you why next.

These projects require good skills and previous experience, because most units were not designed to the now IEEE-696 S-100 standard. It is expected that some of the cards will not work with each other. The cpu was an Ithica Audio Z80 which needs some modifications to work with dynamic memory cards. The memory card had only 48K of the needed 64K for a decent system. One reason for the 48K of memory was the use of a memory mapped video card. This card was one of the first made and was good only for 60 by 16 character presentation. Almost all software and programs use 80 by 24 screens, so this card was not a good choice to use. There were two I/O cards to use, one quite old, the other a CCS 2716, a early product but sound in design. Lastly was the CCS 2422 disk controller with which I have considerable experience.

The changes started by making the cpu card have a proper PDBIN which must not start before the end of PSYNC. I have included the changes in figure 1, also watch for the proper timing cap/resistor ratio. I then added more memory and set the jumpers on the memory board for use with a phantom line. Many of the older boards didn't have phantoms, which must be installed if you have overlapping memory cards. The CCS 2422 uses phantom for the BOOT ROM and will need its jumpers set properly. There are two versions of the 2422, version A and B. The A version is difficult to interface and will require many changes if not used with its mate the CCS 2820. The B version is less trouble and will work with most CPU's but could require some changes to the bus driver chips. The 74LS bus driver chips will sometimes be inadequate in both speed and drive current if there are lots of cards in the system. The use of 74S chips improves both speed and drive



Ithaca audio Z80 PDBIN circuit and changes Rev. 1.0

current. This change will not improve all interface problems. Checking the manufacturers specifications is needed before making changes like this, as some chips change more than speed between types.

I further checked all the cards out by substitution in my own S-100 CCS system. This told me they all worked, and it was now only necessary to set up the I/O. I like the CCS 2716 becuse it has two separate serial chips. One has all software controlled options the other has all hardware selections. By using the hardware options I was able to emulate my own system and thus use the software without any changes. Had I not been able to do this, it would have been necessary to burn a separate PROM, then modify the software before it would work. Unfortunately, the documentation did not agree with the hardware and I spent several hours with a scope finding out which switch worked as stated and which did not. The card select jumpers were all backwards and the command data switch jumpers were half right and half wrong. This is probably the most common problem that will be encountered, incorrect documentation.

I have covered this system generation to show what is to be expected and what previous experience was needed in bringing it up. Had my friend actually decided to fix the unit (no previous S-100 experience), he would have spent over six months of complete frustration in getting it up and running. It is important when bringing up systems like this to have access to a second system. It takes a second system to test cards and write changes to software in hopes it can be made to run. A tool which I have just added to my S-100 collection is an EPROM emulator which can make the software part easier and as you will see in the component descriptions that follows, it may be absolutely necessary.

Component

I consider component systems to be those that require so much skill and equipment, that they should not be attempted by anyone but a sadist at heart. Fitting that category I decided to take on bringing up a Godbout CPU 68K, complete with a Forth type monitor system. When bringing up systems like this it may actually be easier to start from scratch. I have considered this and in fact may actually end up doing that. The reason behind this statement is knowing what the design consideration were and building only what is needed. When bringing up system from scratch it helps to have all the memory, I/O, and CPU on one card so at least their interfacing can be guaranteed to work. I bought the A68K kit from Motorala for $68 and it came with enough documentation (and 68K chips) that you could build a 64K memory system quite easily. The reason this type of project becomes so hard is the absence of good documentation. Godbout makes good products but their documentation leaves considerable to be desired. Their design is intended for a specific application (which is normal for all manufacturers) and of course my use does not fit their original design.

My first problem started with the 16 BIT wide PROM on the card. The CPU is a 68000 and has a 16 bit data structure. The Godbout card uses PALS to change the 16 bit data bus to 8 or 16 bits wide when talking to the S-100 bus. The on board ROM is two 8 bit 27XX type PROMs for 16 bits wide of data. This unit has 4 PALS which I hate dearly and one of which proved to have failed. I sent the card to the factory as older PALS were no longer available. For $75 they updated the board with new PALS and jumpers to make it equal to their current revs. I consider this a resonable price but the idea that I have to rely on them for ever to supply me with PALS (my most common failure item) is rather unnerving. This still didn't get the unit up and running, so I built an EPROM emulator from MICROSYSTEM JOURNAL. This required some changes in the design as it was intended for dual 8 bit emulation and not the 16 bits I needed. Appendix B lists those changes, however I had some other problems first.

The emulation board worked only half way. I found that my long path to trigger the address latch was not working. After scoping around for awhile, I decided that a small capacitor was needed on the address trigger line. This delayed the trigger until after the data was stable. I also found an unsoldered wire and after correcting that it worked just fine. This now allows me to write the software on one machine and have the changes become active immediadately without the long delays of erasing and burning PROMs. With a few important comments from a fellow club member and the emulator, I found out that the Godbout PALS were originally designed to have the ROM space declared as PROGRAM ONLY space. The 68K has status signals which can indicate the difference between data and program information. This normally is not much of a problem, but it is impossible to use it in the ROM space. It means that any time the CPU sees data the prom is turned off.

Once I knew of this problem I simply changed the code to NOT have any data information (a tricky task), and the system started to work. I called the factory back and they assured me the PAL changes should have corrected that. Since it hasn't I have given up trying to use the on-board ROM space and am now interfacing a ROM board into the system. What makes this project more difficult is that the project was not to do a hardware research on 68K but to do some software development. If I can't find the problems soon I will have to drop the cards and consider some other means of getting my software project going. A good option is the Atari 520/1040 systems. These inexpensive units can not use any of my extra S-100 cards as I was trying to do, but the cost of a complete and running system certainly out weighs the time lost fighting hardware problems.

Now I didn't intend for this project to be a component system project but the problems have sure pushed it into to one. I have had to use all my resources and skills just to know the system could work. Yet, I am still considerably away from the original project. There have been times when all the unit would do is go into a software HALT and without good test equipment I would have been completely lost. When building systems from scratch, you will need excellent test equipment, maybe even an logic analyzer. The advantge of modular designs however, do give you a chance to check cards out before hand and thus save considerable time.

APPENDIX A

The boards come in several configurations, usually all under $100. The Xerox 820-I comes in two versions, ETCH 1 and 2 (etch 2 has U117 filled). There is also a Xerox-II which has a separate board for the disk controller. The boards are all rather close in design, each being a slightly better version. The -I is a 64K 2.5MHZ Z80, single density (1771) using 4116's in the memory. The -II uses 4164 and a plug-in floppy or hard disk controller in single and double density. The boards are based on the Big Board I design and will run their software without modification. Their proms and monitor routines (BBI verus 820's) are almost identical, but can not be interchanged without several patches. Since most good programs will make all calls through the jump tables at the begining of the prom, it is possible to run each other's programs without changing proms.

The hardware to put these units together will be a keyboard, power supply, monitor, and disk drives. The keyboard is a parallel type and connects to J2 on the back panel. The pins are: pins 1 thru 8 are bits 0 thru 7 of data; pin 9 is strobe; pin 13 is +5V; pins 14 thru 25 are ground. Modifying the prom could allow a serial keyboard to be used on SIO B, such as is possible with the BBI. Another option would be a serial to parallel converter, similar to those used for printer interfaces.

The power supply should be a 5V at 3 to 4.5A, +12V at 1.8 to 2.5A, and −12V at 0.5A. Ratings will need to be higher if the same supply is used for the monitor or disk drives. The connections at J5 are: pin 1 is −12V, pins 2, and 3 are +12V, pins 4, 5, and 6 are ground, pin 7 is a separate +12V intended for the monitor supply, and pins 8,9 are +5V.

The video is positive going non-composite with separate positive going horizontal sync, and negative going vertical sync pulses. J7 is the video output connector with pin 3 as vertical sync, pin 4 horizontial sync, pin 5 video, and pins 6 thru 10 ground.

The disk drives can be either 5 or 8 inch. This is an advantage over the BBI (8 in only), but you can only use two or three drives depending on the etch model. Etch 1 uses three drives but does not have output for side select. Etch 2 uses the drive three line as side select. Both units have an input which is grounded (at the connector) for 5 inch drives. Etch 2 also has an input called 400/460, but is not explained or used anywhere (pulled hi). The pins are as shown below.

	8"	5"
pin 2 is 8/5 (ground for 5's),		
pin 4 is index,	20	8
pin 5 drive select 1,	26	10
pin 6 drive select 2,	28	12
pin 7 drive select 3 or	30	13
side select,	14	32
pin 8 is head load or motor on,	18	16
pin 9 is step direction,	34	18
pin 10 is step command,	36	20
pin 11 write data,	38	22
pin 12 is write gate,	40	24
pin 13 is track 00 input,	42	26
pin 14 is write protect input,	44	28
pin 15 read data,	46	30
pin 16 is 8 in tk 43 or low current signal,	2	
pin 17 is drive ready,	22	6
pin 18 is +12V,		
pin 19 is +5V,		
pins 20 thru 37 are ground.	all odd pins	

To use Xerox-II without disk controller card, add a pull-up resistor to pin 2 of U-19, the disk wait line.

The utility serial and parallel devices have jumper or header blocks for configuring to modems (SIO) or centronics (PIO) devices. There are many possible configurations and a schematic will be needed to determine the correct jumpers in all cases. It is not necessary to jumper any of these for bringing the system up. Connect the power supply and monitor to the board and after the power is turned on you should see the Xerox prompt.

APPENDIX B

From an article in Microsystems/Journal titled "BUILD AN S-100 EPROM EMULATOR" (Jan/Feb 1986). This unit is an S-100 wire wrapped card with six I/O ports which are used to set emulation mode, prom addresses, and to put data into the prom. It only uses 12 chips and should take about two or three days to fabricate. It will emulate up to 2764, although I suspect larger sizes are possible. The proms U7 & 8 (6264) have their address lines set by two LS273's which latch up the input address. Data is strobed into the proms by their respective input port after the address has been entered. Gates (81LS95) are used to control the output signals into a prom emulation cable. The incoming chip select signal and the prom emulated signal control the output side. Software was given in the article for using hex code from a disk file, but did not provide for changing code once it was loaded.

To use the circuit as a 16 bit wide EPROM the following changes are needed. I added a 74LS244 in the data path between U7 and U8 to allow data to go to U8 only. Enable this chip (1 & 19) with U8's WE* (pin 27). To get the data out use a 81LS95 just like the U11 arrangement. The quarter of U10 used to gate the output device U11 is not needed, take the two inputs that went to the AND gate and use them to drive the EW* line of each output driver. This isolates and provides two separate data paths but with common address paths. I found a better way to arrange the cable pinouts and it is listed below. I also use DDT as I can change code and then write it to the emulator, all within DDT. The code is also below.

CABLES: use 34 pin dual header strip and flat ribbon cable. Use flat ribbon DIP headers and connector for solderless connection. Pin 1 or the stripe will mark GROUND and be pin 12 of 2716/32.

```
    CABLE 1                  CABLE 2                DIP  HEADERS
                                                  2716/32       2764
    GND   1!2  D3          GND   1!2  D3           12!13        14!15
    D2    3!4  D4          D2    3!4  D4           11!14        13!16
    D1    5!6  D5          D1    5!6  D5           10!15        12!17
    D0    7!8  D6          D0    7!8  D6            9!16        11!18
    A0    9!10 D7                9!10 D7            8!17        10!19
    A1   11!12 CE*              11!12 CE*           7!18         9!20
    A2   13!14 A10              13!14               6!19         8!21
    A3   15!16 OE*              15!16 OE*           5!20         7!22
    A4   17!18 A11 #            17!18               4!21         6!23
    A5   19!20 A9               19!20               3!22         5!24
    A6   21!22 A8               21!22               2!23         4!25
    A7   23!24                  23!24               1!24         3!26
  # A12  25!26                  25!26                            2!27
         27!28                  27!28                            1!28

# Lines not used must be grounded to maintain proper
addressing on proms.

Assemble this code at 8000hex where it will be safe even if
DDT must be reloaded after a control C.

8000   21 00 01     LXI H,0100   ;START OF HEX CODE
8003   11 FF 07     LXI D,07FF   ;LENGTH OF CODE
8006   01 00 00     LXI B,0000   ;STARTING PROM ADDRESS
8009   D3 ?1        OUT ?1       ;BASE ADDRS +1, LOAD MODE
800B   AF           XRA A        ;SET ADDRESS TO ALL 0000
800C   D3 ?2        OUT ?2       ;SET LOW ADDRESS TO ZERO
800E   D3 ?3        OUT ?3       ;SET HIGH ADDRESS TO ZERO
8010   7E           MOV A,M      ;GET FIRST BYTE
8011   D3 ?4        OUT ?4       ;PUT IN U7 PROM  (EVEN)
8013   23           INX H        ;POINT TO NEXT BYTE
8014   7E           MOV A,M      ;GET NEXT BYTE
8015   D3 ?5        OUT ?5       ;PUT IN U8 PROM (ODD)
8017   23           INX H        ;POINT TO NEXT BYTE
8018   03           INX B        ;POINT TO NEXT ADDRS
8019   79           MOV A,C      ;GET NEW LOW ADDRESS
801A   D3 ?2        OUT ?2       ;PUT NEW LOW ADDRS OUT
801C   78           MOV A,B      ;GET NEW HIGH ADDRESS
801D   D3 ?3        OUT ?3       ;PUT NEW HIGH ADDRS OUT
801F   BA           CMP D        ;SEE IF HIGH END OF CODE
8020   C2 10 80     JNZ 8010     ;JUMP IF NOT MATCH
8023   79           MOV A,C      ;REGET LOW ADDRESS
8024   BB           CMP E        ;SEE IF LOW BYTE A MATCH
8025   C2 10 80     JNZ 8010     ;JUMP IF NOT MATCH
8028   D3 ?0        OUT ?0       ;SET INTO EMULATE MODE
802A   FF FF FF     RST 7        ;REENTRY CODE INTO DDT

"?" IS BASE ADDRESS OF EPROM EMULATOR....
```

Final Words

In writing this article, I wanted to indicate some of the problems as well as concepts that one must have in building systems. These concepts deal with system design as well with product modifications. As computers become more involved in handling real world problems the need for you to be able to make the interface and system modifications also becomes important. To make these changes, you must develope skills. These skills can come from practical experience in integrating a system. Recent price drops in systems will find many of our readers considering non data uses for computers. My orginal entry into computers was solar system control. Single chips now handle most solar systems, but for data and control applications, forth based Xerox 820's look like very possible and cheap solutions.

As can be seen in the component description, equipment will also be needed to properly check out changes. I had been wanting to build an EPROM emulator for some time and the availabilty of the article and needs for one got me to do it. I was trying to make up my mind as to what 68000 system I would use for my Forth project, when a friend offered me the Godbout 68K card for what I considered a real bargain. My first choice of the Intellicomp 68008 board (advertised in TCJ) now appears to be a better bargain for my use. Had I not had so many S-100 cards already, the Atari units are by far the best buy. What this shows is my false conception of a bargain. Time is always your most expensive product, and never having enough time, I have found it cheaper to buy running systems for the basic unit and modifiy them, than to build them from scratch. ■

The SCSI Interface

SCSI Command Protocol

by Rick Lehrbaum

Introduction

In this part of The Computer Journal's series on the Small Computer System Interface (SCSI) we will focus on the software side of SCSI. We'll discuss the structure of SCSI commands in general, and in the next part we'll take a look at a specific SCSI driver.

We recommend that you obtain a copy of the ANSC X3T9.2 SCSI draft standard. It is available for $20 per copy from:

The X3 Secretariat
Computer and Business Equipment Manufacturers Assn.
311 First Street, N.W. - Suite 500
Washington, DC 20001

(Please include a self-addressed mailing label.)

An Intelligent Bus

As mentioned earlier in this series, SCSI is an "intelligent" bus. SCSI not only defines a very precise hardware interface, but also provides great detail on the command sets for many classes of SCSI devices.

Why is SCSI called an "intelligent" bus? SCSI peripheral devices, called "Targets," accept high level commands, and pretty much mask the characteristics of the actual attached physical devices. The host computer, called an "Initiator," deals with a "logical" rather than a "physical" device over SCSI, with the translation between "physical" and "logical" being handled by software resident on the SCSI device controller.

It is this combination of hardware and software standardization that makes SCSI so useful. For example, changing from one SCSI hard disk drive (or controller) to another might only require you to make a few minor changes to the drive format utility. The fact that any change at all is required stems from the allowances built into the SCSI spec which permit "vendor unique" capabilities and features.

A Typical Command Set

The "logical" interface to the SCSI device is defined by the SCSI command set for the device. SCSI defines several types of devices:

Direct Access Devices (e.g. disk drives)
Sequential-Access Devices (e.g. tape drives)
Printer Devices
Processor Devices
Write-Once Read-Multiple ('WORM') Devices (e.g. optical storage)
Read-Only Devices (e.g. CD ROM)

A particular SCSI Target's command set depends on the type of device, but you can get an idea of the typical functions from the list of commands for Direct Access Devices as shown below:

MANDATORY COMMANDS:
Request Sense
Format Unit
Read
Write

OPTIONAL COMMANDS:
Test Unit Ready
Rezero Unit
Reassign Blocks
Seek
Inquiry
Mode Select
Reserve
Release
Copy
Mode Sense
Start/Stop Unit
Receive Diagnostic Results
Send Diagnostic
Prevent/Allow Media Removal

Most of the commands in this list are self explanatory. As you can see, only four very basic commands are actually required. A brief description of a few of these commands will help explain the nature of the "logical" interface to a disk drive provided by SCSI. Again, these descriptions apply to one type of SCSI Direct Access Device, namely hard disk drives.

Playing With Blocks

Data is transfered between Target and Initiator over the SCSI bus in what are called "blocks." A block consists of between 1 and 16,777,216 bytes of data, though the block length choices available with any particular controller are limited. Typical block lengths used are 256, 512, and 1024 bytes, though the Apple Macintosh PLUS® uses a block size of 530 bytes. Selection of the block length, if the Target provides a choice at all, is accomplished with either jumpers on the controller or (preferably) using the optional Mode Select command.

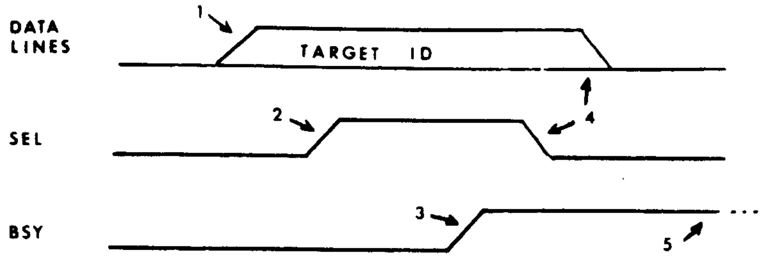
SCSI commands treat the disk drive as a stream of "blocks" of data, using a block number to designate the location of th data rather than specifing head and cylinder. The SCSI Initiato may have no knowledge at all as to how the blocks are actuall distributed on the magnetic media itself. The first block migh not even be on the first cylinder of the media. The drive' physical characteristics other than size, including the number c heads and cylinders, step rate, etc., may not be known to th Initiator at all!

The READ and WRITE commands mainly specify a starting block number and the number of blocks of data to transfer. The mandatory REQUEST SENSE command is used to obtain error information in a standardized format.

The FORMAT command, on the other hand, is the most variable of all SCSI commands. This is because it is often the FORMAT command which informs the controller of the physical characteristics of the device being controlled. On the other hand, once the FORMAT command has been run, many SCSI controllers store the physical parameters of the device either on the controller or on the device (such as in the first few disk sectors). These "self-initializing" SCSI Targets become perfect "black boxes" once formatted.

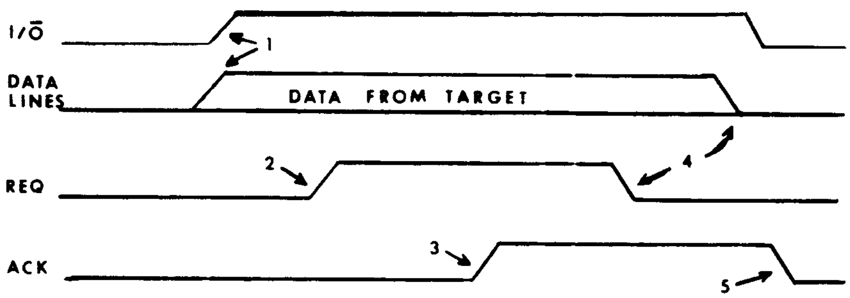
A particularly nice optional SCSI command is the COPY command. As you might guess, a combination disk/tape controller might really make disk data backup easy!

Figure 1: SCSI External Timing Diagrams.



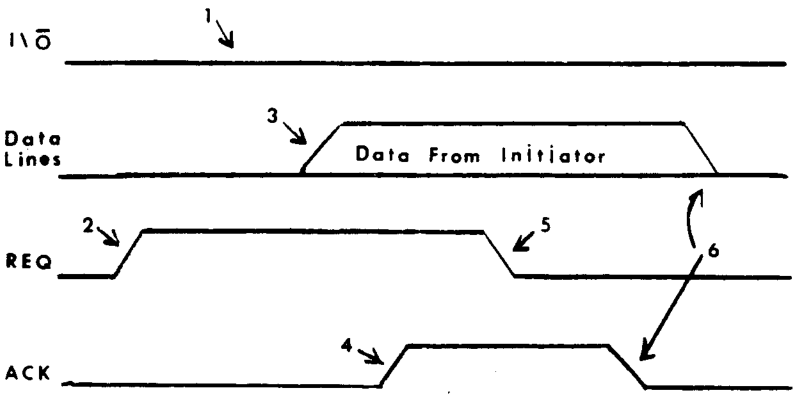
SCSI Selection (Non-arbitrating).

1) Initiator places Target ID on data lines.
2) Initiator sets SEL active.
3) Target sets BSY when it knows it is "selected."
4) Initiator clears SEL, then removes Target ID from data lines, when it sees BSY asserted by Target.
5) Target continues to assert BSY for duration of transaction



SCSI Data Transfer From Target

1) Target places data on the data lines, and sets I/$\overline{O}$ active.
2) Target sets REQ.
3) Initiator reads the data, then sets ACK.
4) Target clears REQ when it sees Initiator's ACK, and can then remove data from data lines.
5) Initiator clears ACK when it sees Target's REQ.



SCSI Data Transfer To Target

1) Target leaves I/$\overline{O}$ inactive.
2) Target sets REQ.
3) Initiator places data on the data lines when it sees Target's REQ.
4) After placing data on the data lines, Initiator sets ACK.
5) Target reads data from the data lines, and then clears REQ.
6) Initiator clears ACK and removes its data from the data lines, after sees REQ go away.

Doing It The SCSI Way

In an SCSI transaction between an Initiator and a Target, the Initiator starts the process by "Selecting" the Target. Once the Target responds, however, the Target controls the remainder of the command sequence. A future segment of TCJ's SCSI Series will include an example of actual software to permit this to occur.

These are the steps which occur in an SCSI command sequence:

Selection Phase

Prior to transfering data to or from the SCSI Target, the Initiator must "Select" the Target. There are two types of selection, depending on whether the SCSI Bus has multiple Initiators, or not. In the former case, a bus arbitration phase must be completed to gain bus access.

Assuming no bus arbitration is required, the Initiator asserts the Target's SCSI ID bit on the SCSI data lines, and then activates the SELECT line. When the Target detects this condition, it responds by activating the BUSY line, which indicates to the Initiator that the desired Target has become selected.

Once the Initiator observes the Target's response (BUSY becomming active), it releases the SELECT signal and removes the Target's ID from the SCSI Bus data lines.

Command Phase

Each SCSI command consists of several bytes. The first byte of the command is called the Operation Code, the last byte of the command is called the Control Byte, and the intermediate bytes have varying functions depending on the command. As an example, the typical READ command sequence for Direct Access Devices is shown in Figure 2.

	Bit							
Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	Logical Unit			Logical block adr (MSB)				
2	Logical block adr							
3	Logical block adr (LSB)							
4	Transfer length							
5	2 VU bits		4 reserved bits				Flag	Link

Figure 2 (Read command sequence)

Once the Target has been selected by the Initiator, and the Initiator has released the SELECT signal, the Target then requests a command from the Initiator by setting the C/D signal to C (for Command phase), setting the I/O signal to O (out), and activating the REQ (request) signal.

When the Initiator observes that the REQ signal is active, it places the first byte of the command on the data lines. When the Initiator senses the REQ signal from the Target, it reads the command byte from the data lines and then activates the ACK signal, which signals the Target that the command byte has been received.

This command byte transfer sequence, with REQ/ACK handshaking, continues until the appropriate number of command bytes has been transferred. It is the Target which controls the number of command bytes transferred in the Command Phase, based on the interpretation of each Operation Code, which is the first command byte transferred following selection of the Target by the Initiator.

Data Phase

After the Target requests and receives the number of command bytes appropriate for a given Operation Code, it then changes the C/D line to D (Data) to indicate that the bus is now entering the Data Phase.

The byte transfer process is similar to that described for the Command Phase, except that the C/D line is held in the D state by the Target and the I/O line is set to I or O depending on whether the direction of data transfer is from Target to Initiator ("In"), or Initiator to Target ("Out").

The number and direction of bytes transferred during the Data Phase is controlled by the Target, and depends on the content of the command bytes previously sent by the Initiator.

Status Phase

Following the transfer of an appropriate number of bytes of data in the Data Phase, the Target next changes the C/D line back to C (Command) and sets the I/O line to I, which signify that the bus is now in the Status Phase. In the Status Phase, the Target transfers a single byte of status information to the Initiator (again using the REQ/ACK byte transfer protocol).

The SCSI spec defines 16 standard status byte codes, including such functions as "GOOD," "CHECK CONDITION," "BUSY," and several others. When CHECK CONDITION or other bad status is returned, the Initiator will normally follow completion of the current command sequence with a SENSE command to determine the cause of the error.

Message Phase

All normal SCSI commands terminate with a single byte Message Phase. To signal the Initiator that the Message Phase byte is being transferred, the Target sets the C/D line to C, activates the MSG line, and sets the I/O line to I. The Target then uses the usual REQ/ACK byte transfer protocol to transfer the message byte to the Initiator.

Most Targets terminate all of their SCSI command sequences with a "Command Complete" message, which contains the value 00H.

Bus Free Phase

After completion of a bus transaction—including Selection, Command, Data, Status, and Message phases—the Target ends the transaction by releasing all signals. The BUSY line, which has been held active by the Target since the Selection phase, is included among the signals released at this time by the Target. This signifies the end of the current SCSI Bus transaction.

Read Command Example

The SCSI Direct Access Device READ command will be used as an example of how SCSI protocol operates.

This command requests a "block" of data from a drive. Six bytes of command information are transferred from the SCSI Initiator to the SCSI Target prior to the data phases of the command sequence. These include: the command operation code, the Logical Unit Number (LUN) of the drive (one controller can have up to 8 drives), the starting block address of the data to read, the number of blocks to read, and a control byte (often 00H).

The READ command sequence consists of the six command bytes shown in Figure 2.

The Logical Unit is a binary value between 000 and 111, indicating one of eight possible drives connected to the controller (most only actually allow two). Bits 6 and 7 of the fifth byte are for Vendor Unique purposes and are generally 0's. The Flag and Link bits have special functions (relating to SCSI command linking) which will not be discussed at this time.

More To Come...

In the next segment of this series we will look in detail at an actually working SCSI driver. You'll see how software can be constructed which takes advantage of the full power and flexibility of SCSI. Later on, we'll cover an actual hardware design example. ■

Introduction To Assembly Code for CP/M

Adding A CLS Function

by Walter E. Pfiester

Introduction

By the end of this article you will be able to assemble a short piece of code that will clear the screen and home the cursor with one command. So what? MSDOS has the CLS function built in. Well, CP/M doesn't unless your operating system has been modified, and you needn't modify the operating system for this piece of code.

Writing The Code

Use any word processor (if it's WordStar®, use the NON DOCUMENT mode) to type in the code shown in Figure 1. I prefer to use VDO which is public domain, and very very fast!

```
ORG  100h   ; Assembly routine to return
PUSH B      ; to the A> prompt without
PUSH D      ; WARM BOOTing every time
PUSH H      ;
MVI  C,2    ; Begin of CLS, home routine
MVI  E,1Ah  ; Put a ^Z in register E **
CALL 5      ; List the output
MVI  C,0h   ; End of CLS routine
POP  H      ; saves the CP/M environment
POP  D      ; put this on your disk as
POP  B      ; the 1st program (with PIP)
RET         ; and it'll execute fast!
```

Figure 1

The portions of the lines following the semicolons are comments, and will not be included in the assembled program. If your terminal uses other than the ^Z to clear the screen, then change the code in the line marked with ** to your terminal's atributes. This program will work with most KayPro's and Osborne's with no changes. Tribute should be given to the KayPro magazine OnLine (disk supplied) for the concept of resetting without warm booting the system and to the First Osborne Group for the program concept.

But so what? isn't the string ^Z<esc><cr> the same thing? Yes! But this program doesn't depend on your fingers being contortionistic in nature. Besides which, it can be submitted (that is, used in automated sequences). Shouldn't it be in the operating system itself? Maybe so, but that's another article in itself.

Assembling The Program

To assemble this program you'll need ASM.COM and LOAD.COM supplied with your CP/M operating system by Digital Research. Simply type: ASM CLS. Then LOAD CLS. Three files will result, CLS.PRN, CLS.HEX, and CLS.COM. You may erase the .PRN and .HEX files.

How To Utilize

Put C.COM on a disk as the very first file (use PIP.COM to do this). To be sure that it is the first file, type DIR. The directory will show which file is first (upper left) and proceed from left to right, top to bottom. Do not use one of the superdirectory programs for this (we don't want the directory sorted). The reason for this is that in order to minimize the execution time the disk head must travel the absolute minimum amount of space on the disk. By the way, this concept should be used on your "A" disks for WordStar, dBase II, etc. Minimize the search time for the most used programs.

Examining The Results

By comparing what you have, in Hexidecimal, to the source code, a one to one correspondence should result. An easy way to see the exact code is to DUMP the program with any of the dump programs used for this purpose. I prefer "PATCH18.COM", available in the public domain. I could not demonstrate the use of it since input/output redirection is impossible with this program because the author set the end address too high in the CP/M operating system. The illustration in Figure 2 was taken from EDFILE.COM, a close second to PATCH18.

In this case the 16 bytes match the assembly code you typed in exactly match the machine code. For more information on just what the registers used are for, I recommend the book, "Soul Of CP/M" by Waite and Lafore (Howard W. Sams, $18.95). This is the best text I know of for getting started in assembly code programming.

You just disassembled this program "by hand." Try disassembling the program with DDT, supplied with CP/M by Digital Research also. Put DDT.COM on the same disk as C.COM.

Type: DDT CLS.COM, then at the - (minus) prompt type L100,0115. DDT will dissamble (list) this program for you. The results are shown in Figure 3.

```
Vers: 01-10-84; by: J.C.Kaltwasser & M.J.Mosko, K3RL
File: CLS.COM  Record: 00000 (0000H) LOF: 00001 (0001H)
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      0123456789ABCDEF
          -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --      ----------------
0100   -  C5 D5 E5 0E 02 1E 1A CD 05 00 0E 00 E1 D1 C1 C9     >EUe....M....aQAI<
0110   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0120   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0130   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0140   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0150   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0160   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
0170   -  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
```

Figure 2

```
B>DDT CLS.COM
DDT VERS 2.2
NEXT  PC
0180 0100
-L0100,0115
  0100  PUSH  B
  0101  PUSH  D
  0102  PUSH  H
  0103  MVI   C,02
  0105  MVI   E,1A
  0107  CALL  0005
  010A  MVI   C,00
  010C  POP   H
  010D  POP   D
  010E  POP   B
  010F  RET
  0110  NOP
  0111  NOP
  0112  NOP
  0113  NOP
  0114  NOP
  0115  NOP
  0116
```

Figure 3

Conclusions

Not all programs can be dissambled this way, only those written with 8080 nmenonics. Other assemblers and disassemblers can be used in a like manner (in pairs). I suggest that you write a small piece of code and assemble it before trying disassembly.

You can write to me (Walter E. Pfiester) in care of THE COMPUTER JOURNAL or direct to my home at 1 Skadden Terrace, Tully, N.Y. 13159. Please include a self-addressed, stamped envelope or a disk formatted to KayPro IV along with a stamped return mailer. I do not respond to mail without a SASE. ■

The C Column
Software Text Filters
by Donald Howes

Well, I'm back again, a little late getting this column in to Art, but better late than never (as they say). There's a good reason for the delay, although I've got mixed feelings about it. I've sold my Compupro system and bought an IBM clone (a no name generic).

A Short Requium

This is a move I've been contemplating for some months, mainly due to business considerations. I do a fair amount of consulting work and it was getting embarrassing not to know much about MS/PC-DOS. Also, my interests are moving into the rather broad area of interface design, so I'm getting more involved in the nitty gritty of systems programming for graphic interfaces. This just wasn't possible on my S-100 system. At least, not without the addition of a rather expensive graphics board and the purchase of a new monitor which the board could drive. With the new PC system, I've spent just about as much on the monitor and EGA board (a NEC multisync and a Quadram EGA+) as I did on the rest of the hardware but have a 640K system with a 20M hard drive for just about $2100.00. What with it on one side of the room and my Atari 520 ST on the other, you should probably expect to be hearing an increasing amount about all types of graphics programming in this column (stay tuned for further developments).

On the down side, I have been programming in the CP/M environment for the last four years and I've gotten pretty comfortable there (sort of like a favorite pair of shoes). I guess that I should look at the changes as a challenge, but, at times, I do miss having the old S-100 boat anchor taking up half of my desk (even if it's only because I was able to heat the house with it during the winter). Enough of that, on to bigger and better (?) things.

A High Pass at Filters

Ok, so that's a bad play on words. But software filters do share common attributes with electronic filters. In general, a filter monitors the data being passed through it and, either on every byte or on bytes which meet the target characteristics defined in the filter, performs an operation on the byte. As promised last time, we'll be looking at three simple filters that are very usefui for any text processing applications you may have in mind. These are: a simple little program which strips off the high bit of a byte, so that files produced by text processors which set bit seven (such as WordStar, the program I'm using to write this column) can be displayed on the screen or dumped to a graphics printer without driving the device crazy; a program for removing tabs from a file and replacing them with blanks; and the inverse program which will remove blanks from a file and replace them with tabs.

Strip.c

This straight forward little program strips bit 7 from each byte by performing an additive bit mask using the value 7F hexadecimal (in binary that is 10000000). This will dump the high order bit off into the bit bucket and allow the byte to be read as a standard ASCII character. The compiler I'm using implements the function toascii() as both a library function and as a defined macro. I am using the macro definition version of the function (although technically it is not a function if it's implemented as a macro) which is found in the header file "ctype.h". Check your own compiler to see if this header file is present. If it is, it's a better idea to use the macro instead of the library function. The reason for this is that the macro definition code for toascii() will be inserted in-line in your code by the preprocessor pass of your compiler. On a program like strip.c, which passes every byte of the file to toascii(), this will save you the system overhead of many thousands of calls to a function and will speed up the execution of the program.

One thing that people may have picked up on in the previous sentence is that the entire file will be passed through the filter and each byte will be masked out. Since every byte of the file will not have bit 7 set high, it would probably seem a much faster approach to only test for the situations where the bit is set and pass only the effected bytes to the filter. The answer to that is both yes and no. While I haven't done any tests myself, I can report on one done by the journal \ C (June, 1985, page 2) where they performed tests using a 55K WordStar file. They found that the program which only stripped the high bit on bytes where it had been set ran only one percent (!) faster than the program which ran every byte through the filter. The time gained by only processing needed bytes is apparently lost in the logical comparison needed to determine if the byte should be sent to the filter (you win some, you lose some).

Let's take a look at the program (Listing 1). Since this is such a simple program, I haven't done any creation of functions (to handle such things as the generation of error messages) and everything is done in a very linear fashion. The program uses command line arguments to indicate the names of the input and output files. So that these can be passed to the program the main() function must be declared using the "argc" and "argv" parameters. The only thing which may be a little confusing is the declaration for argv. Using the right-left rule, this declaration can be parsed to mean that "argv is an array of pointers to characters." These two parameters contain the number of command line arguments (argc) and the contents of each argument (argv).

Listing 1. Source Code for Strip.C

```
/*
 strip.c

    Copyright 1986  Donald Howes
           All rights reserved.

    This program may be copied for personal, non-commerical use
    only, provided that the copyright notice is included in all
    copies.

 This program masks out bit 7 for each byte, stripping the sign
 byte and creating standard ascii files.

 This program should only be used with text files.

 usage : strip <infile> <outfile>

 created : Version 1.0 August 14, 1985
        : Version 1.1 May 4, 1986
 */
```

```
#include     <stdio.h>
#include     <string.h>
#include     <ctype.h>

main(argc,argv)
int argc;
char *argv[];
{
    int c;
    FILE *fd1, *fd2, *fopen();

    if (argc != 3)
    {
      printf("Usage: strip <infile> <outfile> (CR)\n");
      exit();
    }
    if (strcmp(argv[1],argv[2]) == 0)
    {
      printf("Output file must be different from input file.\n");
      exit();
    }
    if ((fd1 = fopen(argv[1],"r")) == NULL)
    {
      printf("Can't open %s\n",argv[1]);
      exit();
    }
    if ((fd2 = fopen(argv[2],"w")) == NULL)
    {
      printf("Can't create %s\n",argv[2]);
      exit();
    }
    while ((c = getc(fd1)) != EOF)
    {
      if (putc(toascii(c),fd2) == EOF)
      {
        printf("write error, disk probably full\n");
        exit();
      }
    }

    fclose(fd1);
    fclose(fd2);
}
```

Within main(), the first thing done is to declare variables. Notice the declaration of fopen() here. This must be done within the program, since fopen() returns a pointer of type FILE which is the file descriptor for the opened file. This file descriptor is used by other functions to refer to the opened file (rather than the filename of the file). The function must be declared, or the C compiler will default to an expected return value of int. This will result in a type mismatch which may or may not be detected by your compiler. If the mismatch isn't detected, this will result in a very difficult bug to root out of your program.

The next thing done is a set of simple error checking routines and their associated error messages. The first routine checks to see if the number of arguments passed to main() is equal to three. Why check for three passed arguments instead of two? The reason is that the first argument passed is unrelated to the program being run. Depending on the compiler being used, the first argument (which would be argv[0]) could be the operating system, the version of the compiler, or possibly the name of the executing program. After checking for the correct number of arguments, we next check to see that the input file and the output file names are different. If this wasn't done, there would be a terrible mess, since the output file is streamed back to disk once the input buffer is filled. This would truncate the input file to the size of the buffer. Also, this is a simple way to take care of possible stripping of binary files, which shouldn't be done, since all the bits in a binary file are significant. However, all you get with this program is an unusable output file.

Now we attempt to open the input and output files. The input file is opened in read mode, while the output file is created and opened in write mode. For the output file, you may have to modify the program, if your compilers implementation of fopen() does not also create the file. Your compiler will have a creat() function, which will handle the creation of the file and the allocation of the file descriptor. The file can then be opened using fopen(). One thing which will have to be remembered is that fopen() will truncate the output file to zero length if the file already exists. Make sure that the output filename does not already exist. Also, your compiler may return ERROR rather than NULL on an error from fopen(). Check your compiler documentation.

Once the files are open, we use getc() to stream bytes in from the input file (checking for EOF which indicates end-of-file) and putc to write the byte stream out to the output file. Note the call to putc(). While this version of toascii() is a macro, using a function as one of the arguments to another function is perfectly valid and provides for a more concise coding style. If there isn't an error generated by putc() trying to write to disk, we call fclose() to close both the input and output files. The call to fclose() will flush the I/O buffer before closing the files. This causes any bytes that are present in the buffer to be written to disk (the buffer is normally only dumped when it is full).

Entab.c

This filter will remove spaces from within a text file and replace the spaces with a tab character. The number of spaces collected is under the users control. Both this filter and the following detab filter are presented in a "demonstration" format. They are actually better used as functions found within a larger program but, for our purposes, it was better to write them as stand alone programs to demonstrate their operation.

Entab (Listing 2) opens with a series of includes and defines. The defines are for constants used within the program and it is customary to write the constant name in uppercase (both for the programmers convenience and since the C language is case sensitive). These constants are replaced in-line within the code by the compilers pre-processor pass. You will notice the declaration for the array err_mess[]. Using the right-left rule again, we have an array of pointers to characters, similar to the declaration of argv[]. However, here we initialize the err_mess[] array with four strings and declare the array to be static. The static variable declaration may be one you haven't seen before, and means that the declared array is permanen The four declared array strings may not be modified within th program after they have been initialized.

A large part of the code in both the entab and detab filters i error checking code (remember what I said last time abou bullet proofing). Luckily, the error code is the same for bot programs, so we will only have to go through it once.

We start out by checking that a filename (at least) has bee passed to the program. If not, the program will print an error message and exit. It is a good idea to code for a minimum number of acceptable command line entries, unless you will be accepting input within the program itself, rather than letting the program run for a while and then discovering it doesn't have the proper information. Usually, for utility programs of this type, it is best either to take the course of having all your options entered at the command line or having the necessary data prompted for from within your program. This will simplify your coding task.

Note that, unlike the code for strip.c, I have removed the error messages from the main() function (they are now in err_mess[]) and have created a p_error() function which handles the display of the appropriate error message. this goes back to what I was talking about last time about functions being black boxes. The p_error() function has all the code necessary to display the error message and I don't need to know any of the details of the internal coding of that function (as long as it works correctly). Also, by writing a separate function to display ap-

Listing 2. Source Code for Entab.C

```
/*
                entab.c

 This is a translation of the Ratfor program 'entab' found in
 Kernighan and Plauger "Software Tools", pp. 35-37.

usage: entab <infile> [-n]

*/

#include   <stdio.h>
#include   <ctype.h>

#define   MAXLINE    132   /* maximum line length, this will fit
                             a wide carriage printer */
#define   MAX_TABLINE (MAXLINE + 1) /* max search space for tabs */
#define   DEF_TAB    8    /* default tab stop spacing */
#define   YES        1
#define   NO         0

static char *err_mess[4] = {
    "Usage: entab <infile> [-n]",
    "Invalid call to entab. Usage: entab <infile> [-n]",
    "Invalid argument: [-n] must be a digit",
    "Invalid file name"
};

main(argc,argv)
int argc;
char *argv[];
{
  int col, new_col, i, tab_spc,
     tabs[MAX_TABLINE],  /* tab positions */
     *str;          /* scratch string pointer for parameter
                       line */
  char filename[13],    /* buffer for input filename */
     c;

  FILE *fopen(), *fp;

  if (argc < 2)  /* check that something has been entered */
  {
    p_error(0);
    exit();
  }

  if (argc > 3)  /* check for a valid number of arguments */
  {
    p_error(1);
    exit();
  }

  if (*argv[1] == '-')  /* check for correct order of command
  {                        line entry */
    p_error(1);
    exit();
  }

  /* parse the command line input */

    if (argc == 3)           /* the second argument is */
    {                        /* the filename, third is */
      if (strlen(argv[1]) <= 12)  /* tab spacing        */
        strcpy(filename,argv[1]);
      else
        p_error(3);
      if (*argv[2] == '-')
      {
      str = argv[2] + 1;
        if (isdigit(*str))  /* make sure that the argument */
        {             /* is a digit */
          tab_spc = *str++ - '0';
          if (*str != NULL)     /* if there's more */
            if (isdigit(*str))  /* is it a digit */
              tab_spc = tab_spc * 10 + *str++ - '0';
            else
            {
              p_error(2);  /* abort with error message */
              exit();
            }
        }
        else
        {
          p_error(2);    /* abort with error message */
          exit();
        }
      }
      else
      {
        p_error(2);
        exit();
      }
    }
    else  /* only the filename is given on the command line */
    {
      if (strlen(argv[1]) <= 12)
      {
        strcpy(filename,argv[1]);
        tab_spc = DEF_TAB;  /* use default tab spacing */
      }
      else
        p_error(3);
    }
/****** the following is the code for entab ******/

   fp = fopen(filename,"r");
   settab(tab_spc,tabs);

   col = 1;

     while (TRUE)
     {
       new_col = col;
       while ((c = getc(fp)) == ' ')
       {
         ++new_col;
         if (tabpos(new_col,tabs) == YES)
         {
           putc('\t',stdout);
           col = new_col;
         }
       }
       if (c == '\t')
       {
         while (tabpos(new_col,tabs) == NO)
           ++new_col;
         putc('\t',stdout);
         col = new_col;
         continue;
       }
       }
       while (col < new_col)
       {
         putc(' ',stdout);
         col++;
       }
       if (c == EOF)
         break;
       putc(c,stdout);
       if (c == '\n')
         col = 1;
       else
         col++;
     }

   fclose(fp);
   fclose(stdout);
}  /* end of main */

int p_error(erno)
int erno;
```

```
{
  printf(err_mess[erno]);
} /* end of p_error */

int settab(t_space,array)
int t_space,array[];
{
  int i;

  for (i = 1;i < MAXLINE;i + + )
    if ((i % t_space) = = 1)
      array[i] = YES;
    else
      array[i] = NO;
} /* end of settab */

int tabpos(col,array)
int col,array[];
{
  if (col > MAXLINE)
    return YES;
  else
    return array[col];
} /* end of tabpos */
```

Listing 3. Source Code for Detab.C

```
/*
          detab.c

 This is a translation of the Ratfor program 'detab' found in
 Kernighan and Plauger "Software Tools", pp. 18-27.

 usage: detab <infile> [-n]
*/

#include  <stdio.h>
#include  <ctype.h>

#define   MAXLINE   132  /* maximum line length, this will fit a
                   wide carriage printer */
#define   MAX_TABLINE (MAXLINE + 1) /* max search space for tabs */
#define   DEF_TAB   8    /* default tab stop spacing */
#define   YES       1
#define   NO        0

static char *err_mess[4] = {
    "Usage: detab <infile> [-n]",
    "Invalid call to detab. Usage: detab <infile> [-n]",
```

plication error messages, it allows me to separate the error messages (which will probably change from one application to the next) from the function designed to display those error messages. This lets me write modular and highly portable functions which can be applied in a number of different situations. Coding of this type is at the heart of what is generally termed structured programming.

After checking that the minimum allowable number of command line options has been entered, we now check that there aren't any extra. If that test is passed, I check that the arguments have been entered in the proper order, with the filename preceding the optional switch for declaring the number of blanks to be collected before a tab is output. It is traditional (descended from Unix) that command line switches are preceded by a minus sign ('−'), generally without any space between the minus sign and the following switch. These command line switches can be any alphanumeric combination you wish. Here, I have used a numeric value to indicate the number of blanks collected. To simplify the coding task, I have decided that the filename must precede the command line switch. Since this switch is optional, allowing for either possible input arrangement would have greatly increased the amount of code necessary to make the program work.

Once the basics are taken care of, we parse out the command line. If the number of arguments is three I know that both a filename and switch are present. I check that the filename is no more than 12 characters long (8 for the filename, a period and 3 for the file extension) then copy the string from argv[1] into the array filename[]. Notice that filename[] is declared as being 13 elements long. This is needed so that the null string terminator ('\0') can be copied into the array if the total length of argv[1] is 12 (a call to strlen() returns the length of the string less the null terminator). I then parse the switch value (which can have a maximum length of two), checking that each of the two possible input characters is actually a digit [using the macro isdigit()]. The value determined here is assigned to the variable "tab_spc." If the value of argc is two, then only the filename was passed to the program. I again check that the maximum length of the filename is not exceeded and copy the string from argv[1] into filename[]. Since the command line switch was not used, I assign the value DEF_TAB (the default tab spacing declared at the beginning of the program) to the variable "tab_spc."

Finally we get to talk about the code which actually does the filtering. After opening the input file, there is a call to the function settab(), which sets tab stops in the tab[] array. The program proper runs within an infinite **while** loop, which continues until an EOF condition is reached in the input. First inside this loop is a **while** test which is executed if the character read from the input is a space. If it is, there is an internal **if** conditional which calls the function tabpos(). This function keeps a count of where the character stream is in the line and whether a tab position has been reached. If one has, a tab character ('\t') is output. Following the **while** is an **if** test to deal with the special case of tab characters possibly present in the input stream. If an input character is a tab, the column counter is incremented until the next tab position is reached and a tab character is output. The next **while** loop checks to see if there are any left over blanks. If there are, then they must be output before any non-tab, non-blank character. The output character is then checked to see if it is EOF and the infinite **while** loop is terminated if the test is true. Otherwise, the input character is output and, if it was a new line character ('\n'), the column counter is reset to one, else it is incremented. When EOF is reached, the input and output files (here the output file is the standard I/O device "stdout") are closed.

detab.c

For this filter, I will only be talking about the code specific to detab, since all of the error checking code is the same as that found in entab (Listing 3). Again, we open the input file and call settab() to set up the tab line spacing. This filter runs within an outer **while** loop which reads in the characters from the input file and continues to loop as long as the character read is not EOF. Within this loop all the logic is contained within a single **if-else** test. In the **if** part, there is an infinite **while** loop which will execute if the input character is a tab character. This loop will output blanks until the next tab stop is reached, then terminate. The **else** part of the conditional is an **else-if**, where the input character is tested to see if it is a new line character. If it is, then a line feed is output and the column counter is reset to one. If it isn't, then the character is output and the column counter is incremented. After EOF is encountered, the files are closed.

A Last Word

After that rather tedious description of code, I hope I still have a few people reading along. I've presented the code for a few filters and, hopefully, have shown just how simple these little programs are to write. You can write a filter to do almost any type of task you wish and, since they tend to be utilities rather

(Listing 3 continued)

```
    "Invalid argument: [-n] must be a digit",
    "Invalid file name"
};

main(argc,argv)
int argc;
char *argv[];
{
  int col, i, tab__spc,
    tabs[MAX__TABLINE],  /* tab positions */
    *str;          /* scratch string pointer for parameter */
                   /* line */
  char filename[13],   /* buffer for input filename */
    c;

  FILE *fopen(), *fp;

  if (argc < 2)  /* check that something has been entered */
  {
   p__error(0);
   exit();
  }

  if (argc > 3)  /* check for a valid number of arguments */
  {
   p__error(1);
   exit();
  }

  if (*argv[1] = = '-')  /* check order of command line args */
  {
   p__error(1);
   exit();
  }

  /* parse the command line input */

    if (argc = = 3)          /* the second argument is */
    {                        /* the filename, third is */
      if (strlen(argv[1]) < = 12)  /* tab spacing */
        strcpy(filename,argv[1]);
      else
        p__error(3);
      if (*argv[2] = = '-')
      {
      str = argv[2] + 1;
        if (isdigit(*str))  /* is the argument is a digit */
        {
          tab__spc = *str + + - '0';
          if (*str ! = NULL)     /* if there's more */
            if (isdigit(*str))  /* is it a digit */
              tab__spc = tab__spc * 10 + *str + + - '0';
            else
            {
              p__error(2);  /* abort with error message */
              exit();
            }
        }
        else
        {
          p__error(2);    /* abort with error message */
          exit();
        }
      }
      else
      {
        p__error(2);
        exit();
      }
    }
    else  /* only the filename is given on the command line */
    {
      if (strlen(argv[1]) < = 12)
      {
        strcpy(filename,argv[1]);
        tab__spc = DEF__TAB;  /* use default tab spacing */
      }
        else
          p__error(3);
    }
  /****** the following is the code for detab ******/

    fp = fopen(filename,"r");
    settab(tab__spc,tabs);

    col = 1;
    while ((c = getc(fp)) ! = EOF)
    {
      if (c = = '\t')
        while (1)
        {
          putc(' ',stdout);
          col + +;
          if(tabpos(col,tabs) = = YES)
            break;
        }
      else if (c = = '\n')
        {
          putc('\n',stdout);
          col = 1;
        }
        else
        {
          putc(c,stdout);
          col + +;
        }
    }
    fclose(fp);
    fclose(stdout);
  }  /* end of main */
```

than full blown applications, they are easier and more straight forward to write. The important thing to remember about filters is that they can be used to make the files written by your application more regular and homogenous, thereby easing your programming task when it comes to file manipulation.

Thanks for stopping by and I'll see you next time. ■

AMPRO 186 Column
Installing MS-DOS Software
by Peter Ruber

A Trivial Foreword

My sudden appearance in the pages of The Computer Journal is the result of mysterious forces. I was recommended by two individuals I do not know to a magazine (I am embarrassed to say) I did not know existed. I have since learned that these culprits are Dave Feldman and Rick Lehrbaum of Ampro Computers—and, in view of their reckless suggestion, the probability exists that they might also be the sinister agents of Fu Manchu.

During the hey-day of computer madness (1981-84), I squandered small fortunes subscribing to nearly every magazine and newsletter I could lay my hands on. I wasn't picky—I read everything from slick, nauseating hype to the more esoteric publications discussing the finer points of Vas Ist DOS?

It was, in some ways, a fascinating period where the stereotypical public relations drum-beaters had a field day peddling computers like patent medicine; and when software houses specialized in the art of vaporware, prolonging the agonizing arrival of "alleged" sophisticated software packages that seemed to make it to the market just as a particular computer system was about to bite the dust.

My friend, Lee Hart, owner of Technical Microsystems in Ann Arbor, MI., and a designer of some interesting hardware for Heath/Zenith computers, has an amusing translation of what certain advertising words and phrases really mean in the computer promotion game. They are too biting and true to keep to myself, so I am passing a few selected descriptions on to you:

NEW—different color from previous model.
ALL NEW—no interchangeable parts with previous model.
IMPROVED—old bugs replaced with new ones.
EXCLUSIVE—imported product.
UNMATCHED—almost as good as the competition.
FOOLPROOF—no provisions for adjustment.
ADVANCED DESIGN—ad copy writer doesn't understand how it works.
FIELD TESTED—manufacturer lacks test equipment.
FACTORY DIRECT—manufacturer in fight with distributors.
RUGGED—too heavy to move.
LIGHTWEIGHT—lighter than rugged.
PORTABLE—has a handle.
HIGH PERFORMANCE—almost meets designer specs.
BREAKTHROUGH—we finally figured out how to sell it.
EFFICIENT—uses 1% less power than previous model.
IBM COMPATIBLE—paint matches IBM PC.
MAINTENANCE FREE—impossible to repair.
SATISFACTION GUARANTEED—ours, on receipt of your check.
RELIABLE—prototype worked at least 1 week between repairs.
FULL SUPPORT—broken units available as spare parts.
OBSOLETE—dependable, reliable, inexpensive and readily available.

I admit, quite shamelessly, that I got caught up in some of the PR hype and acquired a number of computer systems that I ultimately had no use for. My motto seemed to be "A penny earned was another penny to spend on computers." Faced with my wife's threats of exile, I palmed these computers off on my children who now use them to catalog their baseball cards and run the monthly phone bill into the stratosphere with their modems. Even the magazines I used to subscribe to seemed to lose their steam after a while, and vanished with the winds.

But I did have a constant in my computing life—what my wife calls my electronic mistress. My Heath H-89 computer. It was the Rolls Royce of 8-bit systems. The beast that wouldn't die. Some of the most clever enhancements ever designed were created for the old '89, and I took advantage of most of them. Especially, the Winchester systems about which I am writing a long series for REMark.

You may wonder what relation my Heath computer has to do with the Ampro Little Boards. By an odd coincidence, Henry Fale of Quikdata Computer Systems (Sheboygan, WI) whose Winchester subsystem for the '89 I had included in my series, was talking with William Dollar, Ampro's President, last fall, and he suggested that Bill dangle a Little Board under my nose. Henry knew me well enough to suspect that the SCSI hard disk interfaces on the Little Boards would intrigue me, and that I might want to write about them in my REMark column. It was definitely a good omen.

The Little Boards Arrive At The Rat's Nest

The Rat's Nest is my 8' × 22' basement dungeon wherein all the walls are plastered with computer reference books, manuals and magazines. What doesn't fit on the walls is stuffed into boxes around me. Three large work tables hold assorted computers, terminals, floppy and Winchester drives, printers, modems, monitors, miles of cables, and an over-loaded circuit breaker. Things have become worse since the Little Boards arrived. More drives, terminals, and other necessary accoutrements to satisfy the greedy pleasures of my latest computer acquisitions.

It is nigh impossible to walk from one end of the room to the other, because I thrive best in an atmosphere of chaos and general disorder. I am safe here. My wife refuses to enter my dungeon; the children know better. Only the dog is fearless.

I had barely unpacked the Little Boards and skimmed over the manuals when I knew I had something special to work with. While the concept of single-board computers is not new, the miniaturization that Ampro achieved by cramming every I/O port you were ever likely to need (serial and parallel ports, floppy and hard disk connectors) was more than the rest of the industry had to offer. And, as far as I was concerned, they were the greatest invention since the H-89. The Little Boards even made my IBM "clone" dull by comparison.

My favorite was the 186 PC-DOS board. It was a quarter of the size of the CPU board in my "clone", and its 8-MHz 80186 CPU allowed it to run nearly 300% faster. Even the Ampro CP/M Little Board/PLUS outdistanced my "clone", which I have read is faster than most.

I mounted the 186 LB inside my H-89, disengaged the CPU board and hooked one of the serial ports directly into the Terminal Logic Board. I attached two 40-track and two 80-track floppy drives, a 10-MB Winchester with a Shugart 1610-4 hard disk controller card, and I had a system to rival the best of them.

A short while later, I managed to trade some old Heath accessories for a used H-19 terminal and set up the 186 LB in a separate case of its own. The CP/M Little Board went into a case with a Seagate ST506 Winchester drive that I scavenged from a Hewlett-Packard hard disk subsystem. Because Ampro was foresighted enough to include a host of disk conversion utilities, I had no difficulty transferring the key programs in my Heath CP/M library to the Little Board disk format.

But my first attempts to get MS-DOS and PC-DOS software up and running on the 186 PC-DOS board was a challenge of the first magnitude. The primary reason was that the 186 LB is not a true IBM-compatible computer. All "clones" share a common design concept that revolved around the hybrid 8/16-bit 8088 CPU, the 8284 Clock Generator, the 8288 Bus Controller, the 8237 Direct Memory Access Controller, the 8253 Counter/Timer, and the 8254 Interrupt Controller.

In contrast, the 80186 CPU used in the 186 LB is a high performance microprocessor that combines all of the functions of the chips mentioned above into one 68-pin IC. There is also a corresponding drop in the number of logic support chips from about 40 to 18. Peripheral I/O control in a "clone" is performed by the 8250 Asynchronous Communications IC, the 8350 Synchronous/Asynchronous Communications Interface, and the 8255 Peripheral Interface.

The corresponding 186 LB Serial and Parallel Port functions are handled by a Signetics 2681 Dual Asynchronous Receiver/Transmitter (DUART). The "clone" can service two 40-track DS/DD drives through the NEC765 floppy controller. The 186 LB uses the Western Digital 1772 which (with the addition of the AMPRODSK formatting utility) allows you to use up to 4 DS/DD drives in any combination of 40- and 80-track drives. It is worth having at least one 80-track drive for storing your work files, and backing up your hard disk files. You can, of course, add 2 additional floppy drives to your "clone", if you're willing to give up an expansion slot by adding a second controller card.

The missing ingredient in the 186 LB is the IBM's 6845 CRT controller (a chip that dates back to the dark ages of computer design but is still popular—probably because it's cheap), nor the 16k of screen RAM it feeds upon. In fact, there is no CRT control provided by the 186 LB. You must use one of several standard ASCII terminals in order to complete your system.

At the present time, Ampro's TERM.SYS terminal driver program will support the use of ANSI protocol terminals, the Heath/Zenith '19 terminal (but you can use the newer Z-29, Z-39 or Z-49 terminals), the Televideo 910/912/920/925, the ADDS Viewpoint, the Hazeltine 1500 and the Wyse 50. The latter has programmable function keys which can be enabled through Ampro's FUNKEY.SYS utility.

I am partial to the Heath/Zenith terminals for many reasons, not the least of which is brand loyalty, but also because the documentation and technical support may well be the best in an industry noted for incomplete and fuzzy documentation. However, the most important factor is that the terminal control and escape sequences established by the H/Z-19 terminal have been preserved in the Z-29/-39/-49 models. These terminals can also be configured for ANSI mode through keyboard input and have their set-up configurations preserved in non-volatile RAM. The '19s ANSI configuration is accomplished by pushing pin 5 on Switch 401 to the ON position. Late-technology Zenith terminals can also emulate the Hazeltine and a variety of mainframe terminals.

The choice of terminals is therefore an important one because the greater the range of emulation, the more success you will have in using a larger variety of MS-DOS software on the 186 LB.

Any software that directly addresses the IBM graphic screen will simply hang up and go ihto an endless loop. The only way out of this predicament is to perform a hard reset of your system. In the near future, Ampro will be releasing some additional hardware and software that will provide almost total emulation of IBM screen protocols and function keys by intercepting the software codes and translating them to ASCII format.

Since each of the terminals supported by the 186 LB has different characteristics, graphics tables, clear screen and cursor addressing conventions, your first step in setting up your system is to define the type of terminal you are using.

This is accomplished by creating a CONFIG.SYS file through EDLIN (or any Text Editor) that reads:

```
DEVICE = \ SETCON.SYS B9600 D8 S1 PN H
DEVICE = \ TERM.SYS T4
DEVICE = \ ANSI.SYS
DEVICE = \ AMPROKEY.SYS
```

Since I didn't bother to forewarn you, you are probably wondering what I have done. The PC-DOS (and MS-DOS) CONFIG.SYS program works much in the same way as CONFIGUR.COM in CP/M. It establishes the power-up default settings for communicating with I/O devices on your system. Without this, you have a system patiently waiting for you to tell it what to do.

The 186 LB communicates with its terminal through Serial Port A. SETCON.SYS initializes this port when you provide it with the proper Baud Rate, number of Data and Stop Bits, Parity, and whether hardware handshaking is necessary. Handshaking on the H-19 terminal is handled by the software. Later Heath/Zenith terminals require that you set up hardware handshaking because the system looks for it.

TERM.SYS is the master program that translates software screen control to the type of terminal you are using. The "T", followed by a numerical designation defines the terminal you have. If your terminal is capable of communicating through an ANSI mode, it is advisable to add this to your CONFIG.SYS table of devices. If you do, you must remember to add the ANSI.SYS program to your working disk.

Lastly, the AMPROKEY.SYS driver will assign equivalents of the IBM function keys to your terminal as shown below.

Before I discuss some specific examples, I must point out that in order to gain the maximum benefit from the 186 LB, you should purchase a copy of the IBM PC-DOS v3.0. Ampro's ROM BIOS has been optimized to use PC-DOS 3.0 because of its enhanced features and its ability to boot directly from a hard disk drive. However, if you have v2.1, it is usable but not recommended for booting from a hard disk or using a hard disk drive larger than 10-Megabytes. PC-DOS 3.0 also has some advanced features such as allowing you to reassign your drive letters, and creating a RAMdisk.

The Little Board's 80186 CPU is totally compatible with all 8088 and 8086 Object Code and software development tools, among which are TURBO PASCAL, FORTRAN 77, MICROSOFT C, LATTICE C, the IBM MACRO ASSEMBLER, GW BASIC, the INTEL family of COMPILERS and TOOLS, plus PALASM, ABEL, MASM, and many others.

PC Function Key	Key Sequence Required		
F1	<CTRL-^>	1	
F2	<CTRL-^>	2	
F3	<CTRL-^>	3	
F4	<CTRL-^>	4	
F5	<CTRL-^>	5	
F6	<CTRL-^>	6	
F7	<CTRL-^>	7	
F8	<CTRL-^>	8	
F9	<CTRL-^>	9	
F1Ø	<CTRL-^>	Ø	
Shift-F1	<CTRL-^)	!	
Shift-F2	<CTRL-^>	@	
Shift-F3	<CTRL-^>	#	
Shift-F4	<CTRL-^>	$	
Shift-F5	<CTRL-^>	%	
Shift-F6	<CTRL-^>	^	
Shift-F7	<CTRL-^>	&	
Shift-F8	<CTRL-^>	*	
Shift-F9	<CTRL-^>	(	
Shift-F1Ø	<CTRL-^>	)	
Home	<CTRL-^>	W	(or w or <CTRL-w>)
Up Arrow	<CTRL-^>	E	(or e or <CTRL-e>)
PgUp	<CTRL-^>	R	(or r or <CTRL-r>)
Left Arrow	<CTRL-^>	S	(or s or <CTRL-s>)
Right Arrow	<CTRL-^>	D	(or d or <CTRL-d>)
End	<CTRL-^>	Z	(or z or <CTRL-z>)
Down Arrow	<CTRL-^>	X	(or x or <CTRL-x>)
PgDn	<CTRL-^>	C	(or c or <CTRL-c>)

This makes the 186 LB an ideal software development system. Not only is it an inexpensive alternative to a full-blown, overpriced IBM, but it will assemble and compile large programs in RAM at speeds in excess of what even an IBM PC/AT can handle. As an added bonus for IBM-PC program developers, Ampro offers a MONITOR EPROM replacement chip set for the ROM BIOS for only $79. The Monitor was created for program debugging and for testing ROM and EPROM based software. It provides a means for you to access the primitive functions of the 80186 CPU, including access to the 80186 internal registers, I/O, memory and disk devices. Facilities are provided to set breakpoints, load Intel HEX format files and execute programs in memory. The neat feature of the MONITOR EPROM is that you retain the full operation of all peripheral devices.

Productivity and application programs that are usable on the 186 LB fall into the category of what is commonly referred to as "generic" MS-DOS programs. This means that any program that addresses DOS functions or the ROM BIOS calls rather than the IBM screen or the hardware I/O, should be usable on the 186 LB.

As I mentioned earlier, I had some problems getting certain programs operational because I was simply overlooking the obvious. One has a tendency to become mentally lazy when using the IBM or its compatible machines. While it's nice to just load a new program and have it pop on the screen, the system does not encourage you with any significant learning experience.

However, taking the principles from the more sophisticated application programs like LOTUS and SYMPHONY, which you must configure to recognize the type of graphics card and printer you are going to use, you must perform a similar function with MS-DOS "generic" software for the 186.

A good example to discuss is the SPELLBINDER WORD PROCESSING AND OFFICE MANAGEMENT SYSTEM from Lexisoft, Inc (PO Box 1378, Davis, CA 95617). This was, and is, the forerunner of today's "integrated" software packages, and it is available for nearly every CP/M-80, CP/M-86 and MS-DOS computer and terminal on the market today.

SPELLBINDER has a CONFIGSB installation file that allows you to set up the program for the terminal characteristics of all (but the ANSI) terminals supported by Ampro's TERM.SYS utility. It includes a 50,000-word spelling checker program called ELECTRIC WEBSTER, and is capable of spreadsheet, database, mail-list and invoicing functions. A versatile Macro program is provided for user created templates. If you plan to set up SPELLBINDER on a hard disk, you must indicate during the installation process that you have 5 drives on your system. Since Ampro's BIOS supports 4 logical or physical floppy drives, your hard disk becomes the 5th drive.

If you neglect this and use your hard disk as the boot drive, you will have problems accessing SPELLBINDER's help files, because the system does not recognize the hard disk and automatically looks for drive A: to contain the help utilities, even if you only have a hard drive attached. Thus, by telling the program you have 4 logical or physical drives on your system plus a 5th drive, which is the hard disk, you can then use your hard disk as you would floppy drive A: without having to reassign drive letters.

By contrast, T/MAKER INTEGRATED SOFTWARE from T/Maker Software, doesn't provide an installation program for the H/Z-19 terminal through the T/MODIFY utility. But it does support ANSI terminals, and by setting the appropriate switch, I had this program up and running in a matter of minutes. I was also able to test the program in ANSI mode on my IBM compatible and on an IBM PC/XT at the computer department of a local college. T/MAKER is sold by Ampro for $159 and is included in their assembled "Bookshelf" series of Little Boards.

SPELLBINDER lists for $495, but can usually be purchased from dealers for $295. T/MAKER may well be one of the best software bargains on the market. It doesn't have SPELLBINDER's creative printer control features, but it has a far more readable manual and the ability to accept many commands in the form of English words instead of cryptic symbols.

Other programs in my IBM software library that contain configuration files are DBASE II, WORDSTAR and SUPERCALC 2. Several utility programs such as the printer-spooler DSPOOL, SIDEWAYS (a sideways printing program), RAMDISK, FORMIT (a text formatting program), and the NORTON UTILITIES work without modification. I recently acquired DBASE III and discovered that it contains an installation utility for ASCII terminals. I haven't had time to work with it yet, but I'll report on it in a future article.

I am certain that some of you may have different programs successfully installed on the 186 LB. As I would like to begin compiling a list of usable programs, I would appreciate hearing from 186 Little Board users. As the list grows, I will include a listing of these programs in future columns so that all of us can have a wider choice of programs to work with. Readers of this column can write to me at P. O. Box 502, Oakdale, N. Y. 11769.

In the meantime, do not charge out and recklessly purchase any software before you have properly investigated it. If you already own an IBM or compatible system, test out the software you have. Chances are, that if it is a "generic" program or one of the major programs that have made a successful transition from the 8-bit systems to 16-bit format (such as several of the programs I listed above), you should not have problems using it.

If you are in doubt about an unknown program, test the program at the computer store to see if it has a configuration or installation file on the disk. If not, write or call the publisher and inquire if a "generic" version is available that can be configured for different terminals.

Ampro's latest 186 LB software updates includes a nice public domain communications program written by Jerry Haigwood (operator of the AMPRO ONE BBS), called LBCOMM. Ampro is also distributing a powerful communications program called MICROLINK II for both the CP/M and 186 Little Boards for $99. The publisher is Digital Marketing Corp., 2363 Boulevard Circle, Walnut Creek, CA 94595.

There is yet another communication program available from Ampro called SUPERDUO, which was developed by Wordcraft, 3827 Penniman Ave., Oakland, CA 94619. This program will allow the 186 LB to use an IBM or compatible computer as a terminal. Among the features are a one-key toggle to enable the user to gain control of either system. Programs can be run and/or assembled on both computers simultaneously, and the IBM can also use the Ampro's 512k of memory and SCSI hard disk interface. I plan a full report on SUPERDUO in the next issue of The Computer Journal. ■

The Z Column

by Art Carlson

User Areas

In the last issue, we talked about the advantages of ZCPR3 over CP/M in accessing Directories and User areas. Since then, I have reorganized a number of disks by putting the utility programs in user area 15, so that they can be accessed from any user area but do not not clutter up the directories in the working areas. One of the big advantages of this for my work is that I can erase all the files in the working user area when I finish a project, and still have the utilities available in another user area. Of course, I save the few useful files before I wipe an area, but I like working in a clean area with a directory having only the active files.

I set up my BDS C compiler in drive A, and keep various programming projects separated in the different user areas in drive B. (Yes, I even like to keep my work files on a different disk than the program files.) Most .COM files work properly when called from different users areas, but some programs (such as BDS C and WordStar) call other files, and cannot find them if they are not in the current user area. BDS C provides for this with simple configuration patches for CC.COM and CLINK.COM on page 13 and 14 of the version 1.5a manual. For CC.COM, you load the file using DDT, then patch locations 155H and 156H with the default library disk drive and user area, then save the patched version. For CLINK.COM the patch locations are 103H and 104H. You shouldn't have any problems if you have the BDS C and DDT manuals, but let me know if we should take a column to go thru the steps.

When using WordStar under ZCPR3, the system can find the COM file if it is in the search path, but Wordstar looks for its overlay files in the current user area of drive A. I know that there is a WordStar patch which tells it which drive to access for the overlay files, but I don't have a patch for the user area. There is a public domain ZCPR3 utility, PMOVE, on the AMPRO BBS which solves this problem by making selected files "PUBLIC". Then you "move" the files to the current user area. In order to use PMOVE, you'll also need the ZCPR3 utility "PROTECT.COM" which is available on the AMPRO extended ZCPR3 support disks or from the Z-NODE bulletin boards.

The procedure (which is spelled out in the PMOVE.DOC file) is to use PROTECT.COM to set the PUBLIC attribute in the directory FCB:

PROTECT <WS*.OVR> 1

then you can make the files useable from another user area with the command:

PMOVE A:WS*.OVR

When you change to another user area you'll have to PMOVE them to that area. PMOVE doesn't actually move or copy the files, but just changes the user area specification for the files in the directory.

This is not quite as convenient as what we are used to with the PATH search, but it only needs to be used on programs which call secondary programs or overlays.

A Better SAVE Utility

While patching the BDS C compiler, I used another ZCPR3 utility which is much better than the similar CP/M utility. When you load a file with DDT it displays the file size in hexadecimal, but when you are ready to SAVE the patched file under CP/M you have to convert the file size to decimal. Both DDT and SAVE are supplied on the same disk by DRI, and you'd think that they would have written them to use the same number base. But they didn't, and it's because of things like this that we have abandoned CP/M for ZCPR3.

DDT gives the CC.COM ending address as 3A00. With the CP/M SAVE utility you have to convert the first two number to decimal ($3 \times 16 = 48$, plus $A = 10$, $48 + 10 = 58$ decimal) then subtract 1 if the last two numbers are 00. You then enter SAVE 57 CC.COM to save 57 decimal 256 byte pages of memory to disk. The reason that you have to subtract 1 is that the first page of memory is used by the system, but is included in the size reported by DDT.

The ZCPR3 SAVE utility allows you to specify the decimal number of pages, or the hexadecimal number of pages, or the number of 128 byte sectors in either decimal or hexadecimal. For this example we just subtract 1 and enter SAVE 39H CC.COM.

Better Error Trapping

One of my pet peeves is when WordStar dumps out to the system and does a warm boot just because I tried to log onto drive which doesn't have a disk. I complained about this to Cla[illegible] Calkins (C.C. Software) while discussing his CP/M Source Co[illegible] Generator, and he told me that this is not a fault with WordSt[illegible] because the error is caught and handled by the system, and th[illegible] I would have to modify the BIOS to correct it. Incidentally, h[illegible] source code gererator is really amazing—even if it doesn't wor[illegible] for ZCPR3. I used it on my Morrow S-100 CP/M 2.2, and [illegible] generated 64 pages of commented data on the CCP and BDOS. For $45 it's a real bargain if you're interested in tearing into the operating system.

While I was double checking the use of PROTECT.COM for the first part of this article, I put the B: drive disk in upside down, and when I tried to access it the terminal beeped at me. No crash, just a gentle reminder, and I turned the disk over and continued. This made me think about the WordStar problem I used to have under CP/M, and I wondered how ZCPR3 would handle this. I booted WordStar, left the B: drive door open, and tried to log on drive B:. The terminal just beeped, and when I closed the drive door it logged the disk and everything was OK. This is another CP/M problem which ZCPR3 (at least the AMPRO implementation) has solved. One of these days I'll pack a disk and see what happens when WordStar tries to write to a full disk under ZCPR3.

Multiple-Command Lines

The CP/M system includes the SUBMIT transient command to enable issuing multiple commands, but it is so awkward to use that I've never used it. With SUBMIT you have to use an editor or word processor to write a command file and save it to disk. Then you call the SUBMIT command file. That may be of some use in a production environment where you are performing the same operation over and over again, but I'm always doing something different and it takes longer to prepare the submit file than it does to issue the individual commands (one command per line).

Most ZCPR3 implementations include the standard 1K overhead which provides for a number of buffers, one of which is the 200 byte Command Line Buffer for multiple commands on a single line. For example, I used Wordstar to create the assembler source code for FILTER.COM and wanted to test it. The source code was in B2:, and ASM.COM and LOAD.COM were in A15:, and I entered the following command line to assemble, load, and run the program:

```
ASM FILTER;LOAD FILTER;FILTER
```

The mulitple-command feature is a great time saver, but ALIASES are even better for commands you use frequently and can include the IF, ELSE, FI, XIF, EMPTY, ERROR, EXIST, INPUT, and NULL flow commands. As a simple example, an ALIAS for assembling could be set up as:

```
ASM $1.BZZ;LOAD $1;$1
```

and called ASSEMBLE.COM. Now in order to assemble, load, and runa file called FILTER.ASM you would just type: ASSEMBLE FILTER. There is a lot more than this to ALIASES

Echelon Z-Node List #21 — 10 March 1986

NODE	SYSOP	CITY	STATE	ZIP	RAS Phone
8	Thomas Hill,	Anchorage,	AK	99504	
24	Barry L. Bowerman,	Weaver,	AL	36277	
22	Thomas R. Bowerman,	Anniston,	AL	36201	205/238-0012
52	Wells Brimhall,	Phoenix,	AZ	85028	602/996-8739
20	Richard Driscoll,	Phoenix,	AZ	85008	602/939-6734
44	Robert Gear,	Phoenix,	AZ	85014,	602/279-2762
35	Norman L. Beeler,	Sunnyvale,	CA	94086,	408/245-1420
34	Rod L. Blackman,	Visalia,	CA	93291,	209/739-8303
54	Clinton Cook,	Merced,	CA	95348,	209/383-6417
21	Robert Finch,	Glendale,	CA	91205	
6	Andrew Hart,	Palo Alto,	CA	94306,	415/493-4506
2	Al Hawley,	Los Angeles,	CA	90056,	213/670-9465 *
57	Steve Kitahata,	Gardena,	CA	90247,	213/532-3336
1	David McCord,	Fremont,	CA	94536,	415/489-9005
36	Richard Mead,	Pasadena,	CA	91105,	818/799-1632
18	John Rovner,	Union City,	CA	94587	
25	Douglas Thom,	San Jose,	CA	95129,	408/253-1309
19	Fred Townsend,	San Jose,	CA	95132,	408/262-5150
9	Roger Warren,	San Diego,	CA	92109,	619/270-3148
10	Rea Williams,	El Toro,	CA	92630,	714/855-0672 *
28	Stanley K. London,	Aurora,	CO	80013,	303/680-9825
53	Peter Glaskowsky,	Miami,	FL	33156,	305/235-1645
27	Charlie Hoffman,	Tampa,	FL	33629,	813/831-7276 *
32	Allan E. Levy,	Satellite Beach,	FL	32927	
17	Robert B. Tate,	Altamonte Springs,	FL	32701,	305/831-6049 *
29	Edward C. Unrein,	Altamonte Springs,	FL	32701,	305/774-2591 *
51	Edward C. Unrein,	Orlando,	FL	32810,	305/295-0844 *

46	Jim S. Altman	Atlanta,	GA	30316,	404/627-7127	*
15	Richard Jacobson,	Chicago,	IL	60606,	312/649-1730	*
15	Richard Jacobson,	Chicago,	IL	60606,	312/664-1730	*
5	Ron Stone,	Lisle,	IL	60532,	312/420-1722	*
37	Marvin Eyre,	Robards,	KY	42452,	502/521-7011	
3	Jay P. Sage,	Newton Centre,	MA	02159,	617/965-7259	*
43	John D'Ausilio,	Bladensburg,	MD	20710,	301/779-7986	*
41	Larry Mansfield	Baltimore,	MD	21214,	301/254-6277	
30	Ben Ragan,	St. Louis,	MO	63134,	314/423-7038	
48	Brian B. Riley,	Indian Mills,	NJ	08088,	609/268-9597	
11	Michael M. Ward,	Voorhees,	NJ	08043,	609/428-8864	
23	Charles Boghosian,	Durham,	NC	27712,	919/383-6595	
42	Jay Denebeim,	Durham,	NC	27705,	919/471-6436	*
14	Rich Rodeheaver,	Reynoldsburg,	OH	43068,	614/864-2673	*
47	Tom R. Keith,	Ponca City,	OK	74601,	405/762-1651	
49	Kevin Dobb,	Medford,	OR	97504		*
60	Bob Peddicord	Selma,	OR	97538,	503/597-2066	
38	Robert L. Paddock,	Franklin,	PA	16323,	814/437-5647	
4	Don Buzzingham,	College Station,	TX	77843,	409/845-8931	
56	Terry Carroll,	Bedford,	TX	76021,	817/283-9167	
33	Mark R. Evans,	San Angelo,	TX	76904		
31	Richard A. Petersen,	El Paso,	TX	79904,	915/821-3638	*
45	Richard K. Reid,	Houston,	TX	77088,	713/937-8886	
39	Jon Schneider,	El Paso,	TX	79936,	915/592-4976	*
12	Norm Gregory,	Seattle,	WA	98122,	206/325-1325	*
7	Tim Linehan,	Olympia,	WA	98502,	206/357-6757	
16	Jud Newell,	Islington, Ontario		M9A 1A7 CANADA,	416/231-9202	*
40	Terry Smythe,	Winnipeg, Manitoba		R3N 0T2 CANADA,	204/452-5529	*
26	Robert Kuhmann,	Belle Etoile, par St. Martin de la Brasque		84760 FRANCE,	011-33-90-77-60-15 (from USA)	*
50	Mark Little,	Alice Springs, N.T. Australia		5750	011-61 (089) 528 852 (from USA)	*

Notes: 1) Asterisk (*) indicates node is a downloader of Echelon proprietary software. Ask node Sysop for procedure.
2) Lack of RAS (Remote Access System) telephone number indicates node presently may not be up but should be within two months, or we have not yet received number.

and we'll have to devote an entire column (or even several columns to this subject), but you should start using them to make your tasks easier.

More Z Tools

ZCPR3 is a worthwhile replacement for CP/M by itself, but what really excites me is the number of utilities and tools which expand it into a programming and applications system. These tools are all coordinated to work together with the system—not just a bunch of individual programs—and Echelon is adding a lot of new tools. If you're interested in ZCPR3 you should be reading Echelon's Z-NEWS, but for those who don't, I'll mention some of new developments.

One of their most recent releases is the Z-SYSTEM USER'S GUIDE priced at $14.95 plus $4.00 shiping and handling. This is a tutorial to help you get understand and get started with Z. You'll also need ZCPR3: The Manual later for more detailed technical information.

Frank Gaude at Echelon and I both prefer assembly language programming for most applications, but as mentioned above I have been using BDS C for typesetting filter programs because the typesetter is slow (6 char per second!) and they involve a lot of buffered I/O and string handling routines which I didn't wai to code in assembly. I want to stress that this is for a very slo process where the speed of assembly would be of no advantag and they are specific for my typesetter so they will not t distributed to others. Now Echelon has added "ZCPR3: TH LIBRARIES" which contains libraries of assembly languag routines which are called in when needed. This is similar to the use of libraries by C and will speed up assembly language programming because you only have to write the custom parts of the code. You'll hear more about this as I work with it.

In conjuction with the LIBRARIES, I'll be using Echelon's ZAS Relocating Macro Assembler and ZLINK Linker. They can also be used with the HD64180.

For a demonstration of graphics (including windows and pull down menus) on a standard ASCII terminal, download the demo from the Z-Nodes. And speaking of Z-Nodes, I've included a list of their BBS's.

This Is YOUR Column

It's up to you to let us know what features you'd like to see covered in future columns. Do you have any questions about Z? Do you have any tips about Z? Drop us a note (or check to see if our BBS is on line). ■

NEW-DOS

Part 3: The CCP Internal Commands

by C. Thomas Hilton

When last we met we had covered the primary CCP execution loop. What remained in our discussion was a coverage of the internal CCP command routines themselves. As you may recall, when we entered the internal command structure the return address was left on the stack, and we were essentially free to do whatever we wished, within the limits of the space available. We will conttnnue under the same concept, with a few slight changes.

We began this series with few expectations, and certainly didn't expect the response we have gotten from you, our readers. Thanks a lot people, but you have really made life difficult for me. Oh well, I had in mind to do some thing outrageous anyway.

So far, we have dealt with many of the complaints users have had about their operating systems. We knew we needed a "new DOS" to take care of many of these problems, yet many other problems seemed to remain. Well, I got tired of it all and put together a complete operating system for the AMPRO Z80 machines. This operating system was released into the public domain on April 1st 1986. That's right, a complete public domain operating system! Gone are the days when the end user screwed up the installation of software, and making sure the operating system was not shipped with the client's project disk.

The new DOS is called "Hermit DOS," and I do not expect anyone to like it, because it is a bit different. But, you can't complain about the price, and you may not only distribute it on your disks, but are encouraged to do so.

The good folks at AMPRO have done their part in allowing me to distribute an early version of their BIOS, modified for use with Hermit DOS, for noncommercial purposes. This allows the entire system to be free on board to users. The catch is that no portion of Hermit DOS may be used for commercial purposes without not only my express written permission, but also the express written permission of AMPRO Computers, if the version of Hermit DOS you receive "signs on" with an AMPRO copyright notice.

The constraints of serialized publication also have caused a problem in the preparation of this series. If the entire work were published in a bimonthly format it would be about five years before the series was completed. Most of us want the data yesterday.

It is Easter Sunday as I write this, and at this time the Hermit DOS collection is three disks long. The first disk is the user disk with all the tools required to work with Hermit DOS, and comes ready to run, just configure your terminal and boot it. There is a back-up DOS image which may be SYSGENed if something goes wrong, a basic set of utilities, and an 86 page WordStar document mode user manual. This is the disk you need to begin using the DOS, and is Version 2.0.

Disk TWO has nothing but text files (386 KBytes of them!) which may form a technical manual, and may become a book, if I ever learn how to write.

Disk THREE has the balance of the manual text (another 140 KBytes) plus additional utilities and the CCP source code. The CCP's internal command structure is nearly identical to that which we have been working with. To complete the series I have prepared this disk for TCJ using the CCP segments as they appear in the Hermit DOS manual, and which are based upon the actual source code file of the CCP.

Hermit DOS needs a little explaining, because the terms are a bit different. Under Hermit DOS we have an imaginary file cabinet, where each disk drive is a drawer. Within each drawer are 16 "files," or "file partitions." Within each file may be any number of documents, as long as there are no more than 128 documents per drawer. Each file has an "index," and three files have reserved functions.

When you boot the system you will be sent to file 1, which is the base working file. Command documents, documents with the document type of "COM" may be placed out of the way in file 0, the system file of either drive. the system will search the command file for any program it cannot find in the current file. File 15 is the "archive" file, and the Hermit DOS MAKE utility actually moves the document into the archive file if it is given that status.

The command prompts for these three types of files are shown below.

```
Drive A File 1>>
Drive B Archive file>>
Drive A System file>>
Drive B File 12>>
```

Additionally the AMPRO "E" drive is set for immediate reading of a Kaypro 4/10 disk. Just put the Kaypro disk in drive B and select drive E and you are on-line.

The keyboard operates differently as well. ^Q is the ^S function, ^W controls the echo to the prrnnter, ^A aborts and restarts the input command line, ^R reads back the command line, and the ESCApe key serves the ^C function. The DELete, or RUBOUT key and the backspace key have the same function, a destructive backspace.

The next three installments of the CCP section are on this disk, which I will send up to TCJ this evening. When these segments are completed, unless you demand we start immediately, this space will be taken by a user question & answer column devoted to the DOS, as well has "how to do it" short sections. Hermit DOS 2.0 is YOUR DOS, as readers of TCJ and AMPRO users. Where we go from here is up to you. We refuse to make any plans from here on, you let me know what you want, and we'll go for it.

Well, anyway, your questions will be answered, and we are planning to support YOUR DOS, as much as you want it supported. The only indicator we have is your hate mail, so send it in!

Enough of that, now where were we???.

The Standard CCP Personality Module

In this section, (an excerpt from the Hermit DOS Technical Manual) we will take a look at the standard command procedures which give the system its personality, or attributes. When we left off we were discussing the command search procedure. As you may recall, when we jumped to an internal command procedure we left a return address on the stack. When we return to the CCP's main processing loop we will reset the system. Therefore, we may consider ourselves quite independent when we arrive here. Because this is the area where you will be doing the most of your work, adding, modifying, and deleting command procedures, this is where I chose to place the command table. The less we have to interfere with the rest of the

CCP's operational code the better. Again, when we get here we are free to do all that we may choose, as long as there is space enough to do it in. The procedures we have already discussed may be called, as may be their support subroutines, to aid you in the design of your specific personality procedures.

Well, to work! When we left off we didn't really get a good look at the command table. This is where we will begin our discussion. Please use Listing 1 as a reference as we speak.

You may remember that we used a counter for the number of characters in the command string. The above is where the character count came from. In general you may use command strings of any length you wish, padding short command words with spaces, as long as you do not exceed the length of a standard command document name. The command document may be up to eight characters in length. The reason for this restriction on the length of the command procedure's name is due to the fact that procedure names are treated the same as document names, and are formatted in the DCB, which may handle only a maximum of eight characters per name. Remember that our interpretation code assumes that the command line contains first the name of a command document, and that the primary function of the CCP is to fetch and execute a command document, not provide us with accessory services.

In the interest of brevity we will not show the complete command table here. The table entries are quite redundant, each containing the command procedure's name, followed by a two byte address where the procedure itself is located. A partial command table is shown in Listing 1.

We do not have a command table terminator. We instead, as you may recall, keep track of the number of commands in the command table. The equation at the end of the table calculates an equate, and simplifies this process for us. All that is required, when entering or deleting a procedure is to assure that all reference of the procedure is either deleted from, or included in, the command table. The equation must be located after the last table entry, if it is to return accurate information to the command searching routine. I guess then you could say that this equation terminates the command table, though this is not overly accurate.

The first personality procedure returns an index of the current file partition, writing it upon the terminal. Documents, you will remember, may have various attributes. One of these attributes is the "private" attribute. The archive attribute is a functional attribute, and when this attribute is applied by the MAKE utility the document is "moved" into the archive file. For reasons having to do with security, this index routine will not reveal the documents in other files, only the file that is currently open. The various procedure syntax possibilities are shown in the syntax table, and are discussed in greater detail in the Hermit DOS User Manual, on Disk One, and again in the first part of this manual, (Disk Two if you have received this manual in disk format).

This index display is not "sorted," due to the size limitations of the personality module. When the user starts thinking about making room in the personality module for his own commands the index procedure will often be the first procedure to be removed to make space. The public domain utility SORT will read the disk's index from the disk, sort it, and write it back upon the disk for you. In addition a public domain version of the AMPRO DIR.COM utility is included upon most Hermit DOS disks.

The first order of business, (refer to Listing 2) is to load a mask with the most significant bit set for testing the attribute bytes of the index entries. The attributes allowed for use in this basic version of Hermit DOS use the 8th bit of the document type bytes for their functioning.

LISTING ONE

```
; ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][
;
;               C C P   P E R S O N A L I T Y   M O D U L E
;                        ( Standard Version 2.0 )
;
; ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][
;
NCHARS  EQU     5               ;number of chars/command

;       Internal procedure table. Each table entry is composed a name string
;and a two byte address of the procedure itself.
;
CMDTBL: DATA    'INDEX'
        DEFW    INDEX
        DATA    'ERASE'
        DEFW    ERASE
NCMNDS  EQU     ($-CMDTBL)/(NCHARS+2)   ;and here we calculate the number of
                                        ;commands to make it easy to add or
;delete commands without having to set a bunch of variables.
```

LISTING TWO

```
;
;       PROCEDURE IPP.01
;       ----------------
;       Purpose: Display an index of documents in current file partition
;       Verb Syntax:
;               index <ambiguous document name>    display public documents
;               index <ambiguous document name> P  display private documents
;               index <ambiguous document name> A  display all documents
;
;       Local Equates:
;
PRIVAT    EQU     'P'           ;for index command: display private
                                ;documents only
DISPALL   EQU     'A'           ;for index command: display all documents
;
INDEX:  LD      A,80H           ;load attribute mask for later use
        PUSH    AF              ;and store it on the stack
        CALL    PARSE           ;parse a possible drive change index sequence
        CALL    SSELDSK         ;select the new drive, if indicated
        LD      HL,DCBFN        ;load a pointer to DCB because we want to make
                                ;the DCB wild
        LD      A,(HL)          ;get the first character of document name
        CP      ' '             ;is it a space? if so then no document name
        CALL    Z,FILLQ         ;was specified so wild card the field
        CALL    ADVAN           ;then advance to the next valid character
        LD      B,0             ;clear the B register for upcoming functions
        JR      Z,NDX2          ;if advance call returned a set "Z" flag, no
                                ;more valid characters indicated, then jump out
       CP      DISPALL          ;else we may have an attribute command. does
                                ;the operator want all documents shown?
        JR      Z,FNDALL        ;yes, so jump out
        CP      PRIVAT          ;well then, does the operator want private
                                ;documents shown?
        JR      NZ,NDX2         ;no, that's not it either so jump out
```

LISTING THREE

```
        LD      B,80H           ;yes, we are to show private documents so load
                                ;a mask value into B and fall into findall
FNDALL: INC     DE              ;advance pointer
        LD      (CMDPTR),DE     ;and store where we are in the command line
        CP      PRIVAT          ;now lets check the order for private documents
        JR      Z,NDX2          ;and jump out if it is a pdoc order
        POP     AF              ;else get our bit flag back
        XOR     A               ;and set it to exclude private documents
        PUSH    AF              ;and put it on the stack to keep the stack in
                                ;order because the next entry point will
NDX2:   POP     AF              ;get it back and if we didn't take care of the
                                ;stack it wouldn't take care of us
;       Index contents written here. On entry the most significant bit,
;(msb) of the accumulator is set (mask with 80H), if private documents are
;not to be shown.
;
WRTNDX: LD      D,A             ;put the mask, arriving in A into D for safe
                                ;keeping
        LD      E,0             ;define the terminal display column as zero
        PUSH    DE              ;save these two values, mask and position, on
                                ;the stack
        LD      A,B             ;and bring in the private document flag into A
        LD      (PRVTST),A      ;and store it in the test variable

        CALL    SEARF           ;ask FDOS to search for document name in DCB as
                                ;a first occurrence
        CALL    Z,NODOC         ;if it returns with the "Z" flag set it could
                                ;not be found so advise operator of this fact
;       If FDOS finds the document name specified in the DCB then it will
;read in the entire index sector onto a buffer. Because it knows we only
;wanted the name, and FDOS only thinks in terms of 128 byte sectors, it
;returns a position code in the accumulator as to where in the sector buffer
;the requested index name, and associated index control data may be found.
;The standard document control block is 32 bytes long. the position code
;therefore contains an offset value that, times 32 will point to the
concerned document data.
;
NDX3:   JR      Z,NDX11         ;if the "Z" flag is set then FDOS couldn't find
                                ;the original request, or subsequent request,
;so we exit the current write sequence and prepare to return to the CCP
;level
;
```

LISTING FOUR

```
        DEC     A               ;else we adjust the position offset
        RRCA                    ;and convert it to an absolute address
        RRCA                    ;by simulating a multiply of the offset by
        RRCA                    ;32 because computers are terrible at math so
        AND     60H             ;we have to fake it
        LD      C,A             ;and stuff the now absolute address into C
        LD      A,10            ;and then add 10 to to it to point to the
                                ;private document attribute bit
        CALL    NDXPTR          ;and set the index write routine pointer
        POP     DE              ;then get a copy of the bit mask back
        PUSH    DE              ;and restore it as it was removed from stack
        AND     D.              ;then check mask for private document bit
PRVTST  EQU     $+1             ;pointer to private test variable
        CP      0               ;see if we are to show private documents
        JR      NZ,NDX10        ;and jump out on NON ZERO
        POP     DE              ;else get terminal screen position
```

(Listing 4 continued)

```
        LD      A,E             ;and prepare to
        INC     E               ;add one to it
        PUSH    DE              ;put it back
        AND     03H             ;and check to see if four entries have been
                                ;written, if so we want to start a new line
        PUSH    AF              ;but first save the value and flags
        JR      NZ,NDX4         ;and if new line is not required then jump out
        CALL    CRLF            ;else start a new line and
        JR      NDX5            ;jump out
```

LISTING FIVE

```
NDX4:   CALL    SYSPT           ;else write a space between the shown documents
        DEFB    ' : ',0
NDX5:   LD      B,01H           ;and set a counter for the number of characters
                                ;in the document name
NDX6:   LD      A,B             ;and place the character offset into A
        CALL    NDXPTR          ;and set the pointer, (hl now points to the
                                ;first character of the document name
        AND     7FH             ;then mask off the msb
        CP      ' '             ;and see if there is a document name to be
                                ;written at all
        JR      NZ,NDX8         ;if there is then write it to the terminal
        POP     AF              ;else get a copy of values and flags
        PUSH    AF              ;and put them back
        CP      03H             ;check again for display position on terminal
        JR      NZ,NDX7         ;and if not complete jump out
        LD      A,09H           ;else prepare to write the document type
        CALL    NDXPTR          ;and HL now points to first byte of doc type
        AND     7FH             ;so mask out msb
        CP      ' '             ;and check to see if there is a document type
        JR      Z,NDX9
NDX7:   LD      A,' '           ;else pad the type space with a blank
NDX8:   CALL    WRTERM          ;write it
        INC     B               ;advance character count
        LD      A,B             ;get it in A to check if done
        CP      12              ;are we finished yet?
        JR      NC,NDX9         ;no, not yet
        CP      09H             ;are we finished with name?
        JR      NZ,NDX6         ;no, so loop
        LD      A,' '           ;else write a separating space twixt type
        CALL    WRTERM          ;on the terminal
        JR      NDX6            ;and loop
NDX9:   POP     AF              ;else take care of stack
NDX10:  CALL    TRMSTAT         ;and see if operator is in a panic
        JR      NZ,NDX11        ;and if so abort
        CALL    SEARN           ;else ask FDOS to search for next occurrence
        JR      NDX3            ;and loop back for more display
NDX11:  POP     DE              ;we come here to exit so we clean up the stack
        RET                     ;and exit.
```

LISTING SIX

```
;
;       Index searching routines called by INDEX
;
SEARF:  LD      DE,DCBDN        ;tell FDOS where the DCB is
SEAR1:  LD      C,17            ;load the search for first occurrence code
CODRET: CALL    FDOS            ;make the disk related I/O and set the "Z"
        INC     A               ;flag to indicate the function status
        RET                     ;as you leave
;
SEARN:  LD      C,18            ;load the search for next occurrence code,
FDOS
        JR      CODRET          ;remembers DCB address for this function call
                                ;and make the code return FDOS jump
;       We get called here if FDOS cannot find the document name specified
;in the DCB so we have to tell the operator to get it together before we
;exit the INDEX procedure.
;
NODOC:  CALL    SYSPTC          ;write the following message on the terminal on
                                ;a new line
        DEFB    'Document(s) Not Found',0
        RET                     ;and return to caller
;       Point to INDEX entry in the page zero buffer whose absolute position
;is indicated by the contents of the accumulator and the C register. On exit
;the accumulator contains the specified byte from the index sector.
;
NDXPTR: LD      HL,PZBUFF       ;use HL to point to base of page zero buffer
        ADD     A,C             ;add the two together to point to first byte
                                ;if index entry in sector
        CALL    ADDAH           ;formulate the exact position of the byte as
                                ;referenced by the H register
        LD      A,(HL)          ;and load the specified byte into accumulator
        RET                     ;for trip home
;
;       Used by NDXPTR. Adds accumulator to HL pair, (hl=hl+a).
;
ADDAH:  ADD     A,L             ;add accumulator to lower half of HL
        LD      L,A             ;and put result, in A, into L register
        RET     NC              ;return if no carry
        INC     H               ;else adjust carry and then
        RET                     ;return to caller
;       Operator activity check, called first by INDEX. On exit the "Z" flag
;is set if operator has not pressed any key. This routine uses the FDOS
;terminal status function call for its operation.
;
TRMSTAT:
        PUSH    DE              ;save the DE pair on stack
        LD      C,11            ;load FDOS function code for terminal status
        CALL    FDOSB           ;and make the function call saving BC pair
        CALL    NZ,INPTRM       ;if there is a character then get it
BRKBK:  POP     DE              ;else give me back the DE register values and
        RET                     ;then return
;
INPTRM: LD      C,1             ;input character from the terminal with echo
        CALL    FDOSB           ;but save BC pair
        JP      UCASE           ;and capitalize character before returning
;
;       Call FDOS, save BC pair and adjust "Z" flag on exit
;
FDOSB:  PUSH    BC
        CALL    FDOS
        POP     BC
        OR      A
        RET
```

The next order of business is to see if we are to read an index from a drive other than the currently active one. If we do have that kind of indicator the parsing routine will detect it, and if not the "select specified disk" routine will abort if no new drive is required. Please refer to previous discussions if you do not understand the operations of these two primary loop subroutines.

Next we want to see if we have a specified document name in as a procedure parameter. If not then the document name field will be padded with space characters. And if it is padded with space characters then we will want to make the entire name field "wild" by filling it with question marks, the only wild card character the FDOS segment understands.

If the document name field is not padded with spaces then we will want to advance to the next valid character in the DCB/command line. It is quite possible that the next character may be one of our single letter instruction tokens. But, if we have reached the end of the command before finding anything of interest then we will jump out with the B register cleared, indicating no special attribute masking.

If we have not jumped out, then the advance routine has found us a valid character. Is it the single letter token to indicate the display of all documents in the file, without concern as to their status? If it is we will jump out, else we will continue to try and discover what the character may be.

Is the character the token for displaying only documents with the private document status? If not, then we jump out, else we will begin to prepare for the display of private documents. Refer now to Listing 3.

To display the private status of documents we load the B register with a mask for upcoming processing and fall through into the generic, "display all documents" code section.

We begin here by incrementing the parameter pointer and storing it in the command line pointer variable for safe keeping. We want to check to see if for some reason we do have a valid command to display private documents. We will never display private documents, unless specifically told to do so. As FIND ALL documents is a public entry point, it never hurts to check again and be sure.

If we do find a private document display order then we will jump out, else we will recall the bit mask we stored on the stack when we first entered the procedure. Next we will clear it and put it back on the stack.

Notice that we use the exclusive or to clear the accumulator. When we exclusive or a value with itself nothing will remain. The law of the exclusive OR is that, "only one may be one, or none will be one."

Though we just put the cleared accumulator value on the stack, we may have entered INDEX2 from many other places, of which any of them may have placed an altered mask on the stack. By bringing this mask off the stack we not only have the current mask value, but have cleared the stack of this burden.

As we begin the writing of the index we bring the attribute mask, which arrives in the accumulator, into the D register, of the DE register pair, and clear the lower register, the E register, with a null character.

Recall that when we discovered that we were to display documents with the private attribute we loaded a mask into the B register. We now recall that mask and place it into a register we will use for testing private document names, as they are found in the index.

With everything prepared, the DCB formatted for the display of documents, we now call a slave routine to search for the first document in the disk's index which matches the operator's specification. In the event that FDOS does not detect a document which matches our specification we will jump out to an specific error routine which will advise the operator of this fact.

As noted in the listing's commentary, if we didn't find a match, even the first time, the "Z" flag will be set. From a first time failure the "Z" flag will be set even after the return from the NODOC second level subroutine. In that event we have nothing more to do here, so we jump out to the generic exit routine at INDEX 11, (NDX11).

Moving our attention to Listing 4, the FDOS sends us back an offset of the matching document name's position in the sector buffer. The above code partially converts this offset into a literal address. The INDEX POINTER slave routine completes this conversion into a 16 bit literal address.

We next get the attribute mask back off the stack, but only a copy of it, putting it back on the stack for later use. We then get the value into the accumulator using a logical AND. Remember that the terminal column position is in the E register, and we didn't want to mess with it.

Now that we have the bit mask in the accumulator we check to see if we are to shoow private documents. The immediate argument of the compare instruction is our testing variable we loaded above, if we are to display private documents. If we are not to display private documents, then we jump out.

If we are to display the index entry then we get the terminal position counter, increment it, and put it back on the stack. We then check to see if we have reached the end of the display sequence, if four document entries have been displayed. If we have not written four entries then we save the flags and jump out.

As we move along to Listing 5, if we did fill the sequence, indicated by the logical AND above, then we clear a new line on the terminal and jump over the NDX4 code.

Here we write a separating string between the last document entry written, and the next one. Then we load a counter in preparation for the actual writing of the document name, and use NDXPTR to adjust the literal address for us.

(Continued on page 46)

Letters

(Continued from page 3)

Format and sysgen a blank disk, and place the following documents upon it from your CP/M master diskette.

WS.COM etc. (Wordstar itself, and its support documents.)
DDT.COM

Run Wordstar to determine which version you have. I am assuming you have either Version 3.0 or 3.3. The first visual screen should tell you which version you have. When this determination has been made exit Wordstar and return to the system level command prompt.

What we will do now is alter the default settings for the various Wordstar options. The table below shows the addresses of the control group which are within the scope of the stated problem. In this table the "ON" value also indicates the default settings assigned by MicroPro. See figure 1.

At the prompt enter the command shown below:

```
A0>DDT WS.COM
DDT VERS 2.2 (DDT's response)
NEXT PC
4600 0100
```

Now then, if you wish to enter Wordstar every time with the justification option turned off, enter the following DDT commands.

```
-S036E (you enter S036E <RET> - this is for Version 3.3)
036E 00 FF (you enter FF <RET>)
036F 00 (DDT responds)
```

In the above sequence we meet DDT at the "-" prompt, and tell it to "Set" the place to work at address 036E, which is a hexadecimal number. We always work in hexadecimal. DDT then responds with the address specified, followed by the byte in the location specified. You note that the value follows the table above. If you have any value, in this section, other than those shown in the table, something is wrong - abort the sequence! You now want to place the "OFF" value in this location so you enter FF and a return. DDT then advances to the next memory location.

The table shows other locations of interest, as I assume you will do a, "well, while I'm here," act. All of the locations above have either a "yes/no" option with the exception of the line spacing. If you constantly double space your text, then change this value to "2".

Next, we assume you will always enter Wordstar in the document mode, in any automated sequence. If you were a programmer you would want to change this to the FF, or non-document mode. This option is important as you may enter a document directly from the command line.

```
A0>WS MYFILE.TXT
```

The above example would place you into the document MYFILE.TXT, if it existed, or cause it to be created, the document mode entered, and the document prepared for your input.

When you have finished altering the various options, enter the following sequence at the DDT prompt.

```
XXXX . (that is a period character to get the prompt back)
-G0 (you enter G zero)

A0>SAVE 69 WS1.COM
```

Above we use the period character to exit the modification mode of DDT, and return to its normal prompt. Then we want to exit DDT by telling it to Go to location Zero, which is the "warm boot" jump vector for CP/M. This will leave the altered image in memory for us to deal with.

The next step in the sequence is to save the altered image into a usable document. We there SAVE the memory image, and tell the DOS that the image is 69 pages long. Note that we changed the name of Wordstar. If we didn't use a different name we would overwrite our original document. This is O.K. to do, as long as you didn't make a mistake. Let us leave the original document as it was until we are sure of what we have tested our work, eh? Once you are sure you have tamed le' beast then you may rename the altered version to whatever pleases you.

You will note I didn't even get close to altering printer codes. If you have to do a lot of weird printer control, like type setting or something, chuck Wordstar and get T/Maker!

Fine, that part of the problem has been taken care of. The next problem is to see if I actually have a virgin copy of CP/M in The Cave for the next step. Don't laugh, this could be a real problem! Virgin CP/M? Somewhere in here..., (crash, slam, bang...), the things I go through for my readers!

O.K. Now then, open up a document called WS.SUB, using the nondocument mode of Wordstar. This will be a small document. What we want to do now is enter a line of text equal to each lir of your normal start up sequence.

```
SMARTKEY FRENCH.DEF
WS $1
```

And that is all. Save the document and exit back to the CP/ command line. From your CP/M master diskette get tł document called "SUBMIT.COM," and place it on your workin disk.

Ver. 3.3 Address	Ver. 3.Ø Address	Function	Keyboard Code	Value ON	Value OFF
Ø36D	Ø385	word wrap	^OW	ØØ	FF
Ø36E	Ø386	justification	^OJ	ØØ	FF
Ø36F	Ø387	variable tabs	^OV	ØØ	FF
Ø37Ø	Ø388	soft hyphens	^OE	ØØ	FF
Ø371	Ø389	hyphen help	^OH	ØØ	FF
Ø372	Ø38A	show control char	^OD	ØØ	FF
Ø373	Ø38B	ruler line		ØØ	FF
Ø374	Ø38C	figure page breaks		ØØ	FF
Ø375	Ø38D	show page breaks		ØØ	FF
Ø376	Ø38E	line spacing		Ø1	FF
Ø378	Ø392	Entry mode		9Ø	FF

Figure 1

The first line of the submit document is the command line to load both SMARTKEY and the concerned definitions. If this is not the proper sequence alter it, using additional command lines as needed. The second line allows the passing of a document name to the SUBMIT program. To perform the desired functions enter the following at the command prompt.

A0>SUBMIT WS TEST.TXT

The system will whir and spit and then warm boot. After the warm boot each of the stored command lines will be placed before the prompt, and executed. If all has gone well you should end up in a fresh new document, in the document mode, with the justification turned off. The justification may be turned on in the normal way, should you need it.

Of course, if you had an AMPRO, using either the standard AMPRO ZCPR3 enhanced CP/M, or Hermit DOS Series 'E'' you could include these additional function in an "ALIAS," as shown below.

LDR MYTERM.Z3T;SMARTKEY FRENCH.DEF;WS

This example would send you to Wordstar ready to work every time you turned on the system. You might wish to locate a copy of the public domain program SYNONYM.COM, which may be used with standard CP/M systems. To be quite frank, what I remember of standard CP/M wouldn't be of much help. When you start talking advanced functions, you are talking a standard AMPRO!

Copyrights

Dear Tom;

In the first part of your "NEW-DOS" series you said the command processor came from the public domain. In the source listing I see where you have a copyright notice. How can you claim copy rights to something that is in the public domain?

James L.
Portland Oregon

Well James, you pose an interesting question. Copyright law is something I follow closely, and am very concerned about. There are a number of gray areas in copyright law, at least in my view point, areas that are hotly contested. But, let us deal with your specific concepts.

The original ZCPR was created by a group of fellows called the CCP GROUP, to whom I apologize for not remembering each of their names. This "ZCPR" had up to four official versions, and was written in 8080 code with the ASM assembler. When I asked people in the trade, like Mr. Thompson at Micro C, I was told that no Z80 version was available anywhere, the task of converting it to Z80 code was more than they wanted to attempt, and that they knew of no one foolish enough to consider such a project, as the task "was not a trivial one." To my knowledge the only true Z80 versions of this CCP available are of my authorship. The act of converting the code to a standard Zilog format in itself is enough to avoid copyright law. The other changes I have made to the program take me further and further away from the base, moral concepts presented by the original work.

Where to draw the line, especially in computer programs, is difficult to determine. On the one hand, if we followed the letter of the concept there would be no technical advancement. Each individual would have to reinvent the wheel, only one person could make a television, or create ice cream. However, to personally deal with these issues you will note that I claim copyrights to the specific version presented, and have placed this version, with its evolution intact, back into the public domain. I use another version entirely for my public domain offering of Hermit DOS, and yet several others in possible commercial offerings. I feel secure that I have not intruded upon any other person's rights. Further to the point, my legal advisors give me a hard time about this decision and most of the wares I place into the public domain with commercial potential.

The concept of copyrights, though often commented upon, is not really understood, especially as might be applied to computer programs, where techniques are well defined, restrictive, and the numbers of instructions are limited.

In the early days of computers people filed form TX with the Copyright Office of the Library of Congress, only to see their copyrighted works appear on the open market as a retail offering. All people had to do was get a copy of the source code from the Library of Congress and they were in business. This problem was approached in revisions of copyright law in 1976.

Under the Copyright Act of 1976, which covers computer programs and other printed material, a creation has a copyright, belonging to the author at the time it becomes fixed in a permanent medium and is no longer just an idea. This includes all forms of electronic medium, such as a disk. This Act, coupled with revisions in the Trade Secrets Act, allowed the author to secure his copyrights, without revealing the actual source code.

The law is such that, at my level of function, my legal advisors demand that my equipment automatically include a copyright notice every time I open a document for editing. We all enter a realm of responsibility when we create new works based upon established technology. The letter of the law states that the end product may not reasonably appear to be a copy of the original work. This is hardly a difficult law to avoid, or circumvent. It is the individual's responsibility where to draw the line when considering the rights of another.

A case in point is the BIOS I use for my various versions of Hermit DOS. The basic version of this BIOS when it was written in 8080 code was nearly byte for byte that of the CBIOS code listing in the CP/M integrator's manual. AMPRO changed this code "no more than was needed," and admitted this fact in their source listing. AMPRO's contribution to this BIOS was slight, involving the floppy drive controller and the "E" drive, yet they lawfully claimed copyrights to this BIOS. My first act was to convert this code into standard Zilog format. The resultant code, especially when optimised with instructions the 8080 microprocessor could not use, which were Z80 specific, not only met, but exceeded the letter of the law regarding reasonable appearance of a copy of the original source code. The current version of this BIOS has conditional assemblies for everything from restoring the cold boot command, ZCPR3 compatible data tables, to voice synthesis peripheral handling routines. The original source code has very little resemblance to the current BIOS. You will note however, that all my systems sign on with the notice that the BIOS is the property of AMPRO Computers.

AMPRO, with their corporate power could never hope to win an infringement case against me for this BIOS. In fact, I have not yet even sent them a copy of the current version. The BIOS is not only legally, morally, but ethically mine. Yet I give AMPRO copyrights to my work, why? I don't work for AMPRO, but AMPRO has gone out of their way to treat me very well. It is out of respect for their investment in this BIOS, the original version they don't even use any more, it being nearly four versions old. There is the element of respect for the PEOPLE at AMPRO that I not only give them the copyrights to this work, but use a version which does not compete with their commercial product offerings. In return for this respect of their rights, AMPRO gives me the respect, and latitudes I require to place my works into the public domain. Without this mutual respect the amount of support we will be giving AMPRO users just would not be possible.

The bottom line, in this lengthy oration is, after all respect. If we respect the rights of others we have no need for laws. The law comes into play only when people cannot respect each other. When we look beyond the letter of the law, and assure that we do not knowing hurt another person in any way, we know we are doing the right thing. But yes, I feel most secure in claiming copyrights to my version of a concept found in the public domain. When I do so, I am not only well within my legal rights, but am notifying the often unknown original author of the concept that I will carry on with what he, or she started with the original version. My contribution assures the original gift to us all will not be discarded as time passes, but also assures the new version will remain available to everyone. This is what the spirit of the public domain is all about. ■

Editor

(Continued from page 1)
issue of Electronic Engineering Times for more information.

A very useful new chip for RS-232 interfacing is the Maxim MAX232 which provides two RS-232C transmitters and two RS-232C receivers, but only requires a +5 volt supply. It greatly simplifies layout and power supply design by combining both the transmitters and receivers in one chip and eliminating the requirement for the positive and negative 12 volt supplies.

Artificial Intelligence

One of the areas of computer applications everybody talks about is artificial intelligence, but so far it has been more talk than action. Now Carl Landau (Publisher of Computer Language) has launched "AI EXPERT" for the growing field of artificial intelligence. Their premier issue debuts in July, with monthly issues beginning in October, and you can obtain more information from Carl at C.L. Publications, 650 Fifth Street, Suite 311, San Francisco, CA 94107. We wish Carl the best of luck with his new venture because it is needed. We'll be watching and reporting on AI EXPERT.

Another entry in the AI field is Borland's Turbo Prolog. Borland expects to sell even more Turbo Prolog packages than they have for Turbo Pascal! If they can accomplish this, it will greatly expand the AI activity. This is another area that we'll be watching and reporting on.

Be sure to let TCJ know what you are doing (or thinking about) in the AI field. Articles, letters, short notes...anything is welcome,

Give IBM The Credit They Deserve

We all find it easy to criticize what someone else has done, and heaven knows that I do it often enough, but we seldom recognize their accomplishments if they differ from our ideas. A case in point is the IBM PC. The press (TCJ included) frequently points out apparent faults in the PC without spending equal time discussing its strong points. It is time that we stop and give IBM credit for what they have accomplished. Don't get me wrong, I don't agree with everthing that they have done, but that doesn't mean that they were wrong. We all have 100% accuracy when deciding what should have been done 2-4 years ago, but our accuracy for deciding what should be done for the future is very poor.

When the historians analyze the development of the computer, they will undoubtedly assign pre-PC and post-PC as one of the major demarcations because IBM's entry made a pronounced change in the market. In the pre-PC era, there were many smaller companies producing non-compatible systems, and it took someone with the power and credibiltiy of IBM to establish a standardized product acceptable to business. One of the pre-PC leaders was the Apple II+, but it was designed as a game machine and Apple did not make the changes necessary to satisfy the business market. The other leading contenders were designed around the CP/M operating system, but there were too many different implementations and disk formats, and it took a systems programmer to reconfigure the BIOS to implement different interfaces and drivers.

There was a need, and the people in the industry went their various ways instead of cooperating to design a standard easy to use product. They also went out of business, leaving unsupported orphans. In fact some of their products were orphans even while they were in production because they didn't answer customer's calls or letters. I went so far as to write to the President of Morrow trying to get the answer to a simple question, but never received a reply. I'm sure that if you wrote to the president of IBM you would get a reply—in fact if you wrote to the president of KMART your inquiry would not go unanswered! Too many companies just didn't give a damn about their customers (incidently Morrow has filed chapter 11).

IBM's strong points are marketing and customer service, and that's what it took to establish a standard. Others in the business could have worked together on standardization (and also provided customer support); but they didn't, so we can't blame IBM for filling a need. That's just good business. Some people complain that no one company should be so powerful that other companies are forced to follow their lead, but it took a very powerful leader to enforce standardization. Without IBM's lead the microcomputer industry would still be disorganized and much smaller.

The IBM-PC was not a technological breakthrough, it was their strong marketing and support that was needed. I don't like the PC, but I recommend it because of the vast amount of software and peripherals available for it—and that's also the reason that I'll get one.

It's time to stop bitching about IBM's business practices and the PC's technical design, and acknowledge the benefits of their leadership. IBM did what others could have done but didn't, and we should be thankful for what they have done. If someone doesn't like what IBM is doing, all they have to do is to produce a better product with better support. If they can't (or won't), then give IBM the credit they deserve and follow the leader or shoot for a different market.

The LB Can Tell Time

Kenmore Computer Technologies (PO Box 635, Kenmore, NY 14217) has sup-

(Continued on page 47)

ZTIME-I

A Realtime Clock for the AMPRO Z-80 Little Board

by C. Thomas Hilton

Many of the requests I get here at The Cave regard time functions, and the need for a real time clock has been a problem. The clock in the Kaypro never has worked as well as I thought it should, and reminded me of an old Chevy I had once... I have even played with the software clock, (on library disk UTILITY.002), developed at the University of Missouri. The applications which have been presented to me have, however, demanded something better than a "toy clock," which is not to be inferred as degrading any other time keeping system. What I am saying is where computer security and accurate time/date stamping is required, there is no room for compromise.

When Art told me he was sending down the ZTIME-I enhancement card kit, I wasn't expecting a great deal. I have seen so many time keeping schemes, I am very hard to impress.

Assembly Instructions

Anyway, I plugged in the old soldering iron, and settled back with the first cup of coffee for the day, and read the directions. Yes, I always read the directions first—anyone who didn't think I was nuts before now has their proof! Give the designer a break, read his instructions! Reading the directions and support data took me about an hour, so you can forget my usual hand holding assembly instructions for this project. There is nothing I am going to be able to say which would improve the assembly directions provided by Kenmore Computer Technologies, the makers of ZTIME-I.

Documentation

While the kit is relatively simple to assemble, and the card is small, I would not advise "winging it' with this kit. The documentation includes a 27 page manual with a full size schematic plus three sets of support documents each of two pages length. A novice will surely be able to assemble it without the slightest problem, if the directions are followed. I am really not one to wade through documents as large, and complete as the ones provided for this clock kit, but the information I needed literally jumped off the page at me. I had no trouble finding everything which applied to the AMPRO Little Boards in the documentation.

Support Software

The card has been on line for several hours now, and I have played with the support programs enough to make a simple time stamp program for new text documents. The disk which was sent to me was in Kaypro II format, which the AMPRO MULTIDSK utility had not the slightest problem reading. This helped my mood somewhat. I was surprised to see that the people that put this kit together didn't screw around. There are versions of the utilities in the "C" language, Mbasic, JRT Pascal, (which I thought was real nice of them as we have JRT Pascal in our library), and in Turbo Pascal. There seems to be a version for nearly every programmer, and enough cross reference material to put the card on line in the shortest possible time. Impressive.

Installation

The generic version works as is for the AMPRO systems. When I read the assembly instructions about how to cut out the Z80 processor I was not thrilled, but a quick check showed that AMPRO had planned for "daughter" boards and my mood really improved, the Z80 is socketed! At that moment I could have kissed the folks at AMPRO!

In my last AMPRO column I discussed how to disassemble the Series 100 AMPRO systems, so will not repeat these instructions here. Disassembly is required to access the Little Board in the stock Series 100 enclosure.

For Little Board 1A users the ZTIME-I card just plugs right into the Z80 socket, the Z80 is in turn placed into the provided socket in the ZTIME card. For most applications no support nor additional card insulation seemed required on my test board. If you don't like the idea of a card resting atop other chips, and would feel better with some insulation, a thin strip of double sided adhesive foam will make you feel better. This double sided adhesive foam is available at your local Radio Shack. Again, I didn't think it was needed in normal use. Where your application has vibration considerations the application of double sided foam should be considered.

For those of you who are using the 1B CPU card, we do have a problem, which is simply addressed. There are two jumper pads near the Z80 chip, marked as jumpers 5 and 6. These will touch the bottom of the board and must be dealt with. In the case of jumper pad 5, I took off the shunt "box," and using a small piece of wire wrap wire, just wire wrapped the concerned shunt pins together. Both sets of jumpers I then bent carefully out of the way at a 90 degree angle. If you have no intention of ever changing these jumper settings I would advise cutting off the open set of jumper pins, and soldering the concerned pins of any other jumper pads together and then cutting them off as well. This will get them out of the way, while still allowing possible changes later. For myself, I have a responsibility to you, my readers, to keep my system as close to "factory condition" as possible, hence just bent them out of the way.

The next bad point for 1B CPU users is the length of the ZTIME card. It rests upon both an electrolytic capacitor and the oscillator module on my card, which does not allow the card to set "straight," but on an angle. This also did not allow the interface pins to set as firmly, and as "perfect" into the Z80's old socket as I would have liked. This application is, however, very workable, electrically, and structurally sound, even if not overly pleasing to my perfectionist eyes. With tension cards in the drives I gave the little AMPRO a healthy dose of "The Hermit's Shake, Rattle, and Roll," and everything stayed where it was, so I am satisfied with the installation.

EDITOR'S NOTE: The KCT manual suggests using a forty pin IC socket between the ZTIME-I board and the main board so that the board stands high enough to clear the components under it.

I wouldn't consider shortening the board for installation in the 1B systems.

The ZTIME has an on card interrupt line which goes to the offending portion of the card, which you may want to use for something later, and may need a little bit of blank card space for mounting. This interrupt, called "SI" on the card, is a standby interrupt system, which may be used to trigger a relay for actually powering up the Little Board at a preprogrammed time, or other control applications. This is a most unexpected feature, and a pleasant one, though I had hoped I wouldn't be tempted to get into hardware timing experiments.... (sigh).

Operation

For the AMPRO Little Boards no reconfiguration of the port addressing is required, unless you desire it. This means all of the support programs on the demo disk will work without configuring. The two operations programs of concern are DATE.COM, and SETDATE.COM. Each of the programs do just about what their names say.

SETDATE will allow you to set the date and time of the system, which actually does keep working with the power off. The manuals mention that you may want to add trimming capacitors to finely adjust the accuracy of the clock system. Due to the coverage given this topic I was a bit concerned about just how well the card would keep time. Since I began this article I have stopped, had a couple tacos, called AMPRO for a very nice conversation with Bill Dollar, the President of Ampro, and done several other tasks. Thus far the ZTIME card keeps time as well as my CASIO "DATABANK" LCD watch. I am inclined to think the time keeping concerns are over rated. They seem to be concerned with an error rating of 25 pulses per million, and I do not think this is a major concern for any but the most demanding user. SETDATE allows very accurate setting of the system. All system definitions are entered into the program, but the clock chip is not set until you press a key, allowing you to preset the definitions, and then wait for real time to catch up.

DATE.COM does little more than return the complete day, month, and time in consumer format, which is all it is supposed to do.

Programming

At this sitting I have not done a great deal of programming with the system, as just about all I want to do with it is provided for me. There is little doubt, however, that I will have more to say about possible applications of this system, and I would, as always, like to hear about your experiences with ZTIME-I. I did make a few quick changes to the TURBO Pascal access program to allow me to open a file and insert the time/date stamp before editing, but that isn't much of an accomplishment. I did notice one bad point when you begin working with the TDATE.PAS source code, which is the access program for TURBO. Be sure to set the compilation "end" address low in memory. When I compiled the program in the normal way WordStar returned the "no more room at the inn" type error,

ZTIME Listing 1.

```
{  Turbo Pascal version of Date program. Written by: Alan D. Percy
   Date Written: 7/12/84
   Prints date and time from getdate routine.}

Type
  str80 = string[80];
Var
  darry : str80;     {get ASCII string in here}

FUNCTION getdate : str80;
{  Gets an ASCII representation of the time and date as follows:
   Mon Jan 23 10:51:32 AM
   calls: rawdate and number}

Type
  rawtype = array[1..6] of integer;
Var
  rdate : rawtype;    {place to get raw date into}
  tmp : str80;        {place to build string into}
  pmflag : boolean;    {AM/PM flag}

FUNCTION number(num : integer; flag : boolean) : str80;
{  This function returns a string with the character representation of the
   number passed as the parameter 'num'. The parameter 'flag' determines
   whether leading zeros should be given or not (false = no, true = yes).

Limitations: Only a two digit number can be converted. }

Var stmp : str80;
  tmp : integer;

Begin {of number routine}
  tmp := ord('0') + num div 10;   {figure first character number}
  stmp := char(tmp);          {put in string}
  if (stmp = '0') and not flag then {if flag not set and leading zero}
    stmp := ' ';              {make blank}
  tmp := ord('0') + num-10*(num div 10);  {figure second character number}
  stmp := concat(stmp,char(tmp));      {tack on to string}
  number := stmp
End;  {of number routine}

PROCEDURE rawdate(var raw : rawtype);
{  Fills a 6 element array of integers with the decimal time from the
   MM58167 clock chip addressed at the port 'cbase' constant. The form
   of the array is as follows:

   Element    Contents (in decimal)
   --------   --------------------
     1        Month number (1-12)
     2        Day of the month (1-31)
     3        Day of the week (1-7)
     4        Hours (0-23)
     5        Minutes (0-59)
     6        Seconds (0-59)
}
const
  cbase = $0e0;
var
  i, tmp : integer;
  bcddate : rawtype;
  flag : boolean; {true when read twice the same}
begin {of rawdate routine get array filled with BCD value quickly
    ( < 1 second) }

  repeat {until read twice the same}
    for i := 1 to 6 do {read the array in BCD the first time}
      bcddate[i] := ord(port[cbase + 8-i]); {get BCD value from chip}
    flag := true; {assume we get it correct again}
    for i := 1 to 6 do {try again}
      if bcddate[i] < > ord(port[cbase + 8-i]) then
        flag := false {if not the same clear flag and try again}
  until flag;

  {convert from BCD to decimal at our leisure}

  for i := 1 to 6 do Begin
    tmp := bcddate[i] div 16;
    raw[i] := tmp * 10 + (bcddate[i] - tmp * 16)
  end
end;  {of rawdate routine}

Begin {of getdate routine}
  rawdate(rdate);     {read date and time from port}

  case rdate[3] of
    1 : tmp := 'Sunday, ';
    2 : tmp := 'Monday, ';
    3 : tmp := 'Tuesday, ';
    4 : tmp := 'Wednesday, ';
    5 : tmp := 'Thursday, ';
    6 : tmp := 'Friday, ';
    7 : tmp := 'Saturday, ';
    else
      tmp := '*** ';
  end;

  case rdate[1] of
    1 : tmp := tmp + 'January ';
    2 : tmp := tmp + 'February ';
    3 : tmp := tmp + 'March ';
    4 : tmp := tmp + 'April ';
    5 : tmp := tmp + 'May ';
    6 : tmp := tmp + 'June ';
    7 : tmp := tmp + 'July ';
    8 : tmp := tmp + 'August ';
    9 : tmp := tmp + 'September ';
    10 : tmp := tmp + 'October ';
    11 : tmp := tmp + 'November ';
    12 : tmp := tmp + 'December ';
  else
    tmp := tmp + '*** '
  end;
```

meaning there wasn't enough memory to run my TIME, and STAMP programs. Lowering the "end" address down to about A000H solved all of these problems.

However you end up modifying the demo programs, they work exceptionally well when included in a Hermit DOS, or ZCPR3 "alias." The environment does not need to be loaded prior to using the clock, so a rather pleasing, and professional looking startup procedure may be had with very little effort. The TURBO Pascal, public domain, clock system access program is shown in Listing 1.

As may be seen from the above example, the reading of the clock system is very straight forward, and ripe for many many expansions and enhancements to meet your every fancy.

ZTIME Listing continued.

```
    tmp := tmp + number(rdate[2],false) + ' '; {add on day of the month}
    if rdate[4] > = 12 then begin {if after 12 pm convert from military time}
      pmflag := true;
      if rdate[4] > 12 then
        rdate[4] := rdate[4] - 12
    end
    else begin
      pmflag := false;
      if rdate[4] = 0 then   {if 12 am}
        rdate[4] := 12
    end;

    tmp := tmp + number(rdate[4],false) + ':';  {put hour in}
    tmp := tmp + number(rdate[5],true) + ':';   {put minutes in}
    tmp := tmp + number(rdate[6],true);         {put seconds in}

    if pmflag then
      tmp := tmp + ' PM'
    else
      tmp := tmp + ' AM';

    getdate := tmp

  end; {of getdate routine}


  Begin {of date program}
    darry := getdate;   {get ASCII for of date and time}
    writeln('It is now: ',darry)
  end.
```

Conclusion

I should mention that the price of the Z-TIME is $69.00, plus $3.00 shipping and handling. This price seemed a bit steep to me, on first thought, for a clock system. However, when I calculate the time in development that would be required without the extensive documentation, software support disk, chip specification sheets, and full size schematics, I feel the asking price is not that unreasonable. When I then add to this calculation the ease and speed with which the system went together, and became functional, I see what the price is really for; it was nice of them to include the hardware. To go a step further, Kenmore also has an applications support network with user disks, and information exchange. According to the documentation I received the charge is $20 for the current applications disks. You may obtain a copy of the current applications software listing by sending a self addressed stamped envelope to:

Kenmore Computer Technologies
ZTIME-I A.U.G.
P.O.B. 635
Kenmore, New York 14217

I feel as though I have been a bit "light" on the nuts and bolts of this time keeping system. The reason for this is simply because the documentation provided with the system requires no more added information than what has been provided here. I felt I should present information which would help you to decide if you need, or want, this level of time keeping capabilities on your Little Board. If having time/date functions is just a novelty, or a supplementary need, you may be better served with the new AMPRO BIOS Version 3.9, with its clock simulation routines. If, on the other hand, you need serious time keeping functions, then I highly recommend the ZTIME-I clock system. I require the reliability presented by this type of time keeping system for my high security, on line, information systems. In my opinion, if you have any type of time applications beyond "appliance" type computing, you should take a serious look at the ZTIME-I system. ■

ZTIME-I Software Available

KCT has supplied the ZTIME-I software and user programs to be placed on our BBS. They are also available on AMPRO DSDD disk for $10.

* * *

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used marks are acknowledged, and we apologize for any we have overlooked.

Apple II, II +, IIc, IIe, Macintosch, DOS 3.3, ProDOS; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. MBASIC; Microsoft. Wordstar; MicroPro International Corp. IBM-PC, XT, and AT; IBM Corporation. Z-80, Zilog. MT-BASIC, Softaid, Inc. Turbo Pascal, Borland International.

Where these terms (and others) are used in The Computer Journal they are acknowledged to be the property of the respective companies even if not specifically mentioned in each occurence.

Back Issues Available:

Volume 1, Number 1 (Issue #1):
- *The RS-232-C Serial Interface, Part One*
- *Telecomputing with the Apple][: Transferring Binary Files*
- *Beginner's Column, Part One: Getting Started*
- *Build an "Epram"*

Volume 1, Number 2 (Issue #2):
- *File Transfer Programs for CP/M*
- *The RS-232-C Serial Interface, Part Two*
- *Build a Hardware Print Spooler, Part One: Background and Design*
- *A Review of Floppy Disk Formats*
- *Sending Morse Code With an Apple][*
- *Beginner's Column, Part Two: Basic Concepts and Formulas in Electronics*

Volume 1, Number 3 (Issue #3):
- *Add an 8087 Math Chip to Your Dual Processor Board*
- *Build an A/D Converter for the Apple][*
- *ASCII Reference Chart*
- *Modems for Micros*
- *The CP/M Operating System*
- *Build a Hardware Print Spooler, Part Two: Construction*

Volume 1, Number 4 (Issue #4):
- *Optoelectronics, Part One: Detecting, Generating, and Using Light in Electronics*
- *Multi-user: An Introduction*
- *Making the CP/M User Function More Useful*
- *Build a Hardware Print Spooler, Part Three: Enhancements*
- *Beginner's Column, Part Three: Power Supply Design*

Volume 2, Number 1 (Issue #5):
- *Optoelectronics, Part Two: Practical Applications*
- *Multi-user: Multi-Processor Systems*
- *True RMS Measurements*
- *Gemini-10X: Modifications to Allow both Serial and Parallel Operation*

Volume 2, Number 2 (Issue #6):
- *Build a High Resolution S-100 Graphics Board, Part One: Video Displays*
- *System Integration, Part One: Selecting System Components*
- *Optoelectronics, Part Three: Fiber Optics*
- *Controlling DC Motors*
- *Multi-User: Local Area Networks*
- *DC Motor Applications*

Volume 2, Number 3 (Issue #7):
- *Heuristic Search in Hi-Q*
- *Build a High-Resolution S-100 Graphics Board, Part Two: Theory of Operation*
- *Multi-user: Etherseries*
- *System Integration, Part Two: Disk Controllers and CP/M 2.2 System Generation*

Volume 2, Number 4 (Issue #8):
- *Build a VIC-20 EPROM Programmer*
- *Multi-user: CP/Net*
- *Build a High-Resolution S-100 Graphics Board, Part Three: Construction*
- *System Integration, Part Three: CP/M 3.0*
- *Linear Optimization with Micros*
- *LSTTL Reference Chart*

Volume 2, Number 5 (Issue #9):
- *Threaded Interpretive Language, Part One: Introduction and Elementary Routines*
- *Interfacing Tips and Troubles: DC to DC Converters*
- *Multi-user: C-NET*
- *Reading PCDOS Diskettes with the Morrow Micro Decision*
- *LSTTL Reference Chart*
- *DOS Wars*
- *Build a Code Photoreader*

Volume 2, Number 6 (Issue #10):
- *The FORTH Language: A Learner's Perspective*
- *An Affordable Graphics Tablet for the Apple][*
- *Interfacing Tips and Troubles: Noise Problems, Part One*
- *LSTTL Reference Chart*
- *Multi-user: Some Generic Components and Techniques*
- *Write Your Own Threaded Language, Part Two: Input-Output Routines and Dictionary Management*
- *Make a Simple TTL Logic Tester*

Volume 2, Number 8 (Issue #12):
- *Tricks of the Trade: Installing New I/O Drivers in a BIOS*
- *Write Your Own Threaded Language, Part Four: Conclusion*
- *Interfacing Tips and Troubles: Noise Problems, Part Three*
- *Multi-user: Cables and Topology*
- *LSTTL Reference Chart*

Volume 2, Number 9 (Issue #13):
- *Controlling the Apple Disk][Stepper Motor*
- *Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part One*
- *RPM vs ZCPR: A Comparison of Two CP/M Enhancements*
- *AC Circuit Anaysis on a Micro*
- *BASE: Part One in a Series on How to Design and Write Your Own Database*
- *Understanding System Design: CPU, Memory, and I/O*

Issue Number 14:
- *Hardware Tricks*
- *Controlling the Hayes Micromodem II From Assembly Language*
- *S-100 8 to 16 Bit RAM Conversion*
- *Time-Frequency Domain Analysis*
- *BASE: Part Two*
- *Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part Two*

Issue Number 15:
- *Interfacing the 6522 to the Apple][and][e*
- *Interfacing Tips and Troubles: Building a Poor-Man's Logic Analyzer*
- *Controlling the Hayes Micromodem II From Assembly Language, Part Two*
- *The State of the Industry*
- *Lowering Power Consumption in 8" Floppy Disk Drives*
- *BASE: Part Three*

Issue Number 16:
- *Debugging 8087 Code*
- *Using the Apple Game Port*
- *BASE: Part Four*
- *Using the S-100 Bus and the 68008 CPU*
- *Interfacing Tips and Troubles: Build a "Jellybean" Logic-to-RS232 Converter*

Issue Number 17:
- *Poor Man's Distributed Processing*
- *Base: Part Five*
- *FAX-64:Facsimile Pictures on a Micro*
- *The Computer Corner*
- *Interfacing Tips and Troubles: Memory Mapped I/O on the ZX81*

Issue Number 18:
- *Interfacing the Apple II: Parallel interface for the game port.*
- *The Hacker's MAC: A letter from Lee Felsenstein*
- *S-100 Graphics Screen Dump*
- *The LS-100 Disk Simulator Kit: A product review.*
- *BASE: Part Six*
- *Interfacing Tips & Troubles: Communicating with Telephone Tone Control*
- *The Computer Corner*

Issue Number 19:
- Using The Extensibility of FORTH
- Extended CBIOS
- A $500 Superbrain Computer
- Base: Part Seven
- Interfacing Tips & Troubles: Part Two Communicating with Telephone Tone Control
- *Multitasking and Windows with CP/M:* A review of MTBASIC
- The Computer Corner

Issue Number 20:
- Build the Circuit Designer 1 MPB: Designing a 8035 SBC
- Using Apple II Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering and Other Strange Tales
- Build a S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- The Computer Corner

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures and Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition and Control: Connecting Your Computer to the Real World
- Build the Circuit Designer 1 MPB: Part 2 - Programming the 8035 SBC
- The Computer Corner

Issue Number 22:
- *NEW-DOS: Write your own operating system.*
- *Variability In The BDS C Standard Library*
- *The SCSI Interface: Introductory Column to a Series.*
- *Usinf Turbo Pascal ISAM Files.*
- *The AMPRO Little Board Column*
- *The Computer Corner*

Issue Number 23:
- C Column: Flow Control & Program Structure.
- The Z Column: Getting Started With Directories & User Areas.
- The SCSI Interface: Introduction to SCSI.
- NEW-DOS: The Console Command Processor.
- Editing The CP/M Operation System.
- INDEXER: Turbo Pascal Program to Create Index.
- The AMPRO Little Board Column.
- The Computer Corner.

New Products

True RS-232 from +5V Supply

Maxim has introduced a dual RS-232 receiver/transmitter that meets all EIA RS-232C specifications while using only a +5V power supply instead of the +12V and −12V supply usually required. The MAX232 consists of two transmitters, two receivers and two onboard voltage converters. The device is designed to replace the ±12V power supplies and two interface I.C.'s commonly used in RS-232C communication links.

The MAX232 has three sections; a +5V to +10V, and a +10V to −10V charge pump voltage converter; dual RS-232 transmitters, and dual RS-232 receivers. The ±10V charge pump voltage converters use four external electrolytic capacitors, and the converter's chopping frequency is internally set to 16kHz. The transmitter inputs are TTL and CMOS compatible, with the logic threshold set at 1.3V for +5V Vcc. The inputs have internal pull up resistors that force the output of an unused RS-232 transmitter low when the input is not connected. The unloaded RS-232 output swing of the MAX232 is from −10V to 0.6V below the +10V. The RS-232 output is guaranteed to meet EIA RS-232C specification of a minimum output voltage swing of ±5V under the worst case condition of output loading, ambient temperature and power supply voltage.

The MAX232 receivers also conform to the RS-232C specifications, with an input impedance between 3K ohms and 7K ohms. Each input can withstand ±30V even when the device has no power applied. The input switching thresholds are within the ±3V limits of RS-232C and are also TTL and CMOS compatible.

Applications for the MAX232 include all RS-232C communication links, especially where the ±12V power supplies required for the 1488 and 1489 devices are not available and would have to be added at additional cost and space. Contact Maxim Integrated Products, 510 N. Pastoria Ave., Sunnyvale, CA 94086, phone (408) 737-7600 for spec sheets and more information. ■

BV Adds Engineering Software

BV Engineering has added a number of useful products to its already extensive product line. A brief summary of several new products follows, and their just-released Software Catalog #5 is available on request from BV Engineering, 2200 Business Way, Suite 207, Riverside, CA 92501, phone (714) 781-0252.

PDP Plotter Driver Program PDP makes multi-color scientific and financial graphs on pen plotters, and the data may be entered manually or come from previously generated data files. The data files can originate from BASIC, FORTRAN, and PASCAL programs, word processors, text editors, or other BVE software such as ACNAP, SPP, etc. Data from different files may be plotted on the same graph.

PDP will connect data points when told to do so and draw legends on each data point with unique legends per plot. Dotted, dashed, or solid lines may be mixed on a single graph. A background of grid lines can be drawn with 2, 4, 10, or 12 linear divisions, or any number of logarithmic divisions.

Versions of PDP are available under PC/MSDOS and CP/M for many popular plotters including the Hewlett-Packard 7440, 7470, and 7475, the Gould Colorwriter 6120 and DS-10, the Sweet-P 100 and 600, GraphTec MP-1000, Houston Instruments DMP-29, Radio Shack FP-215, Sharp CE-515, and the Mannesmann Tally Pixie 3. The regular PDP is priced at $72.95, and the 8087 version is $82.95.

In order to provide support to a greater variety of plotters, BVE has offered any of TCJ's readers a free copy of the PDP Plotter Driver Program if they will loan BVE a user's manual and plotter which they do not already support. BVE will pay insured shipping costs both ways via U.P.S. and will not keep the plotter more than 10 days. Interested readers must first contact BVE as they do not want to be flooded with plotters of the same make.

TEKCALC Scientific Calculator Program TEKCALC is a programmable scientific calculator for solving complex mathematical problems. Built-in graphics, statistics, user extendable functions, a data table window, and compatibility with other BVE software makes TEKCALC an indispensable tool for sophisticated data analysis. Trigonometric, logarithmic, expontential, special and user-definable functions are solved quickly. Formulas, computations, and data may be saved on disk permitting the creation of a customized math environment.

Templates containing normal mathematic statements to solve complex problems can be saved to disk, which allows for separate customized calculator environments for different fields, tasks, and professions. Mathematical functions, data files, and data tables may be plotted on the screen in bit mapped or character graphics with full labeling. Both auto-scaled and forced scaling are supported, and plots may be dumped to any graphics printer that currently supports your graphics screen dump. Linear regression and standard deviation calculations are easily made using simple TEKCALC commands, and will plot the calculated linear regression line against the raw data.

TEKCALC files are compatible with other BVE software products and ASCII files generated by BASIC, FORTRAN, PASCAL, etc. Values are calculated to 80 bits of accuracy with or without the 8087 numeric co-processor. TEKCALC is priced at $72.95 for the regular version, $82.95 for the 8087 version, and runs on IBM-PCs and compatible PCs with compatible Color Graphics Adaptor. ■

Computer Corner

(Continued from page 48)

registers and handshakes. To make it run well, requires a number of wait for this and that's before next. I guess these new chips are better than the old ones for most operations but not all.

Lastly I must admit that I have actually started to consider buying a clone system. A fellow worker and I have been bemoaning the fact that so many possible clients are using clones now that we need to buy one just to be able to do work for them. I have also had several calls at work to go and fix AT systems, and that means it will not be too long before I will find myself needing to know all the special quirks of these systems. It is a lot like a BMW mechanic who has the only shop in town, if you want to stay in business you had better also repair junkers.

My reasoning for getting rid of my Z100 was its slowness on booting. The PC's are no speed machine when it comes to booting either. If a full 640K or more of memory is present, these machines can take several minutes to boot. My Z80 machines usually takes about 1 to 2 seconds to boot, so it is possible to see why it bothers me. Now for people using hard disks, I certainly agree that treed directories and lots of other special functions are needed. Some people like myself, use sorted floppy disks to separate things, and find the extra features of MSDOS 2.X a real nuisance. The Atari 520ST also has GEM and can take considerable time to load if there are several accessory packages. This whole idea of mine to develope a Forth based monitor, now is heading into a tight (32K max) operating system for the ST. My reasoning for the direction is the need for a simple means of interfacing to the hardware of the unit. Suppose you want to use the ST for a dedicated controller, possibly a bulletin board, where fancy graphics is a big waste. Later this year the price of 520's will hit $200 to $300 each and thus they will make great (and cheapest) 68K controllers.

Our club bulletin board resets complety with each caller. This resetting guarantees a fresh restart for each caller and thus solves lots of hang-up problems. Systems that have long boots would therefore hamper the system's effectiveness. Tight and compact operating systems for developers are a must, especially during the development cycle. Our 520 crashes a lot, and the boot time can get very frustrating. This all reenforces my idea of a fast and simple operating system, something no more complex than the orginal CP/M2.2. Actually the best idea would be a monitor system based on Forth, with standard BIOS type entry points to interface to. A few comands like DIR and SAVE might be enough. If things go well, I may be able to push my activities in that direction next, at least that's what this hacker is trying to do. ■

NEW-DOS

(Continued from page 32)

NDXPTR returns the next character to us in the accumulator. We then mask off the upper bits, and then check to see if there is actually something to be written. If there is, then jump out.

If there isn't a valid character then get a copy of the terminal position off the stack and see if we have finished. If we have not finished then jump out.

We next advance to the document type field, again using the NDXPTR routine, and check to see if we have a document type to write. If not then jump out.

In the NDX7 code segment above we pad the terminal screen with spaces when there isn't a valid character left to work with. When we are finished with the name portion, we add an extra space, (below), between the name and the type. The period, ('.'), is used only for human communications. Then we loop all the way back up to NDX6 until we have the complete index entry written.

When we get to the NDX9 code we have finished writing an index entry, so we clear the stack, check the terminal for an operator abort indication, then get the next index sector, if any, and go all the way back up to INDEX3 to write the entry upon the screen until there are no more entries to be displayed, or the operator aborts the procedure. In both cases we end up at the NDX11 junction.

Our exit routine is very simplistic. We just cleanup the stack and return to caller. The routines which follow may be used by many procedures, but were placed in the index procedure as it was the procedure which called them first. When you go about modifying the procedures in this section be sure not to delete a routine which may be used by others.

The two index searching routines in Listing 6 are very simplistic. To provide the FDOS with the data required we have various entry points based upon the amount of data previously presented. The search services require the address of the caller's DCB, and the DMA buffer address, (which is set before making these calls and remembered by the FDOS segment). If no document is matched the FDOS will return a 0FFH status code in the accumulator. By incrementing an eight bit register, with the value 0FFH in it, we cause it to "roll over," or assume a zero value, which will set the "Z" flag. If the status is other than 0FFH then the "Z" flag will not be set.

The support routines which follow finish converting the sector offset, returned by the FDOS in response to a search request, into a 16 bit literal address. The character at that address is loaded into the accumulator and returned to the caller.

TRMSTAT is a very generic terminal status check. If there is no activity at the terminal then the routine will return to the caller with the "Z" flag set. Because the FDOS will get any character during its status checking procedure, we clear the FDOS internal character register of the key press character returned. We assume, for generic usage, that the key the operator pressed was intended to get our attention, not to have any special meaning.

Because we do have single character questions to ask the operator in a number of procedures, when we do get the terminal character, either as a result of a terminal status check, or to get a response, we convert the character to upper case for internal processing, it saves time and code space in the long run.

The generic FDOS at the end of the listing is used by many. Some require that the B register be saved, others need to know the status of an operation. Wherever possible we install these generic vectors so that we might use relative jumps later, and avoid the duplication of code for functions which may be redundant. Again, many of the Index procedure's support routines are highly generic, but were included within the procedure as it was the procedure which called them first.

Editor's Note: The NEW-DOS disk series has grown. We can supply the original NEW-DOS disk with the first version of the source code and the Crowe assembler for $10, or the 3 disk HERMIT DOS system with utilities, 500 KBytes of manual text, and source code with assembler for $30. Readers who have already purchased the single NEW-DOS disk can obtain the three disk HERMIT DOS set for only $20. NEW-DOS is a generic skeleton suitable for other systems, while HERMIT DOS is a complete system intended for the AMPRO Little Board Computer.

■■■

Editor

(Continued from page 37)

plied Ztime-I Calendar/Clock boards for both Hilton's and TCJ's Little Boards, and you'll hear a lot more about them in future issues. Hilton is working on the software for our BBS, and has incorporated time and date stamping using the Ztime-I. The only bad thing about using it for the BBS is that it won't be available for other uses, so I'll have to buy another one. I'm impressed with it, and especially like its alarm interrupt output which can power up the computer at a specified time for measurements or control. KCT has given us permission to place their software on our BBS, and we expect our readers to add to the collection. The Z-Time-I can be used with almost any Z-80 computer, and we'd like to hear about any applications using it.

Some Thoughts On Books

When taking a first look at a subject such as Turbo Pascal, C language, or the CP/M operating system, it is relatively easy to find books with the basic information. That's fine for the beginning, but after you've gone thru them and want to do something non-trivial you find that the nitty-gritty details are missing. It seems that the author of every book has to cover all the standard classifications such as bytes, and hexadecimal, and data types, etc., and that there isn't any room left for the important details. Or else it's that the publisher wants to target the book at the broadest possible market, and doesn't want any restrictive specific details.

We at TCJ are well aware of the book situation from our efforts to obtain information, and have formed a separate book publishing division (Rockland Publishing) to produce books with the information which we can't find in the usual books.

Our goal is to provide advanced—but practical—application orientated information for people who have access to the raw beginner level material. Our first book is "Turbo Pascal—Advanced Applications" which we are typesetting now for delivery to the printer in June. We have material from many well known authors, each writing about their area of expertise. Some of the topics are Optimization Techniques including Algorithm/Data Structure Optimization, Code Optimization, and Procedure and Function Optimization, etc.; Interrupts From Turbo Pascal including Hardware Interrupts, Software Interrupts, Serial I/O Interrupts, Turbo Procedures for Interrupts, etc.; Exploiting Command Line Arguments including Turbo's Command Line Functions, ParamCount and ParamStr, Extracting Command Line Parameters, Developing a Command Line Parser, etc. These are only three of the 16 chapters, there is lots more.

The book was originally planned for 160 pages to sell at $14.95, (all prices are for delivery in the U.S.) but we uncovered so much good material that it now looks like about 208 8½ 11 pages and will have to sell for $16.95 plus $1.50 postage and shipping. All the programs listings are included in the book (which is one of the things which makes it so big), and will also be available on disk for $10.00 or on our BBS for downloading at no charge. We will also establish an area for the book on the board for discussion, questions, etc. In this issue we've advertised a Pre-Publication sale at $12.95 plus $1.00 postage based on the expected price of $14.95. We'll honor this sale price even though the book will be bigger and sell for a higher price than we anticipated, but the sale price ends July 15, after that it's $16.95 plus shipping.

Our goal is to provide technical information for tightly targeted specific niches, and we have several more books in mind, but we would appreciate your suggestions. Let us know what your information needs are which are not being met, or contact us if you are interested in writing for this book market.

PROM Blasting

Now that we're learning more about operating systems in Hilton's NEW-DOS series, we want to add the abilities to download, patch, and reprogram PROMs. One of my goals for dedicated applications is a diskless system with the operating system/program in PROM, and data storage in battery powered CMOS RAM, EEPROM, or EPROM. For critical applications I'm worried about batteries, and I'll probably put the data directly in EPROMs. Diskless systems (combined with AI) should interest Hilton for his work with the disabled where their handicap makes handling the disks awkward, and PROM based units are also necessary for embedded computers.

We've been looking for a versatile, moderately priced, EPROM programmer which works thru the RS-232 interface so that it can be used with almost any computer. We've selected the Periphco PROGRAMMER/4+ which will handle 2716, 2732, 2764, and 27128 EPROMs, and are working on projects. We would like to contact an author for a tutorial series on programming for PROM based operation. ■

Advertiser's Index

THE COMPUTER CORNER

A Column by Bill Kibler

Well summer is here in California and so start the swap meets. Our club had its first one of the season and it was a big surprise. The last one we held in September of last year had a rather poor showing, but this one started strong at 7 a.m. and slacked off by noon. The surprise was the number of eager people looking for bargains. There were easily twice as many people as the last one, but the buying power was still the same. Anything under a hundred dollars went fast, good system, even for as little as $300 didn't move at all. We had lots of clone sellers and component dealers. They did some sales, but the cheap items were the best sellers. It appears that people are still looking for bargains, but there are too many good systems on the market.

My projects have been stumbling along as slow as ever. It seems there have been some days lately where no matter what system I try and work on it doesn't work. When I have days like that, as I am sure most of us have, forgetting completely about computers does wonders. I have been thinking less and less about 68K systems. My troubles with the Godbout CPU 68K has not helped, as it is still not working, and is begining to look like it may never work. Now this could be the system I have it in and not the card, but I have found some things which I don't like. Their use of PALS is not the only thing that is bad but then I have never liked PALS period. The early PALS were programed improperly and thus they had a high failure rate.

I now have a copy of the original Motorola Z80 to 68000 source code converter program. This is a pascal program that takes Z80 mnemonics and converts them over to 68000 mnemonics. The most interesting thing about the program was the introduction, where they compare Z80 speed to 68000 speed. Taking a Z80 basic interpreter and converting it to 68000 code produced some interesting results. The code assembled 2.6 times longer than in Z80 (uses 2.6 times more memory for operation). In speed tests the basic interpreter on a 8 MHZ 68000 ran at the same speed as a 4MHZ Z80. My own experience has shown that most code intructions are at least two to four times as long to express as they would be in Z80. I still like I/O mapped systems over memory map. Some Z80 code instructions still appear to be considerably faster than any other processor instructions.

The major diffference between the Z80 and the 68000, and which is the 68000's strongest selling point, is the continuously addressable memory space. Even with the 64180, Hitachi's 512k Z80, the address space is not really continous like the 68K. Intel's 8088/86 also does not have freedom of handling memory like the 68K. For operations like advanced graphic, the continous addressable memory is absolutely necessary. What has got me rethinking things is just how important is all this speed and memory for the average user.

I am sitting right now at my 64K Z80 based Superbrain computer. This unit is the most pleasurable word procesor I have ever used. It is fast and reliable. I know it has some faults, but when writing, none of them become appearant. This unit is memory mapped and is fast, which is my favorite point. Compare this unit to a 8 MHZ PC clone and this unit will be faster by a long shot. The reason is much like the difference with the Z80 and 68K. For PC's and most of the graphic type of memory mapped systems, several memory writes are needed to get one character on the screen. A better way of describing these system is to say they are bit mapped not character mapped. My Superbrain is character mapped, it stores one ASCII character per memory location. A bit mapped system on the other hand uses nine memory locations per character on the screen (average of nine). This means my Z80 will make only one write to memory for each character, while a 8088 or 68000 will need nine (or more). Using speed of write as a very simple comparison of speed, the average 4.77MHZ PC is about the same speed as a 1/2 MHZ Z80 system. Let me add also, that for straight text, a non-bit mapped system is by far easier on the eyes, than a color graphics system.

Speaking of great memory mapped systems, I have two Xerox boards running. My packet system (a 820-I) is not working, or at least the interface part is not working. The article for it was full of errors, and the latest copy from a second source is better but I am not sure if it is more accurate. The unit uses an AMD 7910 WORLD CHIP MODEM. This is a 300/1200 baud modem that supports all the US and foreign standards in one 28 pin device. I have two of these chips and both are experiencing the same strange conditions. What I have found out is their need for reset and DTR strobing. The chip should be reset with DTR high and then have DTR go low. This guarantee's the internal software gets started properly. This chip is all digital, which is why they can support all the interfaces, just micro code subroutine changes. If I find out some interesting facts about this chip I'll surely pass it on, like my next topic.

I have been approached by fellow club members about making a disk copying machine. Now several companies make them already, and we felt (or at least I did), that one could be made at very low cost from simple parts (8 or 10 chips). The idea was to do full track copies, using WD 179X chips. Having done lot's of full track snooping, and having what I thought was a good idea of how the chip works, I said "no problem, give me a couple of weeks". Well as I am sure you have guessed, it is not as simple as it may seem. After a week of using my system at work to check the software routines, I was still unable to get the full track write to work. Everything worked fine except for the full track write, which was getting me concerned that I had not read all the find print. After reading the literature for the forth or fifth time, I decided their statement about it checking for F5 through FE character in write mode, said it doesn't do write, just formats. After a quick call to WD, sure enough, the chip doesn't do full track writes, just full track (any format within reason) formats. Checking all other disk controller chips produced the same results. Now this doesn't mean that one of the chips could not be used in a special test mode, but so far as I can tell, all these chips do formatting only, and will not allow pre-filled CRC data or sectors.

Since my work on this concept, a $700 disk copier has come on the market. Our disk librarian has ordered one for his work, and I should have more to say about how they do it later. I had actually considered finding some old Z80 based disk controller circuit and making a copy of it. This would give me full control over the system. Another problem I was having with the system was the need to wait for the WD179X to clear internal

(Continued on page 45)