ISSN # 0748-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 23 March—April, 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
Neil Bungard
C.Thomas Hilton
Donald Howes
Jerry Houston
Bill Kibler
Rick Lehrbaum

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

The subscription rate is $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.

To indicate a change of address, please send your old label and new address.

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

Battle of the Bits

We are being asked to choose between 8-bit systems (usually Apple® or CP/M®) and 16-bit systems (usually IBM PC® compatible). The matter is not really choosing between them, but rather selecting the best tool for the task.

We at TCJ are not ignoring or fighting the 16-bit MSDOS® machines—they are often the best choice because of the software available. When a non-programmer who has an older CP/M machine asked my advice on buying accounting and inventory management software for his system, I advised that it would be better to buy an IBM clone plus the software than it would be to buy the software for his CP/M machine. He needs software, and the best software with the best support at the best price is available for the MSDOS machines. On the other hand, I see no reason to replace my Z-80 systems for writing, editing, typesetting or managing our subscriber data base while the existing software is doing a satisfactory job. I probably will get an AMPRO 186 system (80186 CPU) in order to run some of the great software utilities for Turbo Pascal® , C programming, and to be able to use the 8087 coprocessor—but I will continue to use my three Z-80 ZCPR3 systems too! I'll get the 16-bit system not for the system, but rather because the programs I need are written for it.

The reason that I am choosing the AMPRO 186 instead of an IBM clone is that I don't need hi res graphics or sound, but I do need a well built high quality system with lots of I/O capability. The AMPRO uses a serial ASCII terminal which is much faster than bit-mapped graphics for character oriented writing and editing; plus it has extra ports for RS-232 and a parallel printer, a connector for two more floppy drives, and a powerful SCSI/PLUS® interface port.

Operating Systems

One of the reasons that Tom Hilton and I will continue to support the 8-bit CP/M-like systems is that the operating system is a program on disk and we can modify it, or even write our own operating system. Many people are satisfied using the operating system supplied by someone else, but others have the need to be able to change the system to "Do it my way." As Tom says "If you cannot make the system do as you want it to, there is no purpose in having a computer." I have some odd-ball interests which are entirely different than what the system designers had in mind, and I need to be able to write a system to do what I want, the way I want to do it.

In fact, I have a number of applications which need different operating systems. Where did we ever get the idea that we should only have one version of an operating system for all of our uses? One of the greatest faults with CP/M is that it was designed with the idea that an OEM or system implementor would configure the BIOS once, and the user would use it for ever more—or else they would hire a system implementor to reconfigure it and then use only the revised system.

> **"If you cannot make the system do as you want it to, there is no purpose in having a computer."**

The idea of one unchangeable configuration does not meet today's needs when we may want to choose between three or four different output devices. Perhaps we want to send a rough draft to a parallel port for the dot matrix printer, send the final version to a serial port for the daisy wheel printer, send a copy to another serial port for the modem for transmission to another office, and then send the next document to a lazer printer. CP/M is not designed to handle this, perhaps one of our MS DOS experts can tell us how it can be done with that system. It CAN be done with a Z-80 system working thru the SCSI interface or an S-100 system with extra I/O cards. The S-100 systems are not popular now, but they do offer almost unlimited I/O which is the weakest area in the IBM PC compatibles. The PS's are intended for the normal office environment where they serve one printer and a modem. While they do have an open bus with slots, I don't think that they are practical where you want three or four parallel I/O ports (full bidirectional latched input and output, not just output as implemented in most parallel printer ports) plus six or eight series ports (some of which will need 20ma current loop drivers) I feel that the only suitable systems for these applications are S-100 with a 22 slot mother board, SCSI Plus, or one of the industrial buses.

Yes, I realize that programs I write to

run on my modified operating systems will not be portable because they won't run on anyone else's system. I don't care! I'm writing what I need to do what I need done, and a non-portable package which does my job effectively is a lot better than a portable program which does the job poorly. The operating system is not sacred and I intend to mess with it. If what I develop does the job so much better than anything else that someone else wants to use it, they can buy the same hardware and use my package which includes the operating system. One nice thing about using your own operating system is that you can supply it to others without paying a license fee, and it can be designed to be implemented in ROM for dedicated controllers. The operating system and the application software are both programs, and as they are integrated to work closely together the differences between the system and the program become less distinct and they can be merged—if the system source code is in the public domain. You can strip out the non-essential portions (some ROMed applications won't have drives), combine it with the program, and end up with one combined file. Would you call that an operating program?

Why NEW-DOS?

Some people have questioned whether there is a need for another operating system, and we want to explain our reasons for publishing the NEW-DOS series. One of the primary reasons for the NEW-DOS is educational. Our goal is to show you how an operating system works, how it is written, and enable you to write or modify an operating system. I feel that at the current time, operating systems should be written in assembler, and Tom's step-by-step program development with source code and a public domain assembler will demonstrate that assembler is not as difficult as commonly reported for this application (I wouldn't want to write floating point math, array, or string handling routines in assembler unless they were intended for large volumn distribution). Therefore part of our eduational goal is to encourage you to learn how to use assembly language.

Another reason is to provide a free CP/M compatible system which can be included on our (or your) distribution disks. It is important that source code with step-by-step instructions are available so that you can fully understand how to modify the system. For nuts like me another reason is to provide a small system nucleus which we can change, expand, condense, or whatever for specific non-standard applications.

We chose a CP/M like system because its features are well documented, and it represents a minimum system which is useful for a teaching tool, yet simple enough to understand. I doubt if many would want to follow a series on writing a UNIX® equivalent. The knowledge gained from this project will enable you to modify other systems, such as ZCPR3 or MS DOS® . In fact, I'd like to challenge MS DOS advocates to author a series on writing a public domain equivalent, or at least modifying it (including modifying the ROM's).

I'll probably use ZCPR3 on the AMPRO for the development system because of the power of the system and the large number of utilities they supply, and their upcoming multitaskinr and banked/partitioned versions plus "ZCPR3: The Libraries" will facilitate the programming which will mostly be done in assembly. Licenses from Echelon are very inexpensive, they supply the source code for almost everything, they constantly expand and update their products, offer outstanding support, and have released about 2 million bytes of code which is on over fifty Z-Nodes and has been given to SIG .

Source Code Required

The availability of source code from Echelon brings up another very important point. I don't intend to use programs unless I have the source code. There are a few exceptions—I'm currently using WordStar® and Condor3® for which the source code is not available, but I'll have to disasseble portions of their code to make some very minor (but very necessary) changes which would be easy if I had the source code. There are other programs such as Turbo Pascal® , C compilers, DSD 80® , and the SLR Systems assembler and disassembler for which I wouldn't expect the code. But in these cases there are other programs which could be used with my source files. I don't want to trust my accounting, database, or other files to copy protected software without the source code. Some users are reporting problems with copy protected programs because the companies have gone out of business and crashed disks can no longer be replaced. My goal is to use software where the source code is available whenever possible, I won't even consider using copy protected programs.

The Next Hacker's Bus?

While programmers can write software for their own use with little regard to what the rest of the industry is doing, Hardware Hackers have to design around what is available in the industry. We read the professional engineering magazines (Electronic Design, EDN, Electronic Engineering Times, etc) to keep up with current events and future trends, and have been paying particular attention to the buses being used for industrial applications. The STD bus is one of the older buses still in active use, and has been upgraded to include MS-DOS® compatible operation with the 8088. Multibus® (Intel) has been redesigned as MultibusII® , and both versions are active. The bus with the greatest activity is the VMEbus® , and the most frequently used CPU is the 68020. We can't always afford the latest state of the art products for our personal projects, but quite often hardwarefrom industrial

(Continued on page 6)

The C Column

Flow Control and Program Structure

By Donald Howes

The use of the C programming language is spreading rapidly through the microcomputer community. Once thought to be restricted to the area of "systems" programming, it has spread to become a dominant language in the development of applications programs as well. Uses include such diverse areas as word/text processing, statistical analysis, graphic display and artificial intelligence applications.

Hi, my name is Don Howes, and I'd like to welcome you to the kick-off of The C Column. In this column, I'll be looking at some of the tricks and giving you some tips for writing fast, efficient C code, which you can apply in your own programming. I'll also try to pass on to you any news that I gather about changes in popular versions of C compilers, as well as news about the ANSI C Standards committee (which has already had an impact on new compiler releases).

But about the C language, just why has it become so popular? Here are a couple of reasons drawn from my own previous programming background in FORTRAN and Basic, which may shed some light for people who have never used the language, or have been put off by the cryptic appearing syntax.

Modern Control Flow

This may not seem like all that much, but it is hard to explain just how restrictive it is to be confined to **IF-THEN-ELSE** and **DO**-loop (or **FOR-NEXT** loop) constructs, when other types have become available (we won't talk about **GOTO**'s, since it is my biased opinion that their use leads to spaghetti code, and that mentioning **GOTO** and control flow in the same sentence is a contradiction in terms). The C language has **if-else** ("then" is not used in the language) and **do while** (equivalent to **DO** or **FOR-NEXT** constructs), but in addition includes **while**, **for** and **switch** statements. I have demonstrated some of these in a short (rather useless) program (Figure 1).

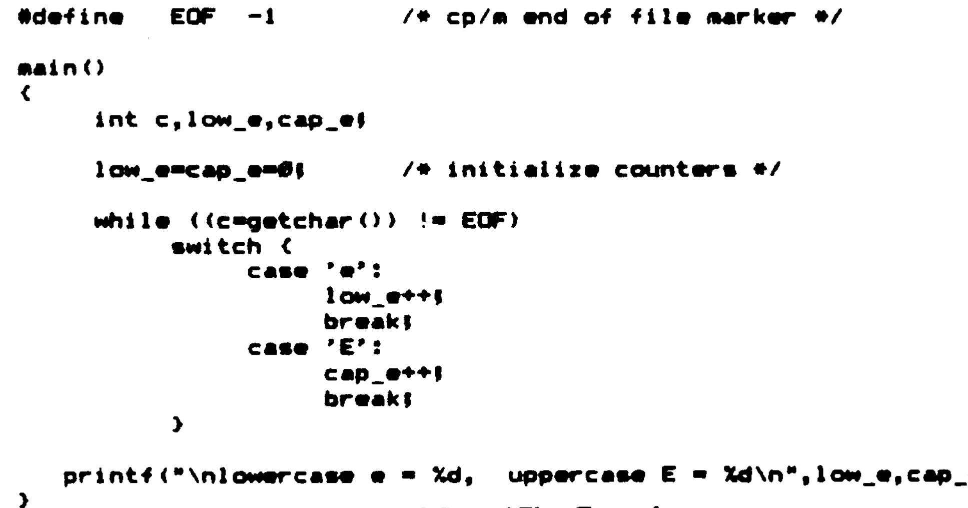
What the program does is to read character input from the keyboard, and keeps a running count of lower and uppercase e's, which are reported after the user has indicated the end of input by typing CNTRL-Z (the CP/M end of file marker). The program could be improved by adding the capability of checking for entry of a backspace character (which could remove one of the previously counted e's), but that would add some complications to what is only an example program. In the program, I have used both the **while** and **switch** constructs to do the counting. The **while** statement performs a loop until the test condition present at the beginning of the loop is satisfied, at which point the loop is terminated. The nice thing about a **while** loop is that the test condition is checked prior to the beginning of each loop (unlike a **DO** loop or a **FOR-NEXT** loop, which tests at the bottom of each loop). This means that the loop is not entered if the test condition is met immediately (in this case, if the first character entered is EOF).

Internal to the **while** loop is the **switch** statement. What a **switch** does is take the value of a variable (in this case, the variable 'c') and compares that value to the cases listed. Here I am checking for upper and lower case e's, all other values for the variable (all other ASCII characters) will fall through the switch. When one of the cases is found to be true, the code following the colon is executed. The **break** statement is needed to prevent the program from falling through to the next case and executing the code (the 'case' is just a label, not an executable statement). **Break** causes an immediate exit from the **switch**.

Program Structure

This means more than just what the code looks like, but goes to the heart of program development using the C language. While it is, of course, possible to write structured (or for that matter, unstructured) code in any language, some languages lend themselves more easily to the task. While there has been a lot of talk in the computer magazines, it does bear repeating that structured code is easier to maintain and extend than an equivalent unstructured program. This is due to the logic of the programming task being explicitly detailed in the layout of the program.

The C language makes structured programming easier through its use of functions. In this language, everything is a function. If you look again at Figure 1, you will see that even the declaration of 'main' shows it to be a function. This is shown by



```
#define   EOF  -1         /* cp/m end of file marker */

main()
{
     int c,low_e,cap_e;

     low_e=cap_e=0;       /* initialize counters */

     while ((c=getchar()) != EOF)
          switch {
               case 'e':
                    low_e++;
                    break;
               case 'E':
                    cap_e++;
                    break;
          }

   printf("\nlowercase e = %d,  uppercase E = %d\n",low_e,cap_e);
}
```

Figure 1: C Control Flow Example.

the use of parentheses [main()]. In this case, there are no arguments being passed to main(), but there can be. The two possible parameters are "argc" and "argv", which are the number of parameters and an array of pointers to characters respectively This is shown in the following code fragment. Suppose you have written a simple copy program (which we will reasonably call copy) for copying an old file into a new file. What you have to pass to the copy program, therefore, is the names of the source and destination files. This can be done with the command

```
copy somefile.txt newfile.bak
```

In the code for the program copy the main() function would be declared as follows:

```
main(argc,argv)
int argc;
char ,argv[];

        .
        .
code to do copy
        .
        .
```

Within the code, you would use "argc" to check on the proper number of input parameters (that there are two filenames) and "argv" would hold the actual filenames of the input and output files. For the program to work properly, there would have to be a fair proportion of error checking code to make sure you weren't trying to copy from the source program to itself, copy into a file that already existed (unless you wanted to), etc. Writing that type of code is a column in itself, but not right now (as an aside, making sure that you have protected users from themselves is generally known as bulletproofing. I'm sure the reference is to not shooting yourself in the foot. Remember Murphy's Law).

The whole idea in C programming is to hide the low level functions from the programmer through the use of user created functions (in this case, the user is yourself, the C programmer). It has been said that the best type of C code has a main() function containing nothing but function calls, but I think this is a little excessive. Generally, the main() function should only contain code that is specific to the application being written. Calls to functions should be generalized so that the same function can be used in a number of different applications and for a number of different data types, if that is necessary (an example of this would be a generic sort routine which could be used equally well on character, integer and floating point data). The only thing which the programmer should need to know about a function is what type of input is required and what type of output can be expected. Everything else is a black box (it really could be magic, for all you want to know).

Where To Go From Here?

I think I've run on enough for one session, which is too bad, since there were other things that I wanted to cover, maybe next time. If I've managed to prick your interest in this language, then I'm happy, but I don't want to leave you stranded. So, here are some books that I've found useful when I was starting out and still use for reference and as sources of inspiration.

1. Brian Kernighan and Dennis Ritchie, 1978, The C Programming Language (this is the bible of C programming [almost always referred to by its initials "K&R"] and is what compiler companies mean when they say a "full K&R implementation", this book is a must).

2. Jack Purdum, 1983, C Programming Guide (this is the first book I bought on C programming, it's now into a second edition. I found it to be very clear, with lots of examples, I still refer to it).

3. Samuel Harbison and Guy Steele, Jr., 1984, C A Reference Manual (this is a reference manual with a vengeance, not light reading but covers everything I've wanted to find. Clears up or at least makes explicit some of the ambiguities present in K&R).

4. Brian Kernighan and P.J. Plauger, 1976, Software Tools (while not about C programming per se, if you want to learn more about the mechanics of structured programming, this is the book for you).

In addition to these books, let me recommend the C Users' Group to you. I am a firm believer in the idea that a programmer only becomes better by looking at and massaging other peoples code. The C Users' Group is the main organization for the dissemination of public domain C code, and you will have plenty of opportunity to learn from looking (and if you don't like the way it was done, have fun changing things, I have). Their address is

C Users' Group
Box 97
McPherson, Kansas 67460
(316) 241-1065

Membership fees are $15.00 domestic (Canada and Mexico) and $25.00 overseas. They publish a newsletter about four times a year and have an extensive library of C code available for little more than the cost of the disk and mailing. Give them a call, you'll be glad you did.

Next time in this column, we'll get down to brass tacks and look at some working, useful code. I'll be talking about filters, something which most people don't know how to write. We'll look at some simple filters to remove and replace tab characters and a handy little Wordstar to ASCII conversion filter. Remember that this column is for you, so if you have any comments, suggestions for future topics or questions you would like me to address, drop me a line C/O The Computer Journal. See you next time. ■

The Z Column

Getting Started With Directories and User Areas

By Art Carlson

ZCPR3 has attracted a lot of attention, and people are asking "What is It?", "What Will It Do For Me?", and "How Do I Use It?" I asked these same questions before I started using ZCPR3, and the purpose of this column will be to help users learn how to USE an implemented ZCPR3 system.

As Tom Hilton stated in one of his articles last issue "I never know where to begin, and where ever I begin I should have covered something else first." In presenting ZCPR3, I also have the problem of deciding where to begin. I feel that one of the reasons that I delayed using ZCPR3 is that the manuals and literature I saw were all aimed at the experienced programmer with page after page of highly detailed technical information on using ALIAS, VMENU, and other advanced features which I didn't even understand. The literature described what the advanced implementor could do with ZCPR3, but it didn't tell the average user how to get started. In this column, I intend to cover ZCPR3 from an average user's viewpoint, starting with the simplest steps (just as I recently started), and slowly advancing to the more involved features. While the user will be the main focus of this column, TCJ won't neglect the advanced programmer whose interests will be covered in other articles.

One of the most confusing aspects of ZCPR3 is that it is not a single, well-defined, program, but rather a large assortment of parts from which you can choose the features you need for your implementation. This provides a lot of flexibility, and I intend to assemble several systems, each tailored for a specific use. The advantage of our disk based operating system is that we are not forced to use one system for everything, but can have different operating systems for different applications.

But, before we learn to configure our own system, we have to understand what ZCPR3 is all about! When I received the AMPRO 122 Bookshelf Computer it came with ZCPR3 installed, and all I had to do was boot the system disk to be up and running, so getting started with ZCPR3 was simple. Working with it is also very simple, and very rewarding.

Getting Started With ZCPR3

Since this column is for the beginning user who wants to know how to use ZCPR3, I will assume that you have it up and running. You can either get an installed version (such as AMPRO), get someone to help you install it, install it yourself, or get Echelon's self-installing version. Later we'll talk about installing different features, but right now we'll learn how to use its basic features, and how it differs from CP/M® . Our examples will be from the AMPRO disk A60101, plus information from "ZCPR3 The Manual" by Richard Con (this book is available from Echelon and is essential for understanding ZCPR3.) The AMPRO implementation comes up with a MENU installed by SHELL, but that's not where we are going to start. You can either use the MENU for now, or if you're familiar with CP/M, just enter a control C to exit to ZCPR3 without the menu feature.

Directories and USER Areas

One of the first advantages of ZCPR3 that you will encounter is that you can access programs and display the directories from other drives and user areas. I feel that this feature alone is enough to justify replacing CP/M with ZCPR3.

Many CP/M users never use the USER area feature because of the very poor implementation under CP/M 2.2. For those who aren't familiar with it, I'll start by describing the CP/M implementation so that you can understand ZCPR3's advantages. Both CP/M and ZCPR3 maintain only one directory on the disk, and the files from the different user areas are all combined in this one directory with the user area indicated in the disk directory (byte 0 of the disk directory entry for the adventurous who want to do some snooping).

With CP/M 2.2, you can only access files or read the directories for the current user area. When you boot the system, you start out in user area 0 (zero). In order to access a different user area, say user area four, you can enter "USER 4" followed by a return. Now you can read the user area four directories of any drives on the system, and you can use the files in user area four, **but you can't access files or directories in other user areas.** You have to use PIP.COM to transfer the files you want to use into the user area, but you can't transfer files into the user area unless PIP.COM is already there! How do you put PIP into the user area in the first place? You have to log into an area where PIP is present, use DDT to load PIP.COM, write down the decimal number of pages as reported by DDT, enter G0 to return to CP/M, enter the the desired user area (such as USER 4), then SAVE XX PIP.COM where the XX is the number of pages to be saved expressed as Hexadecimal (you have to convert it from the decimal number reported by DDT). Now you can use PIP to copy the other files you need into the current user area. You still can't use files in other user areas without first copying them into the current area. And if you want to use a file (such as D.COM) in all user areas you have to transfer a copy of it into every user area, after first putting PIP into every user area. This means that with 15 user areas you would have 15 copies of PIP.COM and 15 copies of D.COM on the disk. It's little wonder that few people used the user areas as implemented by DRI in CP/M 2.2. A number of utilities have been developed to partially alleviate this problem, but few of them approach the power of the ZCPR3 system.

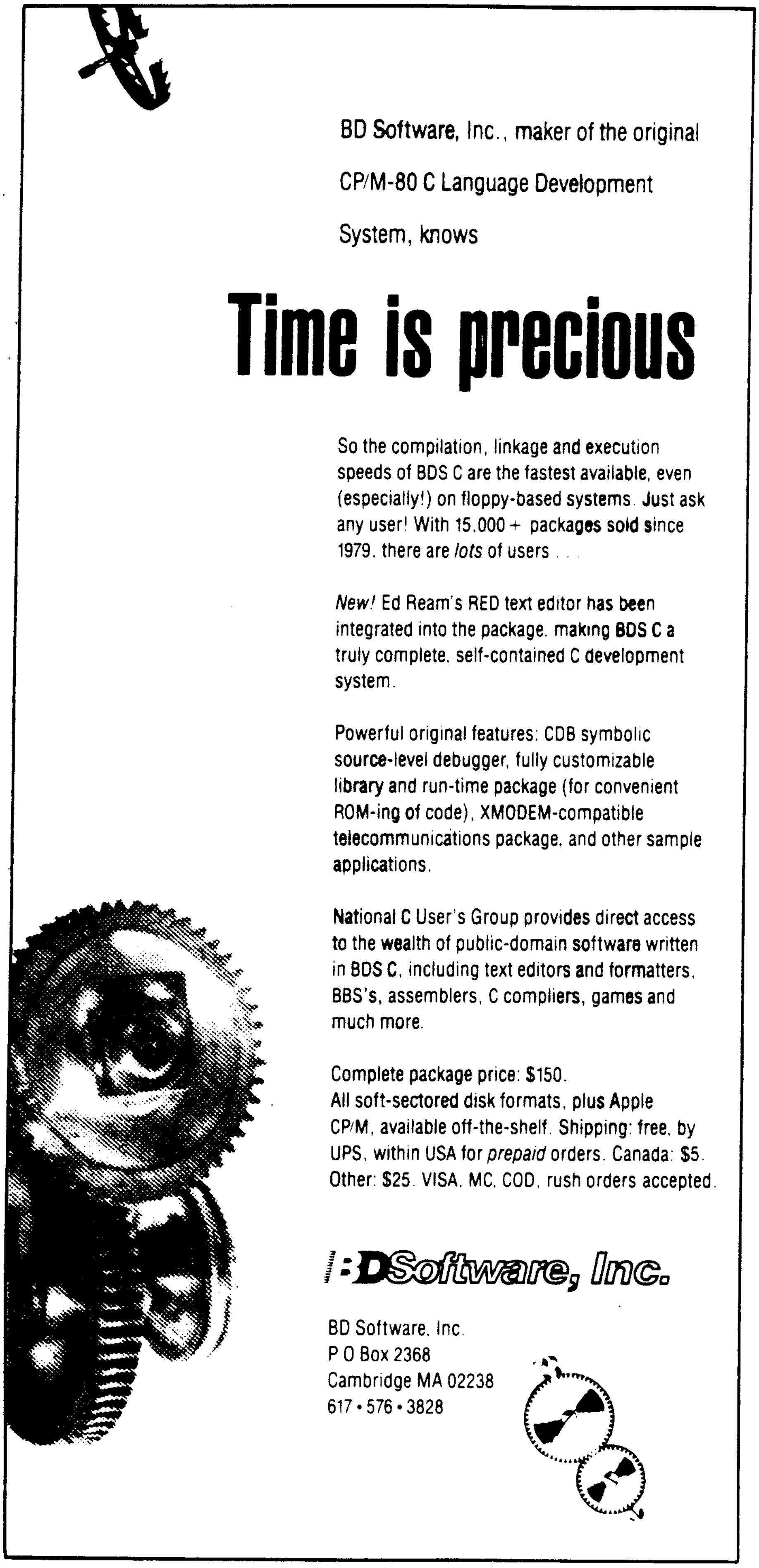
When you boot ZCPR3, the prompt (A0>) tells you which drive and user area you are logged on. If you want to change to user four on drive B you just enter B4: and a return. ZCPR3 incorporates a PATH routine which will search various drive and user areas for a particular file, so if you are in B4 and want to use D.COM which is in A0 you enter "D" (from now on the following return will be assumed) and ZCPR3 will find D.COM and run it without making any extra copies!

You can enter the command "PATH" to display the current searching sequence which AMPRO supplies as:

(1) Current drive, current user.
(2) Current drive, user 0.
(3) Current drive, user 15.
(4) Drive A, current user.
(5) Drive A, user 0.
(6) Drive A, user 15.

The PATH command can also be used with arguments to change the searching sequence, and named directories can be used. With named directories you could name A15 as TEXT and then call up a directory of all your text files in that directory with the command "DIR TEXT:"

(Continued on page 43)



Editor

(Continued from page 2)

overruns and revisions can be economical and powerful additions for our hardware projects.

For hardware hacking on measurement, control, robotics, and other I/O intensive projects we need hardware designed for these applications instead of hardware designed for personal or office computing! There used to be a lot of activity with the S-100 system, but there is very little done with S-100 now. The STD bus is very attractive because it has been in use for a long time and there is a lot of surplus hardware available. My other choice would be the VMEbus because of the 68XXX robotics and control boards being announced every day. The problem with VMEbus is that the products are new and high priced for industrial applications which makes them too expensive for my use.

Do You Want Bus Information?

I am very interested in what's being offered for the new buses, including boards, programming techniques, operating systems, peripherals, etc., but I need to know if I'm boring you. I feel that the hardware hackers computers of the future will come from these developments. The PC bus is not very useful for my needs (it wasn't designed for it), S-100 is stagnant, and we need something on which to concentrate. The SCSI bus has a lot of possibilities, but we'll have to see if industry uses it for applications in addition to interfacing hard disk drives. Perhaps the hardware and software hackers will have to get the ball rolling on this. The question is, "**Do you want TCJ to publish information on current bus developments?**" Write and let me know what you want (see info on RBBS below), or I may just keep this info in my own file.

The TCJ RBBS is Coming

We have just received the good news that the phone company has added multiplexers in order to provide the additional line we need for our RBBS. Now we are looking for the equipment (and the time) to get a board up and running. Tom Hilton is working on the software, and I hope to be able to announce the board in the next issue. Think about what you want on the board in addition to TCJ program listings and messages—we don't intend to offer all the programs which are already on many other boards. Ours should be unique and special. ■

The SCSI Interface
Introduction to SCSI
By Rick Lehrbaum

Introduction

In this part of The Computer Journal's series on the Small Computer System Interface (SCSI), we will take an introductory look at the features and functions of SCSI, both from a software and hardware perspective. This article will serve as a sort of "SCSI Primer." We'll see why and how SCSI is used, what it offers to the computer system (and to the user), and how it is structured architecturally.

The next part (Part 3) of this series will go into greater technical detail on the SCSI software and bus interface protocols, while Part 4 will include some simple SCSI design examples. After we've established the basics in Parts 1 through 4 of this series, we'll explore some new SCSI developments (including SCSI/PLUS, Super SCSI, and Enhanced SCSI!), look at some specific SCSI products (disk, tape, optical, ...), and discuss some unique SCSI applications (scanners, emulators, ...).

Why SCSI?

As we saw in Part 1 (TCJ Issue #22, page 25), SCSI grew out of the Shugart Associates System Interface ("SASI") as a matter of practicality. Shugart was tired of waiting 18 to 24 months for sales to ramp up after the introduction of each new disk drive technology. So Shugart invented SASI to simplify the addition of new mass storage devices to existing systems. SCSI is simply standardized SASI. The Shugart Associates System Interface became the "Small Computer System Interface" (SCSI) when the American National Standards Committee X3T9.2 took on the task of standardizing SASI.

The purpose of SCSI is to create a high level interface to peripheral devices, so that system designers have an easier task interfacing existing systems to new peripheral devices. This is accomplished in two ways:

(1) The hardware interface is reduced to a simple bidirectional data path with a few simple handshake and control signals. The result is a "buffered" bus which is independent of both the computer and the peripheral device to which it is connected—a sort of common meeting ground for the system and the peripheral.

(2) The software interface is generalized so that a common set of software commands can be used to control peripheral devices within a given device class (e.g. random access devices) independent of their specific features and design.

Ideally, this isolates the host computer from the specific details of the peripheral device. When properly implemented, SCSI allows a variety of devices to be interchangeably connected to a computer system, with no change to system software or hardware interfaces. For example, two SCSI hard disk subsystems might appear identical to the host computer with the exception of the number of megabytes of data they can store, even though the subsystems may contain entirely different drive technologies. In fact, one company will soon introduce an SCSI tape drive that looks like a hard disk drive to the host's software.

SCSI therefore offers the possibility of true "plug-and-play" system integration. The computer system has an SCSI "socket" and a set of software SCSI device drivers and formatters. SCSI even provides commands which allow the host computer to ask connected devices what they are and how big they are. Consequently, peripheral device drivers and formatters can be written to automatically configure themselves to the characteristics of the SCSI devices. For example, a host computer might automatically determine what devices are attached to its SCSI bus on powerup, when the system initially "boots," and configure itself accordingly.

Unfortunately, the plug-and-play promise of SCSI has not yet been fully realized. At this time, system software generally can only deal with a few specific SCSI makes/models, and must either be told by a system installer, or must perform some sort of test to determine, what make and model of SCSI device is connected.

There is, however, no fundamental obstacle to plug-and-play SCSI. In fact, two recent events have contributed to an upsurge in SCSI product compatibility: (1) The ANSC X3T9.2 committee has forwarded the final draft SCSI document for public release and final publication; and (2) SCSI product vendors have begun to cooperatively agree on "common command sets" which create an enhanced level of software uniformity relative to that required by the SCSI specification. We'll cover the first of the common command sets, the Common Command Set for Random Access Devices (e.g. hard disk), in a future article in this series.

An SCSI Architecture Overview

Before we begin on our exploration of SCSI, let's define a few terms.

SCSI was intended to provide an "intelligent" interface between computers and peripheral devices. In many ways, SCSI is like a network. Up to 8 intelligent devices can share a single SCSI bus. Any SCSI device on the SCSI bus can communicate with any other device on the bus. This is termed a "peer-to-peer" architecture.

There are three typical configurations of SCSI. Some implementations involve a single computer and a single peripheral device. Others have one computer and multiple peripherals. Still others envolve multiple computers sharing multiple peripherals.

In SCSI lingo, the computer is officially called an "Initiator" and the peripheral device is called a "Target." The three SCSI configurations, shown in Figure 1, are:

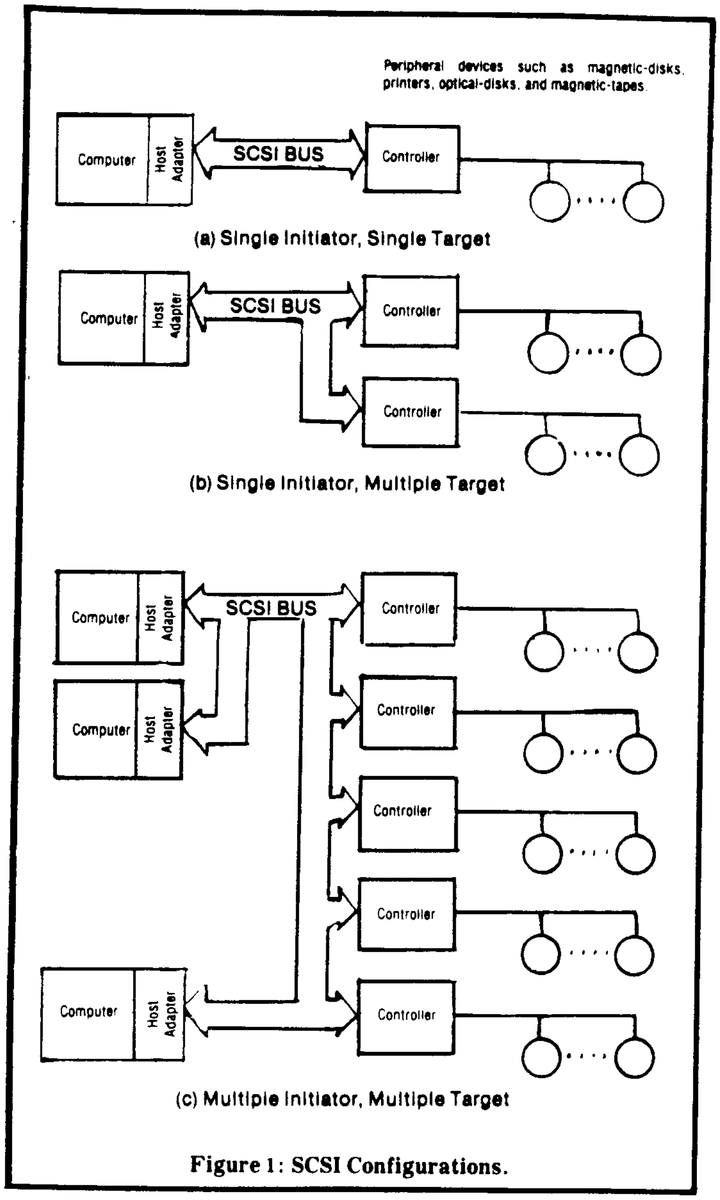
Single Initiator, Single Target
Single Initiator, Multiple Target
Multiple Initiator, Multiple Target

We'll learn more about the differences in these three configurations in the next part of this series. Most systems fall into the Single Initiator, Single Target category, which might be thought of as "simple SCSI."

Breaking up the Controller

What SCSI has done is to "break" up the traditional system into a new set of partitions. (See Figure 2.)

Traditionally, the host computer is usually connected to peripheral devices through a custom device-specific controller. As an example of the traditional system partitioning, consider the typical hard disk drive controller used to interface an S100 system to an ST506/412 type hard disk drive. It contains a


Peripheral devices such as magnetic-disks, printers, optical-disks, and magnetic-tapes.
Computer
Host Adapter
SCSI BUS
Controller

(a) Single Initiator, Single Target

(b) Single Initiator, Multiple Target

(c) Multiple Initiator, Multiple Target

Figure 1: SCSI Configurations.

serializer/de-serializer, a data separator, error correction logic, buffer logic, a system bus interface, and drive interface logic. On the software side, unless the controller card has its own on-board microprocessor, the host system software must oversee every aspect of the controller, including drive control, data control, error recovery, buffer management, formatting, etc.

In an SCSI-based system, the device controller function is divided into two parts One part takes care of the actual physical device (in this example, the hard disk drive). This part is called the SCSI formatter/controller. The other part interfaces the host computer to the SCSI bus, which is how the two parts of this "broken up" controller communicate with each other. This second part is called the Host Adapter, and is often a single IC in microprocessor based host computers. A single Host Adapter can control up to seven SCSI controllers (which can each control many peripheral devices).

The Importance of Intelligence

SCSI not only breaks the traditional controller **hardware** up into two parts, but also divides up the **software**. To do this, each SCSI device controller has its own local intelligence.

The SCSI controller uses an on-board microprocessor to control the SCSI interface. In addition, the SCSI controller's on-board microprocessor contains the firmware required to control and format the peripheral device. For example, the controller's firmware would include such SCSI commands as Format Unit, Read Data, Write Data, Test Unit Ready, etc.

Since the detailed device interface and control algorithms are provided in "canned" routines, accessed through SCSI's high level commands, the expertise and responsibility for the low level device interface rests with the controller manufacturer, rather than with the system's designers and programmers. The system need only properly handle the high level SCSI commands, which are quite similar to operating system functions.

The Results

The peripheral control firmware that resides on the controller provides a tremendous savings in system software design time, expense, and headaches. New or added types of disk drives or other peripheral devices can often be added to a SCSI-based computer system in **days** rather than **months**. Less obvious, but equally important, is an important side benefit: a substantial improvement in both reliability and functionality. After all, the controller manufacturer is an expert in the peripheral technology for which the controller is designed.

The SCSI controller's on-board microprocessor provides a degree of parallel processing which saves the host computer the burden of low level peripheral control and maintenance. In many systems the performance available with SCSI controllers is two to five times that available with traditional device-specific controllers.

SCSI Bus Signals

The SCSI physical bus interface is fairly straight forward. Although we won't be dealing with bus interface details until the next part in this SCSI series, let's look very briefly at the bus signal functions:

DB0-DB7—Eight bi-directional data lines. It is over these lines that the data is transferred between the Initiator and the Target, in either direction. These lines are also used as device selects during the initial selection of a Target by an Initiator.

DBP—Data bus parity. This is an option, and often not used.

BSY—Busy. Indicates bus is in use. This signal is initially activated by an Initiator to gain control of the bus, but later asserted by a Target to indicate that it is active on the bus.

SEL—Select. Used by one device to establish communication with another. Controlled by the Initiator.

C/D—Control/Data. Indicates type of information on data lines. Controlled by the Target.

I/O—Input/Output. Indicates direction of data transfer between devices. From Initiator to Target is called "Output." Controlled by the Target.

MSG—Message. Indicates that the Target desires to send a message to the Initiator while the Target is selected. Controlled by the Target.

REQ—Request. This signal is driven by a Target, once selected by an Initiator, to indicate that it is ready for a byte of data on the data lines.

ACK—Acknowledge. This signal is driven by an Initiator in response to a REQ signal from a Target, once the required data has been placed on, or read from, the data lines.

ATN—Attention. This signal is used by an Initiator to indicate a special condition or status to a selected Target.

RST—Reset. Used by an SCSI device to reset all bus devices.

These signals can either be driven with single-ended or differential line drivers, depending on whether the system chooses to implement SCSI's Single-Ended or Differential Option. We'll look at the differences between these two interface options, as well as the signal timing relationships, next time.

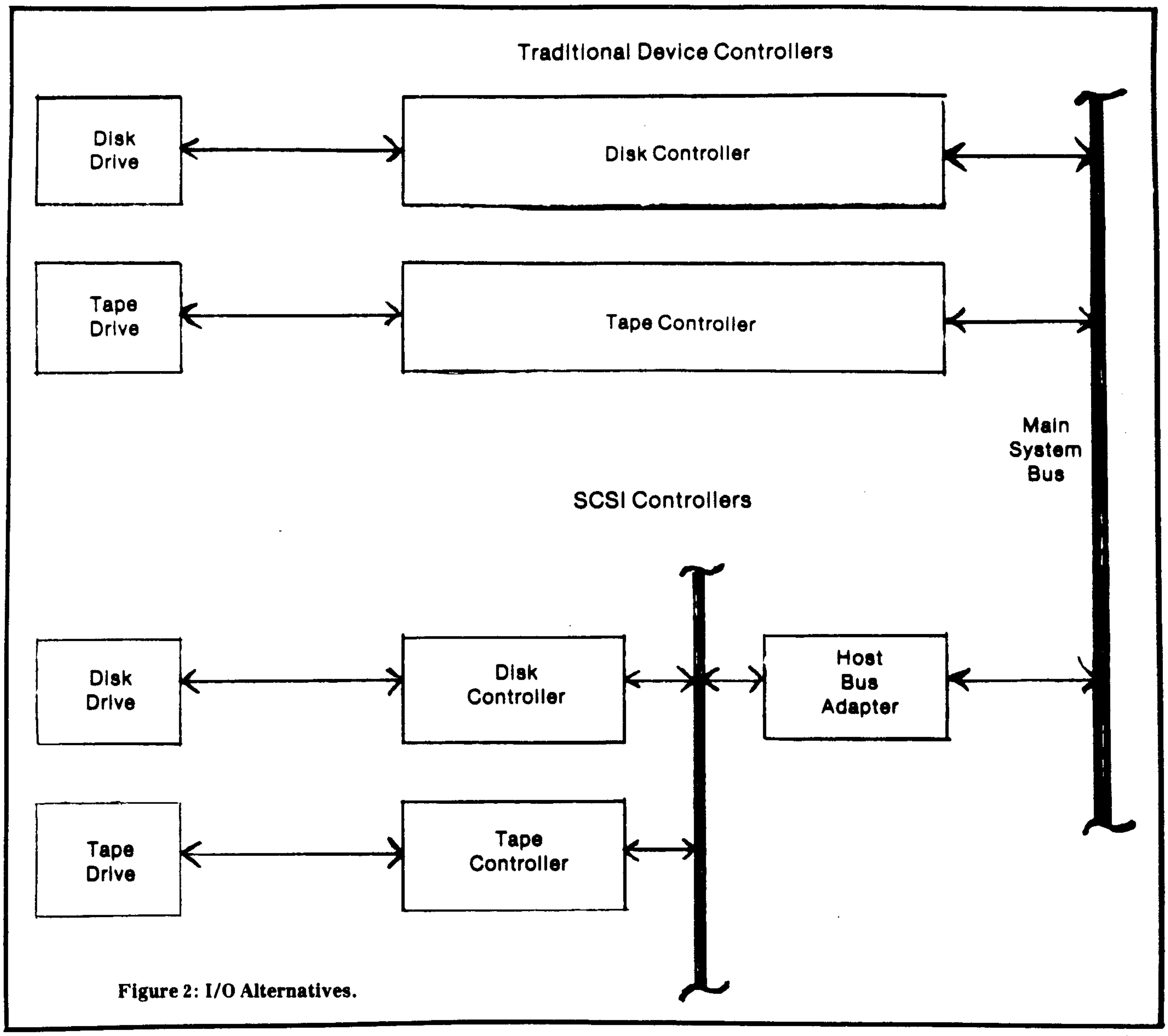


Figure 2: I/O Alternatives.

Adding A SCSI Bus

If you have a bus-based system, you should have no trouble locating an SCSI interface adapter (often called a "Host Adapter") for your system. Host Adapters are available for nearly all bus-based systems, including IBM PC's, XT's, and AT's, and the Apple II, Multibus, VME bus, S100 bus, Q-bus, etc. The new Apple Macintosh PLUS even comes with a standard built-in SCSI port!

If you are planning to design your own SCSI interface, the complexity of hardware and software you will require depends on which SCSI configuration you need to use.

For the single Initiator configuration ("simple SCSI"), SCSI bus signal timing is relatively non-critical, and can be completely controlled by software. In this case, all that is needed are bidirectional digital I/O lines with high current output drivers. In addition, DMA logic is recommended, though not required.

To take full advantage of the peer-to-peer aspects of SCSI (multiple Initiator, Disconnect/Reconnect, etc.), special circuitry must be used to implement such time-critical functions as SCSI bus arbitration. However, now that single-chip SCSI interface IC's are inexpensively available from several sources, a fully implemented SCSI interface is easily achieved.

Single-chip SCSI interface controllers are now available from NCR, OMTI, Western Digital, Fujitsu, and others. These provide the entire SCSI bus interface—including the bus drivers and receivers—in a single IC package.

Making the Bus Run

Once the SCSI bus is in place, SCSI drivers and utilities must be used to format and operate the desired SCSI devices. If you use a ready made SCSI host adapter, those software items are usually provided by the host adapter manufacturer. In this case the only problem is getting the drivers and utilities for the right operating system, and for the desired manufacturer's SCSI controller or device.

If you are creating a custom SCSI interface or host adapter, you will have to generate your own SCSI format and operating software. If this is the case, you'll appreciate the technical details to appear in the next part of this SCSI Interface Series. ■

NEW-DOS
Part 2: The Console Command Processor, Continued

By C. Thomas Hilton

Well gang. I hate to do it to you, but I have had to make some changes in the source code for this series. Time, Tide, And Technology waits for no one, as they say. I have upgraded my system to the new AMPRO® 1B CPU, and AMPRO has changed their BIOS several times since I acquired my system. Additionally, I have been doing a lot of work on the BIOS for my AVS Voice Computing System for the visually impaired. It is just too hard for me to switch back and forth between systems. I would have liked to have screamed, "Stop The Presses," but the changes required are insignificant. Those of you who ordered the support disk for this series will have received the new version of the code, so there is nothing to be concerned about.

```
-----------  --   ----------------------------------------------

-------          ------------------- OLDBIOS  NEWBIOS  OLDSIZE  NEWSIZE
:                                   :  FFFFH    FFFFH     61k      60k
:   BASIC IN/OUT SYSTEM  (BIOS)     :
:--------     ----------------------:  EE00H    EA00H   ___  My System
:                                   :
:  STANDARD CP/M DISK OPERATING SYSTEM :
:-------  ---  --------------------:  E000H    DC00H       My System
:                                   :
:  Console Command Processor  (CCP) :
:--------  ------------------------:  D800H    D400H   ___  My System
:                                   :  D7FFH    D3FFH
:  Transient Program (User Program) :
:  (Dependent Upon Amount of Memory):
:                                   :
:-- --- --- --   ------------------:  0100H   (Start of TPA)
:                                   :
:   BIOS and CP/M Work Area, Page Zero :
:-------  -------------------------:  0000H   (Start Of Memory)

                  SAVE VALUE (Pages)  41      49
                          DDT OFFSET  3180    3580
```

Figure 1: Dual System Memory Map.

Taking a look at our revised memory map you will see where everything is, and there are options you may install to either leave things as they were, or upgrade your code to the current version. There will be no differences in the discussion of the code from version to version.

I have also enlarged the stack by three calls, or 6 bytes, to 32 bytes. Change any source listing showing a 26 byte stack to 32 bytes. This extra space is used only in complex CCP command functions.

Moving On...

Listing One shows the code segments we have already discussed, and as the code stands at this moment. Be sure that you have selected the proper equate for the memory size and BIOS you are using! Please refer to PART ONE of this series for a detailed discussion of what this code is doing, and why.

This "code base" forms the primary CCP loop. In this segment we will discuss the subroutines which are called by this code base. At this point refer to Listing 2. We will begin our discussion with the first subroutine called, beginning at the top of our source code, in Listing 1. The first subroutine call is made after entry at CCP.

```
LISTING 2
--------------------------------------------------------------
;
; reset user number if changed
;
RESETUSR:
TMPUSR  EQU     $+1         ;pointer for in-the-code modification
        LD      A,Ø         ;2nd byte (immediate arg) is tmpusr
        LD      E,A         ;place in e
        JR      SETUSR      ;then go set user
GETUSR: LD      E,ØFFH      ;get current user number
SETUSR: LD      C,2ØH       ;set/get user code
        JR      BDOSJP      ;more space saving
```

To refresh our memories, and having entered at CCP, we expect an Autocommand to be executed. Our first chore is to establish a local stack for use by the CCP. The BIOS has sent us the system drive designator in register "C." Our first concern, after defining a local stack, is to save this system drive information on the stack.

```
;
; start ccp and possibly process default command
;
CCP:    LD      SP,STACK        ;reset stack
        PUSH    BC
```

We then extract the user area, or subdirectory information:

```
        LD      A,C             ;c=user/disk number (see loc 4)
        RRA                     ;extract user number
        RRA
        RRA
        RRA
        AND     ØFH
        LD      E,A             ;set user number
        CALL    SETUSR
```

and load that information, the user area, into register "E." Register "E" is used as we are about to make a BDOS subroutine call, and this information will be the parameter information passed to the DOS system. Looking at Listing 2 we see that SETUSR is but a small portion of another set of subroutines sharing the same segment of code. We are limited by the amount of space the CCP may occupy. Whenever there is a chance to save a little code, we must do so! Redundant code is something a good programmer tries to avoid. By "packing" your code the program will be smaller, and will run faster, though it may be no easier to understand either by yourself, or others.

When the BIOS passes control of the system to the CCP the user code and drive designation is in register "C," we have extracted the user code, and passed the data to SETUSR. SETUSR now calls the DOS system and places this data into the drive/user byte at location four, page zero, for use by other portions of the system. If you find yourself confused about the concept of user areas, and their storage in page zero, refresh your memory by referring to the first installment of this series.

The section of code that we are now concerned with could have been written as:

```
LD    E,A     ;place extracted user area code in "E" for
              ;DOS call
LD    C,2ØH   ;load function call number for set user code
CALL  BDOS    ;and make the DOS call via location five
```

But, again, when we are trying to pack the greatest number of features into a small area, every byte counts, and duplicated code is wasted code.

While we are here, and while we are concerning ourselves with the setting of user areas, note how the rest of this code block is formed. Note also the trick of placing a variable position in the space set aside for literal data. Whomever thought up this one deserves a pat on the back!

```
; reset user number if changed
;
RESETUSR:
TMPUSR  EQU     $+1             ;pointer for in-the-code modification
        LD      A,Ø             ;2nd byte (immediate arg) is tmpusr
```

LISTING 1

```
-----------------------------------------------------------
;                          Hermit  Software's
;                       Modified  CROWE  Assembler
;                           Source Code File
;                  (c) 1985, 1986 C.  Thomas Hilton
;        Primary
;        Hardware: Ampro Series 1ØØ, 1A CPU
;                  (Original Little Board)
;                  Ampro Series 1ØØ, 1B CPU
;                  (Bios 3.4)
;        System:   CP/M 2.2
;                  (Ampro Standard Version)
;        Function: A True Z-8Ø Replacement Console Command Processor To:
;                  1. Restore AUTOCOMMAND Function To Non-ZCPR3 CP/M
;                  2. Enhance Standard CP/M Console Functions
;        Version Author:   C. Thomas Hilton
;        Index:
;                CCP.CRW          CCP Main Loop
;                CCPA.CRW         CROWE Chain File Containing CCP Commands
        LIST
        TITLE   'CCP Main Loop'
        NLIST
;           ___ terminal and 'type' customization equates ___
NO      EQU     Ø
YES     EQU     ØFFH                ;conditional logic boolean declarations
OLDBIOS EQU     NO                  ;set to YES if not BIOS 3.4
NEWBIOS EQU     YES                 ;set to YES if BIOS 3.4 but make only one
                                    ;bios definition to YES, the other BIOS option
                                    ;must be set to NO
        IF      OLDBIOS
AMPCCP  EQU     ØD8ØØH              ;ccp location for ampro series 1ØØ
BIOS    EQU     ØEEØØH              ;and version 1.X - 2.X BIOS systems
        ENDIF
        IF      NEWBIOS
AMPCCP  EQU     ØD4ØØH              ;ccp location for 6ØK Ampro 1B CPU
BIOS    EQU     ØEAØØH              ;location for new Ampro BIOS 3.4
                                    ;and my AVS voice BIOS Version 1.Ø
        ENDIF
CR      EQU     ØDH                 ;character: carriage return
LF      EQU     ØAH                 ;character: line feed
TAB     EQU     Ø9H                 ;character: tab
ESC     EQU     1BH                 ;character: escape
CTRLC   EQU     Ø3H                 ;character: control-c
WBOOT   EQU     ØØH                 ;cp/m warm boot address
UDFLAG  EQU     Ø4H                 ;user num in high nybble, disk in low
BDOS    EQU     Ø5H                 ;bdos function call entry pt
TFCB    EQU     5CH                 ;default fcb buffer
TBUFF   EQU     8ØH                 ;default disk i/o buffer
TPA     EQU     Ø1ØØH               ;base of tpa
MSIZE   EQU     61                  ;ampro cp/m size
CBBUFF  EQU     BIOS+62H
RCPM    EQU     NO                  ;set to true if ccp is for a BBS system
NLINES  EQU     24                  ;number of lines on crt screen
PGDFLG  EQU     'N'                 ;this flag reverses the default effect
MAXUSR  EQU     15                  ;maximum user number accessible
SYSFLG  EQU     'U'                 ;for dir command: list $sys and $dir
SOFLG   EQU     'S'                 ;for dir command: list $sys files only
DEFUSR  EQU     Ø                   ;default user number for com files
;
        ORG     AMPCCP
;
ENTRY:  JP      CCP                 ; process potential default command
        JP      CCP1                ; do not process potential default command
;
BUFLEN  EQU     8Ø                  ;maximum buffer length
MBUFF:  BYTE    BUFLEN              ;maximum buffer length
CBUFF:  BYTE    Ø                   ;number of valid chars in command line
CIBUFF: DATA    '        '          ;default (cold boot) command
CIBUF:  BYTE    Ø                   ;command string terminator
        RSRV    BUFLEN-($-CIBUFF)+1     ;total is 'buflen' bytes
;
; ccp starting points
;
; start ccp and don't process default command stored
;
CCP1:   XOR     A                   ;set no default command
        LD      (CBUFF),A
;
; start ccp and possibly process default command
;
CCP:    LD      SP,STACK            ;reset stack
        PUSH    BC
        LD      A,C                 ;c=user/disk number (see loc 4)
        RRA                         ;extract user number
        RRA
        RRA
        RRA
        AND     ØFH
        LD      E,A                 ;set user number
        CALL    SETUSR
        CALL    RESET               ;reset disk system
        POP     BC
        LD      A,C                 ;c=user/disk number (see loc 4)
        AND     ØFH                 ;extract default disk drive
        LD      (TDRIVE),A          ;set it
        JR      Z,NLOG          ;skip if Ø...already logged
        CALL    LOGIN           ;log in default disk
NLOG:   LD      A,(CBBUFF)      ;is there system command to execute?
        OR      A
        JR      NZ,CBPROC       ;if not zero there is a command
        LD      A,(CBUFF)
        OR      A
        JP      NZ,RS1
        JR      RESTRT
CBPROC: LD      BC,9
        LD      HL,CBBUFF
        LD      DE,CBUFF
        LDIR
        XOR     A
        LD      (CBBUFF),A
        JP      RS1
;
; prompt user and input command line from him
;
RESTRT: LD      SP,STACK        ;reset stack
        XOR     A
        LD      (CBUFF),A
;
; print prompt (du>)
;
        CALL    CRLF            ;print prompt
        CALL    GETDRV          ;current drive is part of prompt
        ADD     A,'A'           ;convert to ascii a-p
        CALL    CONOUT
        CALL    GETUSR          ;get user number
        CP      1Ø              ;user < 1Ø?
        JR      C,RSØØ
        SUB     1Ø              ;subtract 1Ø from it
        PUSH    AF              ;save it
        LD      A,'1'           ;output 1Ø's digit
        CALL    CONOUT
        POP     AF
RSØØ:   ADD     A,'Ø'           ;output 1's digit (convert to ascii)
        CALL    CONOUT
;
; read input line from user
;
RSØØØ:  CALL    REDBUF          ;input command line from user
;
; process input line
;
RS1:    CALL    CNVBUF          ;capitalize command line, place ending Ø,
                                ;and set cibptr value
        CALL    DEFDMA          ;set tbuff to dma address
        CALL    GETDRV          ;get default drive number
        LD      (TDRIVE),A      ;set it
        CALL    SCANER          ;parse command name from command line
        CALL    NZ,ERROR        ;error if command name contains a '?'
        LD      DE,RSTCCP       ;put return address of command
        PUSH    DE              ;on the stack
        LD      A,(TEMPDR)      ;is command of form 'd:command'?
        OR      A               ;nz=yes
        JP      NZ,COM          ;immediately
        CALL    CMDSER          ;scan for ccp-resident command
        JP      NZ,COM          ;not ccp-resident
        LD      A,(HL)          ;found it:  get low-order part
        INC     HL              ;get high-order part
        LD      H,(HL)          ;store high
        LD      L,A             ;store low
        JP      (HL)            ;execute ccp routine
;
; entry point for restarting ccp and logging in default drive


;
RSTCCP: XOR     A
        LD      (CBUFF),A
        CALL    DLOGIN          ;log in default drive
;
; entry point for restarting ccp without logging in default drive
;
RCCPNL: CALL    SCANER          ;extract next token from command line
        LD      A,(FCBFN)       ;get first char of token
        SUB     ' '             ;any char?
        LD      HL,TEMPDR
        OR      (HL)
        JP      NZ,ERROR
        JP      RESTRT
;
;-------------------------------------------------------------
; NOTE: The segments of code above are discussed In Part One of
;       this series. Please note the changes to allow a 6ØK
;       system using the Ampro BIOS 3.4.
;
;       The code which follows is discussed in Part Two of this
;       series.
;-------------------------------------------------------------
;
```

The equate TMPUSR is set to the current program counter value, plus one. Since the current position, once assembled is the load instruction, "LD," TMPUSR is pointing to the position occupied by the "0." Any byte that is loaded into TMPUSR will be loaded into the "A" register in the place of the default zero value. Such a sequence would appear as:

```
LD      (TMPUSR),A
```

where we will say that "A" has the value of 15. This would be the same as saying, when RESETUSR was encountered,

```
; reset user number if changed
;
RESETUSR:
TMPUSR  EQU     $+1                 ;pointer for in-the-code modification
        LD      A,15                ;2nd byte (immediate arg) is tmpusr
        LD      E,A                 ;place in e
        JR      SETUSR              ;then go set user
```

After having defined the user area we wish the system to place in location four, we then jump down to SETUSR.

```
LISTING 3
-----------------------------------------------
;
; bdos function routines
;
;
; return number of current disk in a
;
GETDRV: LD      C,19H
        JR      BDOSJP
;
; set 8Øh as dma address
;
DEFDMA: LD      DE,TBUFF            ;8Øh=tbuff
DMASET: LD      C,1AH
        JR      BDOSJP
;
RESET:  LD      C,ØDH
BDOSJP: JP      BDOS
;
LOGIN:  LD      E,A
        LD      C,ØEH
        JR      BDOSJP              ;save some code space
```

Other portions of the program, in our main loop, such as the code portion noted below, also use the GETUSR routine.

```
; print prompt (du>)
;
        CALL    CRLF                ;print prompt
        CALL    GETDRV              ;current drive is part of prompt
        ADD     A,'A'               ;convert to ascii a-p
        CALL    CONOUT
-----> CALL    GETUSR              ;get user number
        CP      1Ø                  ;user < 1Ø?
```

In this example we need to know what the current user area is so that we can display it in the "AX>" prompt, where "X" is the current user area.

DOS function 20H, used when we jump to SETUSR to load the "C" register with 20H, is one of CP/M's dual mode function calls. If we put FFH in register "E" the current user number will be returned in the "A" register. If a value other than FF is placed in the "E" register, that value will be set, by BDOS, as the current user area. FFH serves as a "flag" to determine which function, set the user code, or return the user code, the DOS routine is to perform.

```
GETUSR: LD      E,ØFFH              ;get current user number
SETUSR: LD      C,2ØH               ;set/or get user number
        JR      BDOSJP              ;more space saving
```

After any of the above conditions we do a jump to a common BDOS handling routine. The symbol BDOS has been defined to represent location five, in page zero. This actual code is shown at the bottom of listing 3, as:

```
BDOSJP:     JP      BDOS
```

Moving down our primary source loop to the next subroutine call, we find that RESET is our next victim.

Having changed the user area code, we now have to be sure that everyone is aware of the fact, as all portions of the system must work together, if the system is to work properly.

```
RESET:  LD      C,ØDH
BDOSJP: JP      BDOS
```

The RESET function, DOS function 13, restores the file system, and directory, to a read/write condition, selects drive "A" as the system disk, and redefines low memory variables. The default DMA buffer, used for all disk input and output, is reset to 0080H.

You may note that we have been concerned only with user areas to this point, not the drive designator which occupies the lower four bits of the drive/user byte. With this call we set the system to its default state, a fresh start, but with the BIOS defined user area. At this time I should note that CP/M doesn't really care about user areas, it only "tolerates" this byte. Therefore, everything we want to do, which concerns user areas, we must do ourselves, if we want it done right.

Following our primary CCP loop, to the next subroutine call we see that the next thing we do is get back the disk/user specification byte sent to us by BIOS, which we "pushed" onto the stack. At this time we are "logged into" disk drive "A" and the user area given to us upon entry to the CCP.

```
        POP     BC
        LD      A,C                 ;c=user/disk number (see loc 4)
        AND     ØFH                 ;extract default disk drive
        LD      (TDRIVE),A          ;set it
        JR      Z,NLOG              ;skip if Ø...already logged
        CALL    LOGIN               ;log in default disk
NLOG:   LD      A,(CBBUFF)          ;is there system command to execute?
```

Disk designators are not defined by letters, but by numbers. The letter designation is something purely for human consumption, whose reason is unknown to me. Perhaps I am not smart enough to use numbers, I don't know, (sigh). Figure 2 shows the internal designation table for disk drives.

```
        ØØH = DRIVE A     Ø8H = DRIVE I
        Ø1H = DRIVE B     Ø9H = DRIVE J
        Ø2H = DRIVE C     ØAH = DRIVE K
        Ø3H = DRIVE D     ØBH = DRIVE L
        Ø4H = DRIVE E     ØCH = DRIVE M
        Ø5H = DRIVE F     ØDH = DRIVE N
        Ø6H = DRIVE G     ØEH = DRIVE O
        Ø7H = DRIVE H     ØFH = DRIVE P

        NOTE: The upper four bits, represented by a "Ø" in the above
figure is where the user area code is stored.
```

Figure 2

In any event, we get the byte back off the stack, and perform a logical AND upon it. I am assuming that you know how a logical AND works. Since the upper nybble of the operator is a zero value, in hex, the user code is stripped off, not included in the resulting value. The "F" value, or 16 decimal, is large enough to permit the total number of disk drive designators in Figure 2 to pass through this boolean function.

When this logical operation is performed against the drive/user byte certain Z-80 flags are set. If the disk drive number is other than a zero value the zero flag will not be set. While the flag will be set at the time the operation is performed, we do not want to act upon it at this time.

Before we act upon the logical AND, we want to store the default drive value for later, internal use. We do not want to bother BDOS any more than we have to. It resets more values than we want changed. We therefore store this value in a temporary drive register, which is no more than a memory location set aside for the purpose. This is how variables are handled in machine language.

If the value of the drive sent us was a zero value, which Figure 2 tells us is the normal "A" drive, we do not have to log in a new drive. In this case we jump over the call to LOGIN.

In the event that the BIOS has specified that we begin, and always return to, a drive other than drive "A" then the "Z" flag will not be set, indicating a nonzero result of the logical AND function. If this is the case we must then assign the specified disk drive.

```
LOGIN:  LD      E,A
        LD      C,ØEH
        JR      BDOSJP          ;save some code space
```

The log in function is accomplished by loading the value left in the "A" register, from the AND function, into the "E" register, and calling BDOS function 14.

BDOS function 14 is the "SELECT DRIVE" function. When this function is called, the selected disk drive, represented by the number in register "E," is placed in an "active" mode. The current directory information is discarded, and the new directory information from the specified drive is loaded into memory. As we have already defined the user area, the directory information acted upon by various programs will return only those programs whose directory code represents the current user area.

We have placed the system on-line as per our instructions from the BIOS. Having done all of this we now check to see if there is an Autocommand function to be executed. As might be inferred, (I heard that guy in the back!), yes, it is possible to boot to a drive other than drive "A." All that is required is to place the desired disk and user area into register "C," and jump into the CCP! It's not my fault no one told you this before!! The system must know where to get the DOS and CCP from the disk, but once loaded no one really cares all that much.

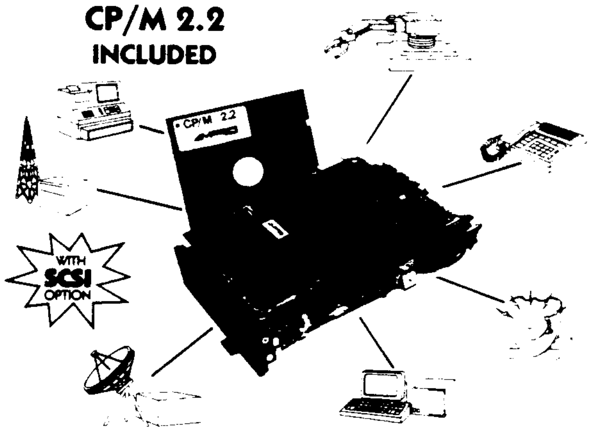
Several of the other subroutine calls in the main loop also make BDOS calls, which are also shown in Listing 3. These are relatively simple functions, which I will allow you, the reader, to explore by yourself. At this sitting we have a great deal more to discuss, of higher priority, so we must move along quickly.

```
LISTING 4
----------------------------------------------------------
;
; output char in reg a to console and don't change bc
;
; output <crlf>
;
CRLF:    LD      A,CR
         CALL    CONOUT
         LD      A,LF            ;fall thru to conout
;
CONOUT:  PUSH    BC
         LD      C,Ø2H
OUTPUT:  LD      E,A
         PUSH    HL
         CALL    BDOS
         POP     HL
         POP     BC
         RET
```

Listing 4 presents the basic character output routines called by the section of the main loop concerned with printing the prompt, at segment:

```
; print prompt (du>)
;
        CALL    CRLF            ;print prompt
        CALL    GETDRV          ;current drive is part of prompt
        ADD     A,'A'           ;convert to ascii a-p
        CALL    CONOUT
        CALL    GETUSR          ;get user number
```

The first thing we want to do is clear a fresh line to print the "A0>" prompt upon. CRLF does this for us, using CONOUT, the generic "print a character" subroutine. BASIC programmers should note that the expected carriage return/line feed sequence is not automatic at the assembler, or machine language level. No one at this level of the system is going to take the blame for printing any character it isn't supposed to. The ruling concept is to expect nothing, take nothing for granted, and specifically define exactly what must be printed. "If you don't, it won't."



```
; output char in reg a to console and don't change bc
;
; output (crlf)
;
CRLF:   LD      A,CR
        CALL    CONOUT
        LD      A,LF                    ;fall thru to conout
```

To present the "cursor back to start, and scroll a line," function we first load a RETURN into the "A" register and call CONOUT as a subroutine. We then reload the "A" register with a line feed, (CR and LF equates at head of Listing 1), and "fall through" to CONOUT, returning to the caller from CONOUT.

```
CONOUT: PUSH    BC
        LD      C,Ø2H
OUTPUT: LD      E,A
        PUSH    HL
        CALL    BDOS
        POP     HL
        POP     BC
        RET
```

The operation of CONOUT actually requires very little explanation. I would make note of the fact that all user registers not actually used by the current routines are saved upon the stack. This is done because most DOS systems, who in turn call the BIOS, do not take the time to save the users registers. They do not return them to the caller with the same values they had before the call. I have said it a thousand times, and I will repeat it here, "SAVE THOSE REGISTERS!!" In doing so you will save yourself a great deal of grief.

Moving right along, locate the following section of code from our main source loop:

```
; read input line from user
;
RSØØØ:  CALL    REDBUF                  ;input command line from user
;
; process input line
;
RS1:    CALL    CNVBUF                  ;capitalize command line, place ending Ø.
```

At this point we are ready to input a command from the operator. Make note that we have printed the disk drive designator, "A," and the user area, "0," which forms the familiar "A0>" prompt. Note also that we have not yet printed the ">" character. We shall print the remaining portion of the CCP prompt as we prepare to get input from the console.

See Listing 5.

```
; input next command to ccp
;
REDBUF:
RB1:
        CALL    SETUD                   ;set user and disk
        LD      A,'>'
        CALL    CONOUT
        LD      C,ØAH                   ;read command line from user
        LD      DE,MBUFF
        CALL    BDOS                    ;and fall through to SETUØD
```

The first thing we want to do is save the drive and user data, as it stands at this moment, before any command is accepted or acted upon. These functions are performed by SETUD.

```
; set user/disk flag to current user and default disk
;
SETUD:  CALL    GETUSR                  ;get number of current user
        ADD     A,A                     ;place it in high nybble
        ADD     A,A
        ADD     A,A
        ADD     A,A
        LD      HL,TDRIVE               ;mask in default drive number (low nybble)
        OR      (HL)                    ;mask in
        LD      (UDFLAG),A              ;set user/disk number
        RET
```

Once these functions of saving the current drive and user area, internally, are performed we are ready to input our command line from the operator. We print the ">" character via CONOUT, and then set up the system to do a BDOS Function 10 call. DOS call 10 is the "read a line of text" function. This call requires that we define the maximum number of characters allowed in the input line, and where this line of text is to be placed. It has to go somewhere! The parameters for this function are defined at the top of our source program. This "buffer," or place to put the console command is located at CCP+6. For a full discussion of this location please refer to PART ONE of this series. Again, the set up for input of a command line is:

```
        LD      C,ØAH                   ;read command line from user
        LD      DE,MBUFF
        CALL    BDOS                    ;and fall through to SETUØD
```

Note below, that MBUFF contains the maximum number of characters the command line may receive. MBUFF+1, or CBUFF will, upon return of the DOS function, contain the number of actual characters in the command line. DOS function 10 will return only when a carriage return is input, to signify the end of the input string, or when the maximum number of characters defined in MBUFF has been reached.

```
BUFLEN  EQU     8Ø                      ;maximum buffer length
MBUFF:  BYTE    BUFLEN                  ;maximum buffer length
CBUFF:  BYTE    Ø                       ;number of valid chars in command line
CIBUFF: DATA    '                       ;default (cold boot) command
CIBUF:  BYTE    Ø                       ;command string terminator
        RSRV    BUFLEN-($-CIBUFF)+1          ;total is 'buflen' bytes
```

The remainder of this code defines, for the assembler, the space to be reserved for this buffer, and its terminating null, or zero value character.

When the input command line has been placed in the command buffer, CBUFF, the code then drops into SETU0D, which sets an internal register.

```
;set current disk number in lower params  set user/disk flag to user Ø
;and default disk
SETUØD:
TDRIVE  EQU     $+1                     ;pointer for in-the-code modification
        LD      A,Ø                     ;2nd byte (immediate arg) is tdrive
        LD      (UDFLAG),A              ;set user/disk number
        RET
```

Having received the input command line, and taken care of a little housekeeping, we now must concern ourselves with trying to make sense of the data sent to us by the operator. This is not always an easy task. Our first task is to convert the data into a generic, universal format. In this system all terminal input is converted to uppercase characters by CNVBUF.

```
; process input line
;
RS1:    CALL    CNVBUF                  ;capitalize command line, place ending Ø.
```

In addition to converting the command string into all upper case characters, CNVBUF assures that there is a terminating null at the end of the command. This terminator is just one more way to tell when we have reached the end of the human's demands of us. The computer does not understand human speech. We must find a way to simulate understanding. The conversion to upper case is the first step in this process.

```
; capitalize string (ending in Ø) in cbuff and set ptr for parsing
;
CNVBUF: LD      HL,CBUFF                ;pt to user's command
        LD      B,(HL)                  ;char count in b
        INC     B                       ;add 1 in case of zero
CB1:    INC     HL                      ;pt to 1st valid char
        LD      A,(HL)                  ;capitalize command char
        CALL    UCASE
        LD      (HL),A
        DJNZ    CB1                     ;continue to end of command line
CB2:    LD      (HL),Ø                  ;store ending <null>
        LD      HL,CIBUFF               ;set command line ptr to 1st char
        LD      (CIBPTR),HL
        RET
```

The conversion process begins by setting a 16 bit register to point to the beginning of the command character string. Remember that CBUFF, upon return from DOS function 10, contains the actual number of characters returned in the command line. We set this value in register "B" so that we can use a powerful command of the Z-80 command set, Decrement and Jump Not Zero, or DJNZ. This command is unique to the Z-80, which is why sane people do not use an 8080 assembler on a machine using a Z-80 processor, though some people do try...

Having set the number of characters in "B" and having advanced the "HL" register, serving as our "pointer," to point to the first valid character in the command line, we now load the "A" register with that character. With the command character in "A" we may now call UPCASE, who converts the character, if required, into uppercase format.

```
LISTING 5
----------------------------------------------------------------
;
; input next command to ccp
;
REDBUF:
RB1:
        CALL    SETUD               ;set user and disk
        LD      A,'>'
        CALL    CONOUT
        LD      C,ØAH               ;read command line from user
        LD      DE,MBUFF
        CALL    BDOS                ;and fall through to SETUØD
;set current disk number in lower params  set user/disk flag to user Ø
;and default disk
SETUØD:
TDRIVE  EQU     $+1                 ;pointer for in-the-code modification
        LD      A,Ø                 ;2nd byte (immediate arg) is tdrive
        LD      (UDFLAG),A          ;set user/disk number
        RET
;
; set user/disk flag to current user and default disk
;
SETUD:  CALL    GETUSR              ;get number of current user
        ADD     A,A                 ;place it in high nybble
        ADD     A,A
        ADD     A,A
        ADD     A,A
        LD      HL,TDRIVE           ;mask in default drive number (low nybble)
        OR      (HL)                ;mask in
        LD      (UDFLAG),A          ;set user/disk number
        RET
;
;
; capitalize string (ending in Ø) in cbuff and set ptr for parsing
;
CNVBUF: LD      HL,CBUFF            ;pt to user's command
        LD      B,(HL)              ;char count in b
        INC     B                   ;add 1 in case of zero
CB1:    INC     HL                  ;pt to 1st valid char
        LD      A,(HL)              ;capitalize command char
        CALL    UCASE
        LD      (HL),A
        DJNZ    CB1                 ;continue to end of command line
CB2:    LD      (HL),Ø              ;store ending <null>
        LD      HL,CIBUFF           ;set command line ptr to 1st char
        LD      (CIBPTR),HL
        RET
;
; convert char in a to upper case
;
UCASE:  CP      61H                 ;lower-case a
        RET     C
        CP      7BH                 ;greater than lower-case z?
        RET     NC
        AND     5FH                 ;capitalize
        RET
```

```
; convert char in a to upper case
;
UCASE:  CP      61H                 ;less than lower-case a?
        RET     C                   ;yes, so leave it alone
        CP      7BH                 ;greater than lower-case z?
        RET     NC                  ;yes, so ignore it as well
        AND     5FH                 ;capitalize
        RET
```

I will leave the process of the actual conversion to upper case routine to you, the reader, to explore. Get out your ASCII code chart, and keep in mind the actual hex code value of all lower case characters, and the function of the logical AND.

We now repeat the incrementing of the pointer, converting of characters to upper case, and the decrementing of the character count until the character counter reaches a zero value. This loop has been brought to you by the good folks who gave us DJNZ, the programmer's friend!

When the "B" register zeros out we fall through into CB2, who places the ending null character at the end of the string as noted by the, "number of characters in the line" indicator, not the number of characters that could have been in the line. In this way we do not try to interpret random trash that may fill a command line with less than 80 characters. Yes, I know other CCP programs allow longer command lines. I thought meaningful error messages were more important. The normal CP/M SUBMIT function is not implemented in my CCP. Why? Because I never use it. I prefer EX14.COM or other submit type programs that do not erase the submit file when they are finished.

```
CB2:    LD      (HL),Ø              ;store ending <null>
        LD      HL,CIBUFF           ;set command line ptr to 1st char
        LD      (CIBPTR),HL
        RET
```

Now we have everything set into a universal format, and are ready to try and figure out what those upper case characters mean! But, before we can get to the "good" stuff we have set the DMA address to be used in any upcoming function:

```
; set 8Øh as dma address
;
DEFDMA: LD      DE,TBUFF            ;8Øh=tbuff
DMASET: LD      C,1AH
        JR      BDOSJP
```

Even though RESET has set the default DMA buffer for all disk in/out to 0080H, do you REALLY trust BDOS? I don't, and something may have changed if we have executed an internal CCP command. Since I will never take anything for granted, nor assume a thing, let's reset the DMA buffer, OK? Call me paranoid if you want! I don't care!

The next thing we have to do is store everything we have done thus far, because it is almost certain that we will be leaving the CCP for awhile, real soon.

```
        CALL    GETDRV              ;get default drive number
        LD      (TDRIVE),A          ;set it
```

From this point we check to see if the user has input a command to only change a drive, or wants to get a file from another disk drive.

A Little Homework

TCJ is a bimonthly magazine. I am taking far too long in discussing code that we will be using over and over again. There are other, important, functions I want to cover in this issue. If you have been following me thus far you are ready to strike out and do some work on your own. Of all the segments I considered, the SCANER code segment, in Listing 6, is the one I feel you would benefit the most from exploring yourself. I had to cut back somewhere, and this code, while it looks complex, contains groups of rather elementary routines, and is well documented. I would recommend, as reference works, the CP/M manual from Digital Research, and Ampro, and/or:

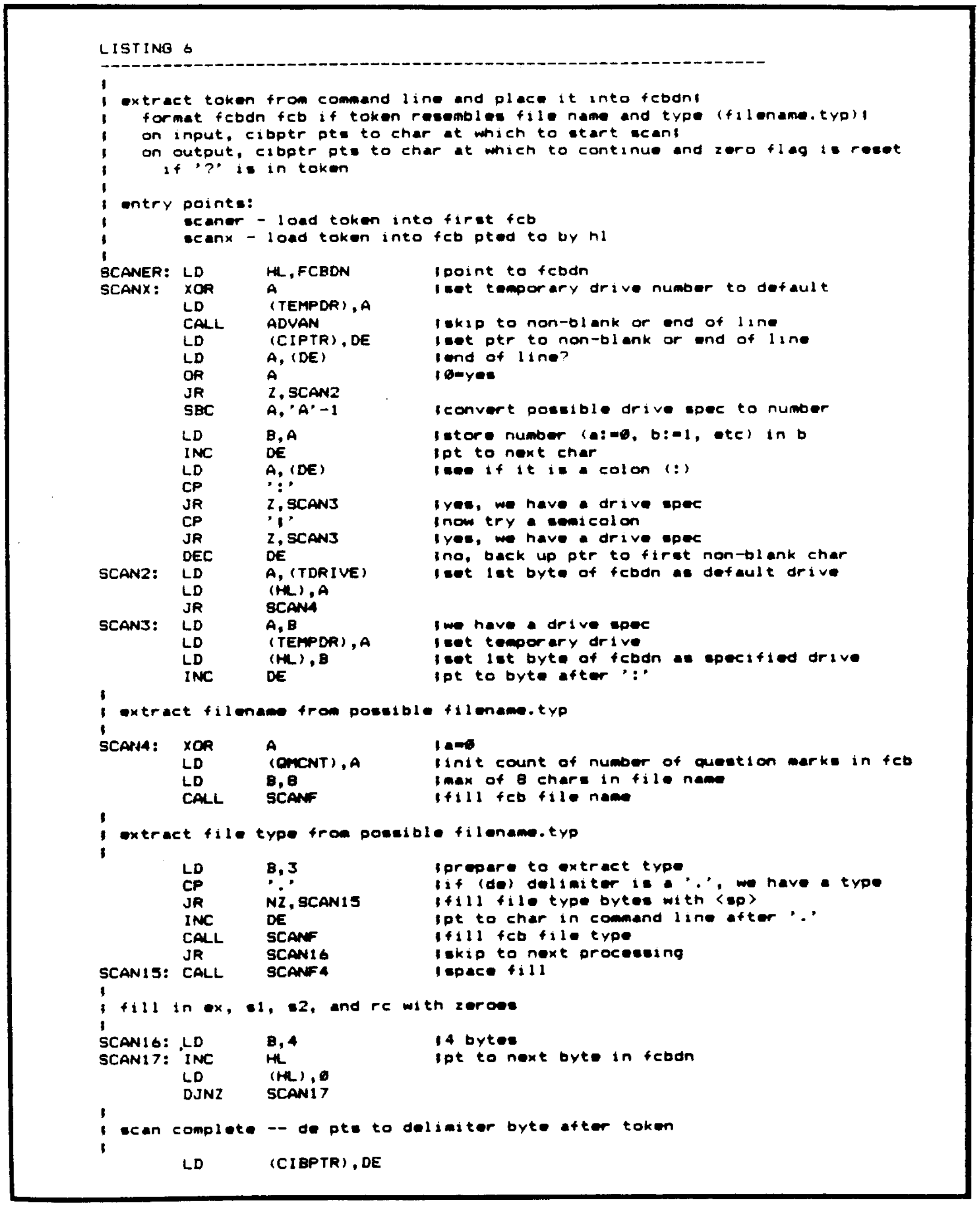
INSIDE CP/M, A GUIDE FOR USERS AND PROGRAMMERS
By David E. Cortesi
CBS COLLEGE PUBLISHING
383 Madison Avenue, New York, NY 10017
ISBN 0-03-059558-4

When next we meet you will be expected to understand the functioning of the following code segments taken from the main loop:

```
CALL    SCANER          ;parse command name from command line
CALL    NZ,ERROR        ;error if command name contains a '?'
```

Another reason I do not want to spend a great deal of time on the SCANER code is that this is a three part series to make you feel comfortable with the CCP's internal workings. The first three parts serve no other real function, though the CCP is quite usable. In later portions, when we begin modifying the understood CCP, it is doubtful that we will be using the SCANER code system.

Meanwhile, back at the bit works, if SCANER detects an error, based upon the "Z" flag, which is used for the purpose of telling the caller an error has occurred, we jump out of the loop. Control is then passed to the ERROR routine, and the CCP is



```
LISTING 6
-------------------------------------------------------------------
;
; extract token from command line and place it into fcbdn;
;   format fcbdn fcb if token resembles file name and type (filename.typ);
;   on input, cibptr pts to char at which to start scan;
;   on output, cibptr pts to char at which to continue and zero flag is reset
;     if '?' is in token
;
; entry points:
;       scaner - load token into first fcb
;       scanx - load token into fcb pted to by hl
;
SCANER: LD      HL,FCBDN        ;point to fcbdn
SCANX:  XOR     A               ;set temporary drive number to default
        LD      (TEMPDR),A
        CALL    ADVAN           ;skip to non-blank or end of line
        LD      (CIPTR),DE      ;set ptr to non-blank or end of line
        LD      A,(DE)          ;end of line?
        OR      A               ;Ø=yes
        JR      Z,SCAN2
        SBC     A,'A'-1         ;convert possible drive spec to number

        LD      B,A             ;store number (a:=Ø, b:=1, etc) in b
        INC     DE              ;pt to next char
        LD      A,(DE)          ;see if it is a colon (:)
        CP      ':'
        JR      Z,SCAN3         ;yes, we have a drive spec
        CP      ';'             ;now try a semicolon
        JR      Z,SCAN3         ;yes, we have a drive spec
        DEC     DE              ;no, back up ptr to first non-blank char
SCAN2:  LD      A,(TDRIVE)      ;set 1st byte of fcbdn as default drive
        LD      (HL),A
        JR      SCAN4
SCAN3:  LD      A,B             ;we have a drive spec
        LD      (TEMPDR),A      ;set temporary drive
        LD      (HL),B          ;set 1st byte of fcbdn as specified drive
        INC     DE              ;pt to byte after ':'
;
; extract filename from possible filename.typ
;
SCAN4:  XOR     A               ;a=Ø
        LD      (QMCNT),A       ;init count of number of question marks in fcb
        LD      B,8             ;max of 8 chars in file name
        CALL    SCANF           ;fill fcb file name
;
; extract file type from possible filename.typ
;
        LD      B,3             ;prepare to extract type
        CP      '.'             ;if (de) delimiter is a '.', we have a type
        JR      NZ,SCAN15       ;fill file type bytes with <sp>
        INC     DE              ;pt to char in command line after '.'
        CALL    SCANF           ;fill fcb file type
        JR      SCAN16          ;skip to next processing
SCAN15: CALL    SCANF4          ;space fill
;
; fill in ex, s1, s2, and rc with zeroes
;
SCAN16: LD      B,4             ;4 bytes
SCAN17: INC     HL              ;pt to next byte in fcbdn
        LD      (HL),Ø
        DJNZ    SCAN17
;
; scan complete -- de pts to delimiter byte after token
;
        LD      (CIBPTR),DE
```

Listing 6 continued

```
;
; set zero flag to indicate presence of '?' in filename.typ
;
        LD      A,(QMCNT)       ;get number of question marks
        OR      A               ;set zero flag to indicate any '?'
        RET
;
;  scanf -- scan token pted to by de for a max of b bytes; place it into
;    file name field pted to by hl; expand and interpret wild cards of
;    '*' and '?'; on exit, de pts to terminating delimiter
;
SCANF:  CALL    SDELM           ;done if delimiter encountered - <sp> fill
        JR      Z,SCANF4
        INC     HL              ;pt to next byte in fcbdn
        CP      '*'             ;is (de) a wild card?
        JR      NZ,SCANF1       ;continue if not
        LD      (HL),'?'        ;place '?' in fcbdn and don't advance de if so
        CALL    SCQ             ;scanner count question marks
        JR      SCANF2
SCANF1: LD      (HL),A          ;store filename char in fcbdn
        INC     DE              ;pt to next char in command line
        CP      '?'             ;check for question mark (wild)
        CALL    Z,SCQ           ;scanner count question marks
SCANF2:
        DJNZ    SCANF           ;decrement char count until 8 elapsed
SCANF3:
        CALL    SDELM           ;8 chars or more - skip until delimiter
        RET     Z               ;zero flag set if delimiter found
        INC     DE              ;pt to next char in command line
        JR      SCANF3
;
;  fill memory pointed to by hl with spaces for b bytes
;
SCANF4: INC     HL              ;pt to next byte in fcbdn
        LD      (HL),' '        ;fill filename part with <sp>
        DJNZ    SCANF4
        RET
;
;  increment question mark count for scanner
;    this routine increments the count of the number of question marks in
;    the current fcb entry
;
SCQ:    LD      A,(QMCNT)       ;get count
        INC     A               ;increment
        LD      (QMCNT),A       ;put count
        RET
;
; fill fcb at hl with '?'
;
FILLQ:  LD      B,11            ;number of chars in fn & ft
FQLP:   LD      (HL),'?'        ;store '?'
        INC     HL
        DJNZ    FQLP
        RET
;
; command file control block
;
FCBDN:  RSRV    1               ;disk name
FCBFN:  RSRV    8               ;file name
FCBFT:  RSRV    3               ;file type
        RSRV    1               ;extent number
        RSRV    2               ;s1 and s2
        RSRV    1               ;record count
FCBDM:  RSRV    16              ;disk group map
FCBCR:  RSRV    1               ;current record number
PAGCNT: BYTE    NLINES-2        ;lines left on page
CHRCNT: BYTE    0               ;char count for read
```

```
QMCNT:  BYTE    0               ;question mark count for fcb token scanner
;
        END

;
ADVAN:  LD      DE,(CIBPTR)
;
; skip string pted to by de (string ends in 0) until end of string
;   or non-blank encountered (beginning of token)
;
SBLANK: LD      A,(DE)
        OR      A
        RET     Z
        CP      ' '
        RET     NZ
        INC     DE
        JR      SBLANK

; check to see if de pts to delimiter; if so, ret w/zero flag set
;
SDELM:  LD      A,(DE)
        OR      A               ;0=delimiter
        RET     Z
        CP      ' '             ;error if less than a space
        JR      C,ERROR
        RET     Z               ;<sp>=delimiter
        CP      '='             ;'='=delimiter
        RET     Z
        CP      '.'             ;'.'=delimiter
        RET     Z
        CP      ':'             ;':'=delimiter
        RET     Z
        CP      ';'             ;';'=delimiter
        RET     Z
        RET

;
; invalid command -- print it
;
ERROR:  CALL    SYSPTC          ;print message
        DATA    'No File Or Command Found By Name Of ' ! BYTE 0
        LD      HL,(CIPTR)      ;pt to beginning of command line
ERR2:   LD      A,(HL)          ;get char
        CP      20H             ;get out of loop if space
        JR      Z,ERR1
        OR      A               ;get out of loop if 0
        JR      Z,ERR1
        PUSH    HL              ;save ptr to error command char
        CALL    CONOUT          ;print command char
        POP     HL              ;get ptr
        INC     HL              ;pt to next
        JR      ERR2            ;continue
ERR1:
        JP      RESTRT          ;restart ccp
                                ;reset must be done before login
```

restarted in the hopes that the human can get it right this time.

Failing dection of a drive/user type command, we fall through into the following code.

```
LD      DE.RSTCCP       ;put return address of command
PUSH    DE              ;on the stack
LD      A,(TEMPDR)      ;is command of form 'd:command'?
OR      A               ;nz=yes
JP      NZ.COM          ;immediately
```

I must digress a bit, though I am assuming that previous discussions have armed you to the point where you may proceed without my help, and discuss the way subroutines are handled. Whenever we issue a CALL instruction the current program counter value is placed upon the stack. This value indicates the 16 bit address which is to serve as the "way home" when a RET instruction is encountered. The RET instruction takes the first two ~~bytes~~ from the top of the stack, and assembles them into a 16 bit address. It assumes this 16 bit number is the way back to the caller of a subroutine. It would follow that if the stack becomes filled with garabage and a RET instruction is executed, the program will go into high orbit, or just insane. We must always pay attention to what is on the stack, and that it is always pointing to the proper data at the proper time.

This knowledge also allows us to simulate the function of a CALL instruction for our own purposes. This is what the code below actually does.

```
LD      DE.RSTCCP       ;put return address of command
PUSH    DE              ;on the stack
```

As it is almost certain, unless our human has input some real trash, that we will either be executing an internal CCP command, or executing a user program. In either event we may return to the CCP instead of doing a "warm boot." What the code above does is place the address of the "ReSTart the CCP" routine onto the stack. This is done by loading the "DE" register pair with the address and pushing it upon the stack. Whenever a RET instruction is encountered, in our internal CCP commands or perhaps a user program, the address of RSTCCP will be the return address, not the code segments following the initial jump. How we make the call we will discuss in a very short time.

```
LD      A,(TEMPDR)      ;is command of form 'd:command'?
OR      A               ;nz=yes
JP      NZ.COM          ;immediately
```

Having prepared for an execution jump, not a call, but a jump, we now check to see if there is a command that requires us to fetch a program from another disk drive. If the temprorary drive register is any value other than drive "A" there is no use continuing our guessing game. We do a direct jump to the command file handling routine, shown in Listing 7.

```
LISTING 7
-----------------------------------------------------------------
;
; com file processing
;
COM:
        LD      A,(FCBFN)       ;any command?
        CP      ' '             ;' ' means command was 'd:' to switch
        JR      NZ,COM1         ;not <sp>, so must be transient or error
        LD      A,(TEMPDR)      ;look for drive spec
        OR      A               ;if zero, just blank
        JP      Z,RCCPNL
        DEC     A               ;adjust for log in
        LD      (TDRIVE),A      ;set default drive
        CALL    SETUØD          ;set drive with user Ø
        CALL    LOGIN           ;log in drive
        JP      RCCPNL          ;restart ccp
COM1:
        LD      A,(FCBFT)       ;file type must be blank
        CP      ' '
        JP      NZ,ERROR
        LD      HL,COMMSG       ;place default file type (com) into fcb
        LD      DE,FCBFT        ;copy into file type
        LD      BC,3            ;3 bytes
        LDIR
        CALL    MEMLOAD         ;load memory with file specified in cmd line
        RET     NZ              ;return (abort) if load error
;
; callprog is the entry point for the execution of the loaded
;   program on entry to this routine, hl must contain the execution
;   address of the program (subroutine) to execute
;
CALLPROG:
        CALL    DLOGIN          ;log in default drive
        CALL    SCANER          ;search command line for next token
        LD      HL,TEMPDR       ;save ptr to drive spec
        PUSH    HL
        LD      A,(HL)          ;set drive spec
        LD      (FCBDN),A
        LD      HL,FCBDN+1ØH    ;pt to 2nd file name
        CALL    SCANX           ;scan for it and load it into fcbdn+16
        POP     HL              ;set up drive specs
        LD      A,(HL)
        LD      (FCBDM),A
        XOR     A
        LD      (FCBCR),A
        LD      DE,TFCB         ;copy to default fcb
        LD      HL,FCBDN        ;from fcbdn
        LD      BC,33           ;set up default fcb
        LDIR
        LD      HL,CIBUFF
COM4:
        LD      A,(HL)          ;skip to end of 2nd file name
        OR      A               ;end of line?
```

Listing 7 continued

```
        JR      Z,COM5
        CP      ' '                 ;end of token?
        JR      Z,COM5
        INC     HL
        JR      COM4
;
; load command line into tbuff
;
COM5:
        LD      B,Ø                 ;set char count
        LD      DE,TBUFF+1          ;pt to char pos
COM6:
        LD      A,(HL)              ;copy command line to tbuff
        LD      (DE),A
        OR      A                   ;done if zero
        JR      Z,COM7
        INC     B                   ;incr char count
        INC     HL                  ;pt to next
        INC     DE
        JR      COM6
;
; run loaded transient program
;
COM7:   LD      A,B                 ;save char count
        LD      (TBUFF),A
        CALL    CRLF                ;new line
        CALL    DEFDMA              ;set dma to ØØ8Ø
        CALL    SETUD               ;set user/disk
;
; execution (call) of program (subroutine) occurs here
;
        CALL    TPA                 ;call transient
        CALL    DEFDMA              ;set dma to ØØ8Ø, in case
                                    ;prog changed it on us
        CALL    SETUØD              ;set user Ø/disk
        CALL    LOGIN               ;login disk
        JP      RESTRT              ;restart ccp
;
;
; transient load error
;
COM8:   POP     HL                  ;clear return address
        CALL    RESETUSR            ;reset current user number
                                    ;reset must be done before login
ERRLOG: CALL    DLOGIN              ;log in default disk
ERRJMP: JP      ERROR
COMMSG: DATA    'COM'
;
; load memory with the file whose name is specified in the command line
;   on input, hl contains starting address to load
;
MEMLOAD:
        CALL    MLOAD               ;user memory load subroutine
        PUSH    AF                  ;save return status
        CALL    RESETUSR            ;reset user number
        POP     AF                  ;get return status
        RET
;
;  memory load subroutine
;       exit points are a simple return with the zero flag set if no error,
; a simple return with the zero flag reset (nz) if memory full, or a jmp to
; com8 if com file not found
;
MLOAD:  CALL    GETUSR              ;get current user number
        LD      (TMPUSR),A          ;save it for later
        LD      (TSELUSR),A         ;temp user to select
;
;   mla is a reentry point for a non-standard cp/m modification
```

```
; this is the return point for when the .com (or get) file is not found the
; first time, drive a: is selected for a second attempt
;
MLA:
        CALL    SLOGIN              ;log in specified drive if any
        CALL    OPENF               ;open command.com file
        JR      NZ,MLA1             ;file found - load it
;
; error routine to select user Ø if all else fails
;
DFUSR   EQU     $+1                 ;mark in-the-code variable
        LD      A,DEFUSR            ;get default user
TSELUSR EQU     $+1                 ;mark in-the-code variable
        CP      DEFUSR              ;same?
        JR      Z,MLAØ              ;jump if
        LD      (TSELUSR),A         ;else put down new one
        LD      E,A
        CALL    SETUSR              ;go set new user number
        JR      MLA                 ;and try again
;
; error routine to select drive a: if default was originally selected
;
MLAØ:
        LD      HL,TEMPDR           ;get drive from current command
        XOR     A                   ;a=Ø
        OR      (HL)
        JP      NZ,COM8             ;error if already disk a:
        LD      (HL),1              ;select drive a:
        JR      MLA
;
; file found -- proceed with load
;
MLA1:   LD      HL,TPA              ;set start address of memory load
ML2:    LD      A,.HIGH.AMPCCP              ;get BODS entry point page
        CP      H                   ;are we at top of defined memory?
        JR      C,PRNLE             ;error if so
        PUSH    HL                  ;save address of next sector
        EX      DE,HL               ;... in de
        CALL    DMASET              ;set dma address for load
        LD      DE,FCBDN            ;read next sector
        CALL    SYRD
        POP     HL                  ;get address of next sector
        JR      NZ,ML3              ;read error or eof?
        LD      DE,128              ;move 128 bytes per sector
        ADD     HL,DE               ;pt to next sector in hl
        JR      ML2
;
ML3:
        DEC     A                   ;load complete
        RET     Z                   ;ok if zero, else fall thru to prnle
;
; load error
;
PRNLE:
        CALL    SYSPTC
        DATA    'File Too Big To Load!' ! BYTE Ø
        LD      A,1                 ;set non-zero to indicate error
        OR      A                   ;set flag
        RET
;
; check for specified drive and log it in if not default
;
SLOGIN: XOR     A                   ;set fcbdn for default drive
        LD      (FCBDN),A
        CALL    COMLOG              ;check drive
        RET     Z
        JR      DLOG5               ;do login otherwise
;
; check for specified drive and log in default drive if specified<>default
```

```
;
DLOGIN: CALL    COMLOG          ;check drive
        RET     Z               ;abort if same
        LD      A,(TDRIVE)      ;log in default drive
;
DLOGS:  JP      LOGIN
;
; routine common to both login routines; on exit, z set means abort
;
COMLOG:
TEMPDR  EQU     $+1             ;pointer for in-the-code modification
        LD      A,0             ;2nd byte (immediate arg) is tempdr
        OR      A               ;0=no
        RET     Z
        DEC     A               ;compare it against default
        LD      HL,TDRIVE
        CP      (HL)
        RET                     ;abort if same
;
;
OPENF:  XOR     A
        LD      (FCBCR),A
        LD      DE,FCBDN        ;fall thru to open
;
OPEN:   LD      C,0FH           ;fall thru to grbdos
;
GRBDOS: CALL    BDOS
        INC     A               ;set zero flag for error return

        RET
;
CLOSE:  LD      C,10H
        JR      GRBDOS
;
SEARF:  LD      DE,FCBDN        ;specify fcb
SEAR1:  LD      C,11H
        JR      GRBDOS
;
SEARN:  LD      C,12H
        JR      GRBDOS
;
SYRDF:  LD      DE,FCBDN        ;fall thru to read
SYRD:   LD      C,14H           ;fall thru to bdosb
;
; call bdos and save bc
;
BDOSB:  PUSH    BC
        CALL    BDOS
        POP     BC
        OR      A
        RET
```

LISTING 8

```
---------------------------------------------------------------
;
; cmdtbl (command table) scanner
;   on return, hl pts to address of command if ccp-resident, 'Z' flag set
; means ccp-resident command
;
CMDSER: LD      HL,CMDTBL       ;pt to command table
        LD      C,NCMNDS        ;set command counter
CMS1:   LD      DE,FCBFN        ;pt to stored command name
        LD      B,NCHARS        ;number of chars/command (8 max)
CMS2:   LD      A,(DE)          ;compare against table entry
        CP      (HL)
        JR      NZ,CMS3         ;no match
        INC     DE              ;pt to next char
        INC     HL
        DJNZ    CMS2            ;count down
        LD      A,(DE)          ;next char in input command must be <sp>
        CP      ' '
        JR      NZ,CMS4
        RET                     ;command is ccp-resident (zero flag set)
CMS3:   INC     HL              ;skip to next command table entry
        DJNZ    CMS3
CMS4:   INC     HL              ;skip address
        INC     HL
        DEC     C               ;decrement table entry number
        JR      NZ,CMS1
        INC     C               ;clear zero flag
        RET                     ;command is disk-resident (zero flag clear)
```

LISTING 9

```
---------------------------------------------------------------

; ccp built-in command table
;
NCHARS  EQU     4               ;number of chars/command
;
; ccp command name table
;   each table entry is composed of the 4-byte command and 2-byte address
;
CMDTBL:
        DATA    'DIR '
        WORD    DIR
        DATA    'LIST'
        WORD    PLIST
        DATA    'HELP'
        WORD    HELP
        DATA    'READ'
        WORD    READ
        DATA    'USER'
        WORD    USER
        DATA    'PATH'
        WORD    PATH
;
        IF      RCPM=NO         ;for non-rcpm
        DATA    'ERA '
        WORD    ERA
        DATA    'SAVE'
        WORD    SAVE
        DATA    'REN '
        WORD    REN
        DATA    'RUN '
        WORD    RUN
        DATA    'JUMP'
        WORD    JUMP
        DATA    'DO  '
        WORD    DO
        DATA    'LOAD'
        WORD    LOAD
        ENDIF
;
NCMNDS  EQU     ($-CMDTBL)/(NCHARS+2)
;
; print string (ending in 0) pted to by hl
;
SYSP1:  LD      A,(HL)          ;get next byte
        CALL    CONOUT          ;print char
        LD      A,(HL)          ;get next byte again for test
        INC     HL              ;pt to next byte
        OR      A               ;set flags
        RET     Z               ;done if zero
        RET     M               ;done if msb set
        JR      SYSP1
;
CIBPTR: WORD    CIBUFF          ;pointer to command input buffer
CIPTR:  WORD    CIBUF           ;current pointer
        RSRV    32              ;stack area
```

Listing 9 continued

```
        STACK    EQU      $                  ;top of stack
        ;
        ; command file control block
        ;
        FCBDN:   RSRV     1                  ;disk name
        FCBFN:   RSRV     8                  ;file name
        FCBFT:   RSRV     3                  ;file type
                 RSRV     1                  ;extent number
                 RSRV     2                  ;s1 and s2
                 RSRV     1                  ;record count
        FCBDM:   RSRV     16                 ;disk group map
        FCBCR:   RSRV     1                  ;current record number
        PAGCNT:  BYTE     NLINES-2           ;lines left on page
        CHRCNT:  BYTE     0                  ;char count for read
        QMCNT:   BYTE     0                  ;question mark count for fcb token scanner
        ;
        ;======================================================================
        ;  NOTE: THE CODE PRESENTED THUS FAR WILL ASSEMBLE BY ITSELF INTO A WORKING
        ;        CCP WITHOUT INTERNAL COMMAND STRUCTURES.  THE ONLY ERROR RETURNED
        ;        WILL BE UNDEFINED LABELS FOR COMMAND ADDRESSES IF THERE ARE SUCH
        ;        ENTRIES IN THE COMMAND TABLE.  OTHER THAN THIS THE CODE PRESENTED
        ;        THUS FAR IS STANDALONE IN NATURE.
        ;
        ;        CHAIN FILE "A" CONTAINS ALL INTERNAL CCP COMMAND MODULES.
        ;======================================================================
        ;
               CHAIN    'CCPA'           ;chain to internal command module file
```

Failing an indicator to load a command file, (XXXXXXXX.COM), we have to check to see if the command line contains a request for an internal CCP command function. The routine CMDSER is the routine responsible for this function. In my opinion CMDSER is the most important CCP code segment! It may be considered an interpreter, not unlike what may be found in a programming language. In fact, I have written industrial process control languages in code similar to that shown in Listing 8.

```
        CALL    CMDSER          ;scan for ccp-resident command
        JP      NZ,COM          ;not ccp-resident
        LD      A,(HL)          ;found it:  get low-order part
        INC     HL              ;get high-order part
        LD      H,(HL)          ;store high
        LD      L,A             ;store low
        JP      (HL)            ;execute ccp routine
```

For a basic overview of CMDSER, and the remainder of our main loop: if a valid internal CCP command or function is found by CMDSER the "Z" flag is set as an indicator for the main loop. If the "Z" flag is not set then no internal command has been detected, and we must assume that the data in the command line is the name of a program on the current drive. If this is not the case then the COM routine will have to deal with that fact.

You will note that the form of our command table is that of a character string, representing the command, followed by the address where that command may be found. If the command table scanner detects, by a match of characters in the command line with characters in a command name, the address of the routine is loaded into the "HL" register pair.

Once the address is loaded we do a JUMP, not a call, to the address where the routine is located. This is an important concept. Remember that we placed the address of the CCP restart routine onto the stack. If our routine, when it has completed its function, performs a RET instruction, then program control will be passed to the restart routine. In a similar fashion, we could JUMP to an error routine that ends with a RET instruction and the address returned to from that routine would also be the RESTART routine. This assumes, of course, that our command routine has not trashed the stack, making the proper return address not the top two bytes on the stack.

Understand fully that when we enter the internal command we enter the routine without commitment to any register usage, there is nothing to be saved, and the way home is on the top of the stack. What we do in that routine is entirely our affair, provided we do not trash the stack with the address of the way home at its top. This is a profound thought, and what we have worked so hard to present in an understandable fashion. At this point in time you should be familiar enough with the assembler, and the thought processes involved in presenting this material, to begin designing your own CCP commands. We will digress for a moment and look at the commands in this basic CCP, and the way they are organized.

Recall the RCPM equate at the beginning of the primary source listing. This equate is used to prepare a system that may be open to the public, a "Remote CP/M System." This is a system whereby others may call your computer, by telephone, using a modem, and operate your equipment as if sitting at the keyboard. In all walks of life there are paraniod psychotics, and we all have to deal with people of this type. This is all fine, and part of life, until they start messing with our computers. To protect ourselves we have installed an equate that will cause the assembler not to include code that may be used by one of these disgusting creatures to trash our system. Excluding the commands noted will help to protect our system from these "twits," but cannot render us completely safe.

Unfortunately these commands are no longer available to us, once excluded, as they might be in ZCPR3, but at least no one else can use them against us. Pardon my hostility towards the lower life forms, but I find them far more trouble than they are worth. Of course, this may have a lot to do with why I am a hermit.

The command table consists of strings based upon a four character command string. The number of characters per command may be expanded to no more than 8 characters. Be sure to change the equate that defines the number of characters in a command and also to fill out the number of characters with spaces if the command name is less than the standard number of characters in your system.

Remember that once we JUMP to our command routine we have but one restriction upon us, not to trash the stack, which holds our return address. We do not really need to use the stack, and we could save it if we have to make use of the limited stack space available. The key issue here is that we can do whatever we wish, within the confines of the space allowed the CCP program itself.

The concept of named directories is possible to implement isn't it? What if the routine to log in a given directory were inserted as a command name? And, what if, when we loaded the user area number to print the command line prompt, we used the user area number to index a table of user area names? Or what if we had a hardware sequence we used all the time and installed as a CCP command? The answer to all these questions is "why not," "or is there a reason why we couldn't do all these things?" The only restriction we have upon us is the size of the CCP itself, and the reality that if we add a command, we would have to delete another command to make room for it. In ZCPR3 they did away with the internal directory program, allowing the user to select from many sorted directory programs set aside as COM files.

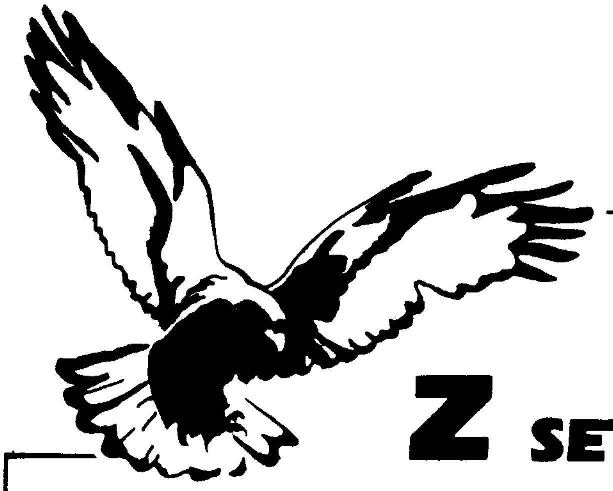
What can we do from here? Well, I know it is rude to answer a question with a question, but what are the limits of your imagination?

In the next installment of this series I will present the "normal" CCP commands and show you how I develop new CCP commands. I would hope that you would study those portions of the code I have left to your own investigation, and having done that, you will think about the way internal commands are processed. We just jump to a spot inside the CCP, having entered the name of the command in the command table, followed by its address. We also have the way home sitting on the stack, should we need it. What we may do from this point onward is only a matter of what we want to do in the space allowed for such things. But then again, there is nothing that requires the address following the command-name string to be in the CCP, it can be anywhere, now can't it. Hmmmmmm Something to think about eh? ■

* * *

See page 45 for ordering NEW-DOS disk.

* * *



Editing The CP/M Operating System

by Walter E. Pfiester

Introduction

Editing, changing and operating on your CP/M® operating system can be a real hassle using "DDT.COM® " or "XMAN.COM". These machine code editors work directly on the system tracks. It is much easier to work on these tracks if they are saved as a file first. This article will detail how to save the operating system tracks as a file and later, after editing or changing this file, placing this file back on the disk in the proper location for use as your new version of CP/M.

Saving the Operating System as a File

You will have to have "MOVCPM.COM" and "SYSGEN.COM" on the disk you want to operate on. In addition, I find it useful to have 'SD.COM" (or the enhanced version, "S.COM", "STAT.COM", and 'EDFILE.COM" on the same disk. The first two files are used to measure the size of the files. The later file is a machine code editor, in the public domain used to dump and edit, full screen, any file using HEX formats OR ASCII codes. It is not the intent of this article to delve in to the use of "EDFILE.COM'. That can be best handled by downloading the "EDFILE.DOC" file from your RCP/M library.

The first thing that you must do is to measure the size, in hexadecimal pages of memory, of your operating system. The easiest way to do that is to type:

```
A>movcpm 63 *
```

For our purposes here we want to find out how many pages of memory are required for a 63K CP/M system. If you are using a 64K system then type movcpm 64 *. For this article I will use a 63K CP/M system. As a result of the command above, what results is as follows:

```
CONSTRUCTING 63k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM63.COM"
        :
        +-------> This is the number we want!
```

The 63 above designates the operating system size. If you are going to change the size of the system for later editing, recompiling or whatever, type the size in place of 64, ie MOVCPM 55 *.

The important thing here is to write down the number after the word SAVE. What MOVCPM has done for us is automatically calculate the amount of memory required to save the CP/M image as a file. You are done using "MOVCPM.COM".

Now type "SYSGEN". In response to the question SOURCE, type a <CR>. When prompted as to the destination, type another <CR>. Your entire CP/M operating system is now in RAM, starting at 100h. To save this image as a file type "SAVE 34 CPM63.NEW". 34 is the number of hexadecimal pages that you came up with above with MOVCPM.COM above. CPM63.NEW is the file name given to save the system image as. It could just as easily be named anything! It should also be noted at this time that the operating system saved has ZCPR on it (Micro Cornucopia's version) and it was created using instructions on their disk K22.

Editing and Using the New (SYSTEM) File

Now the file "CPM63.SYS" can be edited, changed, whatever. I routinely use "EDFILE" to edit machine code. It is a full screen editor that can be used in both the ASCII and Hex fields. Besides which, it is one of the best FREE, public domain pieces of software!

Using EDFILE

As an example in using this powerful tool, I will change the logon message on cold boot to something more meaningfull than:

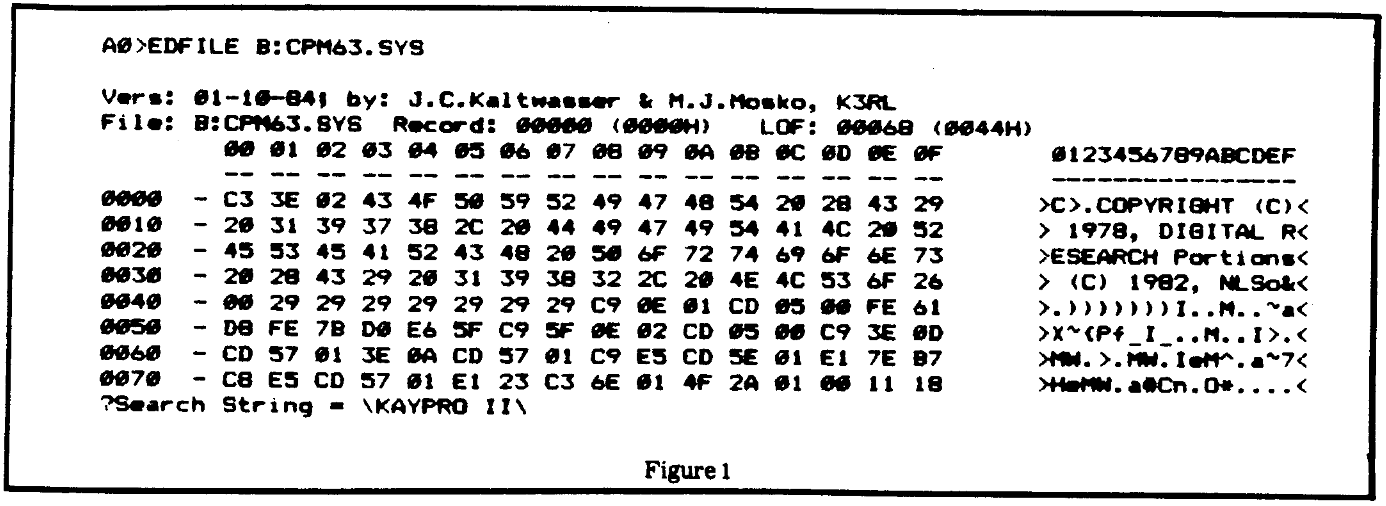
KAYPRO II 63k CP/M vers 2.2

to:

63K ZCPR-2 9/20/84
882-2.2

In addition I will show you how to edit the operating system to autoload a file on cold boot.

The file we'll be editing is CPM63.NEW created above. To edit using EDFILE, type the following: "EDFILE CPM63.NEW" What will result is shown in Figure 1.



```
AØ>EDFILE B:CPM63.SYS

Vers: Ø1-1Ø-84; by: J.C.Kaltwasser & M.J.Mosko, K3RL
File: B:CPM63.SYS  Record: ØØØØØ (ØØØØH)   LOF: ØØØ68 (ØØ44H)
         ØØ Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8 Ø9 ØA ØB ØC ØD ØE ØF     Ø123456789ABCDEF
         -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --     ----------------
ØØØØ   - C3 3E Ø2 43 4F 5Ø 59 52 49 47 48 54 2Ø 28 43 29    >C>.COPYRIGHT (C)<
ØØ1Ø   - 2Ø 31 39 37 38 2C 2Ø 44 49 47 49 54 41 4C 2Ø 52    > 1978, DIGITAL R<
ØØ2Ø   - 45 53 45 41 52 43 48 2Ø 5Ø 6F 72 74 69 6F 6E 73    >ESEARCH Portions<
ØØ3Ø   - 2Ø 28 43 29 2Ø 31 39 38 32 2C 2Ø 4E 4C 53 6F 26    > (C) 1982, NLSo&<
ØØ4Ø   - ØØ 29 29 29 29 29 29 29 C9 ØE Ø1 CD Ø5 ØØ FE 61    >.))))))) I..M..~a<
ØØ5Ø   - D8 FE 7B DØ E6 5F C9 5F ØE Ø2 CD Ø5 ØØ C9 3E ØD    >X~{Pf_I_..M..I>.<
ØØ6Ø   - CD 57 Ø1 3E ØA CD 57 Ø1 C9 E5 CD 5E Ø1 E1 7E B7    >MW.>.MW.IeM^.a~7<
ØØ7Ø   - C8 E5 CD 57 Ø1 E1 23 C3 6E Ø1 4F 2A Ø1 ØØ 11 18    >HeMW.a#Cn.O*....<
?Search String = \KAYPRO II\
```

Figure 1

At the prompt above, type an "S". EDFILE will ask you for the string to search on. What we are doing, is searching for an ASCII string (EDFILE can also search for a HEX string). Reply with a backslash " \ " and the string itself followed by another backslash, as shown above. Edfile will then locate the string as shown at address 1EE0 as show in Figure 2. This is the beginning of the log on message we are going to change.

Type <CTRL> E. This will place your cursor in the HEX field. To move about, use the arrow keys. To change to the ASCII field, type another <CTRL> E. Again, the arrow keys will move you about this field. To edit the message, type over the message already there. Remember, the carriage return and line feed can not be edited in the ASCII field, they need to be added in the HEX field. Now that you have a new log in message, let's save it to disk. First, type a <CTRL> W. This will write all your changes out to RAM and place you at EDFILE's command level. To write the changes to disk and return to CP/M, type a "Q" (for QUIT). Your are done!

You have a new file to load in place of your old CP/M operating system. Once you have completed changing this file, placing it back in the correct location on disk is very easy. Type SYSGEN CPM63.SYS <CR>. The prompt will ask you for the drive to place the system image to. Type A (or B or whatever drive you want it to go to) as shown in Figure 3.

That's all there is to it. The next time you boot off the drive you'll have your new system image on it with the new log in message.

Let's do this again, in addition to the new log in message, have the system autoload "S $AL". "S" is a modified version of "SD-92" that I have modified. The $AL command tail will lok at all user areas, and list the directory of all libraries. Now let's modify the operating system, again.

This time, use the ASCII search again to look for the word "COPYRIGHT" (see Figure 4).

Note the pattern of HEX 20's above. At location 0887 (which now contains 00h) place the number of HEX bytes the command line will contain, in this case, 5. At location 0888 start the actual command. The result is shown in Figure 5.

Save this file again, and put it in place of your system tracks. Now the next time you cold boot, the log in message will be meaningfull and you will automatically run "S $AL".

Summary

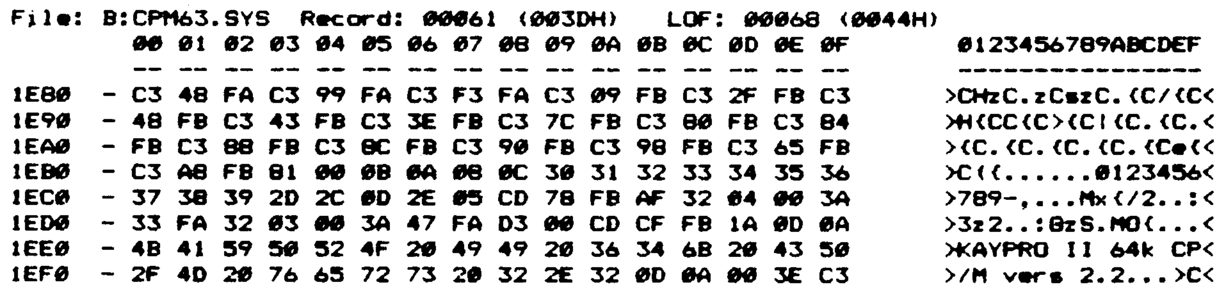
When you obtain a copy of "EDFILE.COM" make sure to obtain a copy of the accompanying .DOC file, it is absolutely essential. Another feature I have not heretofor mentioned, EDFILE.COM has HELP features built in!

I also use this method of editing my CP/M system in order to test my compiled Turbo Pascal® files under different size operating systems (saved on disk as CPM64.COM, CPM63.COM, CPM62.COM, etc.). Additionally, I have used this method to change my operating system (changed the resident CP/M command "USER" to "U", logon procedures, autoload functions, prompt messages, and different size operating systems with new logical assignments for my disk drives. Try this system and I don't think you'll ever go back to using "XAMN.COM" or "DDT.COM" to modify your .COM files again!

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.

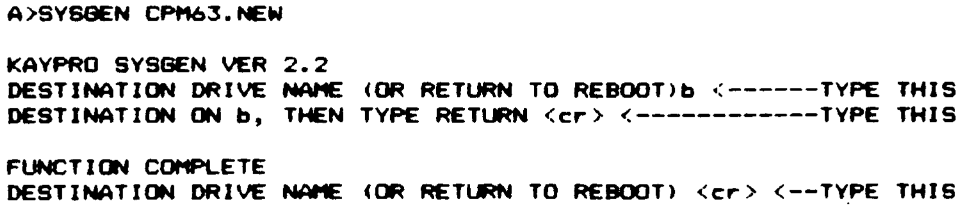
Special Disk Available

The EDFILE and SD-92 programs Walt uses are not currently part of our users' disk library, but are available from The Computer Journal in 8 inch SSSD or a number of 5.25 inch formats for $10. Ask for the "EDFILE" disk, and fully specify the format for your machine. Other selected programs will be included, depending on the disk capacity. ■



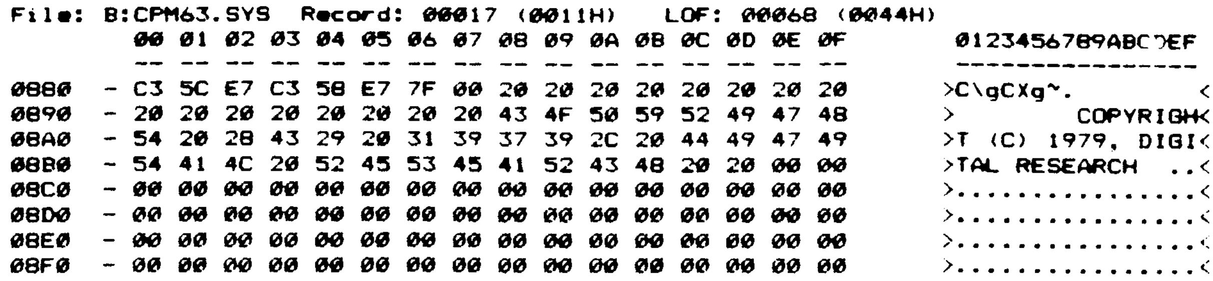
```
File: B:CPM63.SYS  Record: ØØØ61 (ØØ3DH)   LOF: ØØØ68 (ØØ44H)
         ØØ Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8 Ø9 ØA ØB ØC ØD ØE ØF       Ø123456789ABCDEF
         -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --       ----------------
1E8Ø   - C3 48 FA C3 99 FA C3 F3 FA C3 Ø9 FB C3 2F FB C3      >CHzC.zCszC.{C/{C<
1E9Ø   - 48 FB C3 43 FB C3 3E FB C3 7C FB C3 8Ø FB C3 84      >H{CC{C>{C|{C.{C.<
1EAØ   - FB C3 88 FB C3 8C FB C3 9Ø FB C3 98 FB C3 65 FB      >{C.{C.{C.{C.{Ce{<
1EBØ   - C3 A8 FB 81 ØØ ØB ØA Ø8 ØC 3Ø 31 32 33 34 35 36      >C({......Ø123456<
1ECØ   - 37 38 39 2D 2C ØD 2E Ø5 CD 78 FB AF 32 Ø4 ØØ 3A      >789-,...Mx{/2..:<
1EDØ   - 33 FA 32 Ø3 ØØ 3A 47 FA D3 ØØ CD CF FB 1A ØD ØA      >3z2..:GzS.MO{...<
1EEØ   - 4B 41 59 5Ø 52 4F 2Ø 49 49 2Ø 36 34 6B 2Ø 43 5Ø      >KAYPRO II 64k CP<
1EFØ   - 2F 4D 2Ø 76 65 72 73 2Ø 32 2E 32 ØD ØA ØØ 3E C3      >/M vers 2.2...>C<
```

Figure 2



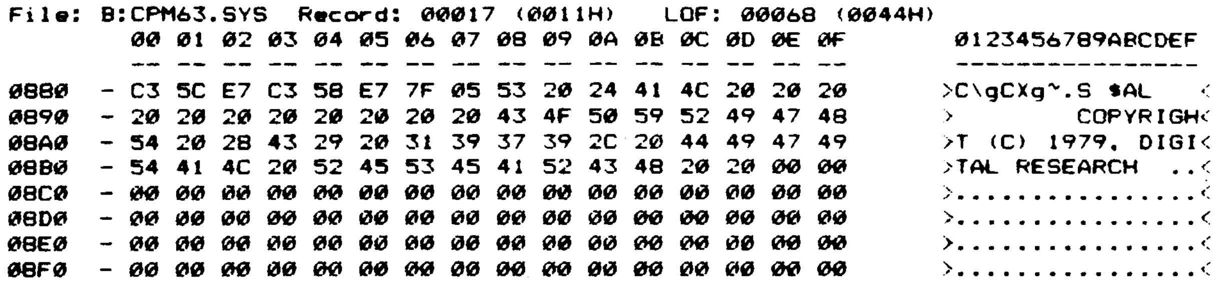
```
A>SYSGEN CPM63.NEW

KAYPRO SYSGEN VER 2.2
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)b <------TYPE THIS
DESTINATION ON b, THEN TYPE RETURN <cr> <------------TYPE THIS

FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <cr> <--TYPE THIS
```

Figure 3



```
File: B:CPM63.SYS  Record: 00017 (0011H)   LOF: 00068 (0044H)
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      0123456789ABCDEF
        -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --      ----------------
0880  - C3 5C E7 C3 58 E7 7F 00 20 20 20 20 20 20 20 20     >C\gCXg~.        <
0890  - 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48     >        COPYRIGH<
08A0  - 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49     >T (C) 1979, DIGI<
08B0  - 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00     >TAL RESEARCH  ..<
08C0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08D0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08E0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08F0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
```

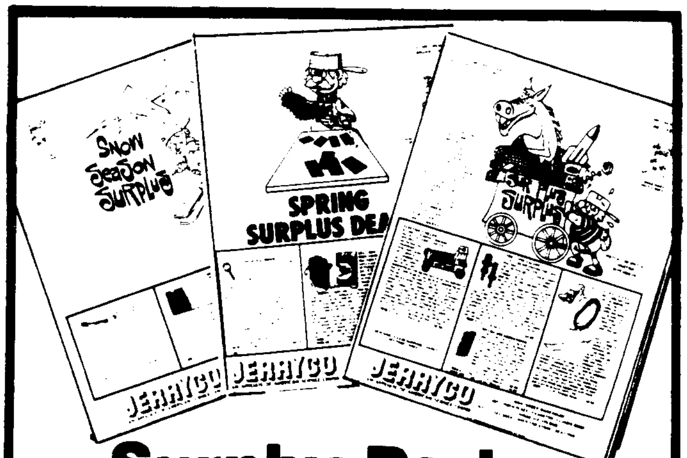
Figure 4



```
File: B:CPM63.SYS  Record: 00017 (0011H)   LOF: 00068 (0044H)
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      0123456789ABCDEF
        -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --      ----------------
0880  - C3 5C E7 C3 58 E7 7F 05 53 20 24 41 4C 20 20 20     >C\gCXg~.S $AL   <
0890  - 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48     >        COPYRIGH<
08A0  - 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49     >T (C) 1979, DIGI<
08B0  - 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00     >TAL RESEARCH  ..<
08C0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08D0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08E0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
08F0  - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     >................<
```

Figure 5

* * *



INDEXER
Turbo Pascal Program to Create Index For Almost Any Purpose
by Jerry Houston

For a while I worked my way through school as a baker on weekends, and it didn't take too many months of pounding dough for me to decide I'd rather be working with a computer. The owner of the bakery had bought an Apple II+® to operate a relatively simple database to keep track of recipes, and I was given the task of writing the program.

Well, that simple database turned out to be SO useful that my boss kept thinking of other aspects of the operation that could benefit from some computerizing, if I didn't mind writing the programs instead of busting my butt in the bakery. Let me tell you, when HE ran out of new ideas, I supplied all I could think of!

Eventually, of course, the programs were integrated into a database system that operated on multiple Apple IIe's, connected to hard-disk storage from ICE (Space Coast Systems in Florida), with a file server. I wrote routines that let the various computers share files without conflict, but that's another story for another time. The point is, when it came time to document this system of programs—both for the non-programmer users and for the benefit of any future maintenance programmer—I ended up with more than 200 pages of system manual.

For any kind of system manual to be useful, it just HAS to have an index. The index needs to be sorted alphabetically, and I figured that I would need this service often enough to make writing a program to do it worthwhile. In fact, this has become one of my most-often-used utility programs, as there seems to be no end of indexing jobs for any writer or programmer.

There are many good uses for an indexing program, besides the obvious ones mentioned. Using INDEXER, for example, you can make a sorted index of a year's worth of The Computer Journal, making it easy to find a particular article not only by name, but by any number of subject cross-references. You can use INDEXER to catalog a record collection, or nearly any other kind of collection.

The program is designed so that sorted index files can be appended, making it possible to create a simple index in one sitting, then go back and do the heavy-duty cross-referencing later. The files that are created by INDEXER are ordinary text files, with no special characteristics other than that they are sorted alphabetically. Thus, once the index file has been created, it's an easy matter to use your favorite word-processor (I can't help it - mine's still WordStar®) to format that plain-vanilla listing into a good looking index.

I have provided source code in the past for several programs, usually making it for Kaypro 4-84® . This time there are listings for Osborne Executive® (which I use at work), and for IBM-PC® and true compatibles. I'm not about to "abandon" CP/M in my writing, but I do intend to feature newer systems as well.

If anyone would like to save the time needed to type this source code, The Computer Journal can supply the program on diskette for MS/PC-DOS, Osborne Executive and Kaypro 4-84. Though they can't be expected to re-write the program specifically for additional computers, they can supply one of the above-mentioned versions on a different diskette format, and you can do your own customizing. Ordering information is at the edn of this article.

As my writing is always tutorial, besides offering what I think are useful program ideas I always go into detail about some of the features of Turbo Pascal. This time, it's additional power that came with v3.0, and in particular, command line parameters and windowing.

Running The Program

INDEXER is nearly self-documenting, because there just isn't the need for much complexity in a program like this. There are a few things you need to know, however.

When the program has been compiled as a .COM file, it can be run with a parameter supplied on the command line. That parameter will be the name of the index that's to be created, without an extension. INDEXER will append the extension .NDX to the name you choose.

If the file you've named already exists, INDEXER will open it for appending data to the end of that file, present a screen that will let you type in more entries and page numbers, and let you know how many records were supplied from the file.

If the file did not exist before, it will now—INDEXER will create the file for you and present the data-entry screen. If you try to run the program INDEXER without specifying a file name at all, there's a "sorry about that..." message printed and you're returned to the system. There just isn't anything useful you can do without supplying a file name, so the program insists.

Having started successfully, you'll be at the data-entry screen. The CP/M version of the program simulates using a window for data entry, as the 8-bit version of Turbo Pascal does not support windowing (it would have to be custom-implemented for every possible terminal type). The version written for 16-bit machines makes use of a window for data input, making the program easier to write and smoother-running in the end.

Fields are defined for 40-character items, and for up to 6-character page numbers. The page numbers are represented internally as strings, not integers, making it possible to use decimal points or hyphens in the page numbers if desired. There are good reasons for doing it this way—page numbers are often best referenced according to sections or chapters, such as 1-23 or 1.23 to indicate page 23 of section 1. Also, for catalogging articles from magazines, you might want to include an issue number followed by a page number, such as 22:45 to indicate page 45 of issue 22. If you find that you need more than 6 places for page numbers, there are only a few simple and obvious changes needed in the program.

When the last entry has been made (for this session), enter one called 'END'. The program will also be looking for 'End' and 'end', so any of these can be used. Remember, of course, you won't be able to have an actual entry in your index called 'END' (or any of the above variations on that), but that shouldn't be a problem.

If you're customizing this program, it's possible to write and use a function called GetStr which will get a string from the user, but that will also return a message that a FUNCTION KEY was pressed. I'll include a listing of that function at the end of this article for anyone who's interested—it can be very useful in many programs. Here, though, it seems perfectly natural and logical just to type 'END' to end the session.

The file will be (re)sorted at this point. The program uses a 'shaker sort' that is an improvement on the simplest bubble sorts, but still simple and convenient to code. The example index that accompanies this article took about 5 seconds to sort on an XT-compatible. If you've added entries to an existing index file, they will be integrated into the file in sorted order. Then the finished file is written to disk, with the name you used and an extension of .NDX.

The records are written to the index file with the entries separated from the page numbers by commas. This has been most useful whenever I've used the program, but that - of course - could easily be modified. You might prefer a hyphen, semicolon, or just spaces as a separator.

The resulting file is a standard text file (file of characters) in which each line is terminated by a carriage-return. It's really easy to edit the index file with a word processor, separating the entries into blocks of records that start with the same letter.

The Program Source Code

Since the two versions are nearly the same, I'll provide a listing and a detailed description of the MS/PC-DOS version. I'm sure that readers intending to modify the Osborne version for other CP/M computers will find it an easy job, and there just aren't enough differences to justify printing two separate source listings.

Anyone with an '84-or-later Kaypro should refer to page 5 of issue #21 of The Computer Journal for a review of the constants that can be used for special screen attributes, such as reverse video, underline, etc.

We should begin with the Main Logic, which is the last part of a Turbo Pascal program, and comes between the 'Begin' and the 'End' that has a period after it. Some of the instructions given in the main logic refer to functions and procedures that are supplied as part of Pascal, and others refer to functions and procedures that we have defined earlier in the program. That's why the main logic section comes last—to be used, a variable, function, or procedure must already be 'known' to the compiler.

The first statement in the main logic shows what to do if ParamCount is zero—in other words, if no filename parameter was supplied with the program name when it was run. Indexer will either append new information to the end of an existing file, or it will create a brand new file if needed. But there's just nothing it can do without a file at all, so this block of code prints an error message, makes an unflattering sound, and returns the user to DOS.

Incidentally, this first statement is a compound statement, as it contains a block of code starting with a 'Begin' and finishing with an 'End;'. Whenever the syntax of Pascal calls for a statement, several statements can be included as need, provided they are formed into a block as shown.

The second statement assigns the value of the filename given with the program to a variable called 'STR' and tacks on the extension '.NDX'. This is how the parameters that are added at the command line can be accessed within your program. ParamCount is a reserved variable that will contain the number of parameters that followed the program name on the command line, and ParamStr() is a reserved string array that contains the actual parameters as typed. If ParamCount is zero, then no parameters were provided for the program.

The variable 'Index' is used throughout the program to keep track of the number of entries that have been made. It is initialized to zero before an attempt is made to read an existing index file with the procedure 'ReadFile'. If a previous version of the file exists, then 'Index' gets incremented as each record is read into memory. Thereafter, 'Index' is incremented as new entries are supplied by the user.

That happens in the procedure called 'GetEntries', after which a window statement resets the default window to a full screen. A 'Please WAIT...' message is displayed while the array of entries and page numbers is sorted in the procedure 'ShakerSort', then a message verifies how many records now exist in the index file.

Pressing <RETurn> then executes the procedure 'WriteFile', which ends it all.

Operating the original version of this program on a CP/M computer with floppy disk drives took long enough at each I/O step that the 'progress' or 'status' messages were displayed on the screen long enough to be useful. Changing to a 16-bit computer with a hard drive made such a speed difference that I inserted a number of statements that say 'Delay(1000)', just to keep a message on the screen for a full second. Even if all the message says is 'Please WAIT... Sorting File', it's rude for it to flash by so fast that the user can't read it. These delays have absolutely nothing to do with the processing that goes on in the program, and they may all be removed to speed things up.

Example of Index

For the unconvinced, here's an example of an index that I made in just a few minutes. I indexed a copy of The Computer Journal that was handy, so the subject could be something of interest that's available to all the readers. This example is reasonably thorough, but as you read through it you'll think of many other entries that could be used to cross-index even further. The idea is to find each important topic or concept, then enter it as many times as you can think of ways to refer to it. Then when you want to look up that subject in the future, there will be that many chances of finding it where you look first.

Naturally, the first entries to make are the titles of the various articles, so you can find in which issue and on what page a particular one is printed. Then additional entries are made on a page-by-page basis. Since the index file can be started at one time and re-used many times afterwards, there's no need to try to index a whole issue in one sitting, or to index it thoroughly the first time through. If the object IS to catalog several issues of a publication, the file can be appended with each issue that comes out.

Listing 1

```
Program INDEXER(Input,Output,Results);
{
Version for MS/PC-DOS computers, requires Turbo Pascal v3.0 or higher.
Program to produce an INDEX of entries.  Written by J. Houston, 5/85.
Originally used to create an alphabetic listing of terms used in the
documentation for the Ener-G Foods, Inc. Database System, designed and
written by the author.

Provides an array up to 1000 string entries, with corresponding page
numbers.  Will sort those entries, then write a text file to disk that
can be edited with WordStar to produce the necessary index pages.
}

LABEL        Finish;

TYPE         Result =
                record
                   AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
                end;

CONST        Count              :   Integer = 0;

VAR          iret               :   Result;
             Entry              :   Array[1..1000] of String[40];
             Page               :   Array[1..1000] of String[6];
             Comparison         :   String[40];
             TempEntry          :   String[40];
             Str                :   String[40];
             TempPage           :   String[6];
             Number             :   Integer;
             En,
             Pair,
             Left,
             Right,
             Index              :   Byte;
             Results            :   Text;
             FileFound          :   Boolean;
             Letter             :   Char;

Procedure Curson;                   (Cursor on)
Begin
     iret.AX := $0100;
     iret.CX := $0C0D;
     intr($10,iret);
End;

Procedure Cursoff;                  (cursor off)
Begin
    iret.AX := $0100;
    iret.CX := $0F00;
    intr($10,iret);
End;
Procedure HiInt;                    (High intensity)
Begin
   TextBackground(0);
   TextColor(15);
End;

Procedure LoInt;                    (Low intensity)
Begin
   TextBackground(0);
   TextColor(7);
End;

Procedure RevVideo;                 (Reverse Video)
Begin
   TextBackground(15);
   TextColor(0);
End;
```

```
Procedure SoundUp;                         (Sound effect ascending)
VAR   i : Integer;
Begin
   i := 800;
   While(i < 8000) do
      Begin
         Sound(i);
         i := i + 5;
      End;
   Nosound;
End;

Procedure SoundDown;                       (Sound effect decending)
VAR   i : Integer;
Begin
   i := 8000;
   While(i > 800) do
      Begin
         Sound(i);
         i := i - 10;
      End;
   Nosound;
End;

Procedure SoundError;                      (Sound effect error)
Begin
   Sound(200);
   Delay(500);
   Sound(80);
   Delay(1000);
   Nosound;
End;

Procedure TestSwap;                       (Swap routine for Shaker Sort)
Begin
   If (Entry[Pair] > Entry[Pair + 1]) then

      Begin
         TempEntry        := Entry[Pair];
         Entry[Pair]      := Entry[Pair + 1];
         Entry[Pair + 1]  := TempEntry;
         TempPage         := Page[Pair];
         Page[Pair]       := Page[Pair + 1];
         Page[Pair + 1]   := TempPage;
      End;
End;


Procedure ShakerSort;                      (Sorts index file before storing)
Begin
   Left := 1;
   Right := Index - 1;
   Repeat
      For Pair := Left to Right Do TestSwap;
      Right := Right - 1;
      For Pair := Right Downto Left Do Testswap;
      Left := Left + 1;
      Until (Left > Right);
End;


Procedure PageT[illegible];
Begin
   GOTOXY(1,1);
   Write('INDEX/SORT Program - Type "END" as Last Entry, then <RET> to Sort and
Write File');
   GOTOXY(15,3);
   RevVideo;
   Write('              ENTRY              ');
   GOTOXY(58,3);
   Write('   PAGE    ');
   HiInt;
End;
```

```
Procedure GetEntries;               (Main Data Entry Procedure)
Label Loop, Continue;
Begin
   ClrScr;
   Cursoff;
   PageTitle;
   If FileFound then
      Begin
         GotoXY(1,25);
         LoInt;
         Write('          Number of Previous Records from File = ',Index);
      End;
   Index := Index + 1;
   HiInt;
   Window(1,6,80,22);
   GOTOXY(10,1);
Loop:
   LoInt;
   Gotoxy(5, WhereY);
   Write(Index:4,' |
   Write('|        |');                                        !        ');
   HiInt;
   CursOn;
   GOTOXY(11,WhereY);
   Read(Entry[Index]);
   If ((Entry[Index] = 'END') or (Entry[Index] = 'End')
         or (Entry[Index] = 'end')) then Goto Continue;
   GOTOXY(61,WhereY);
   Readln(Page[Index]);
   CursOff;
   Index := Index + 1;
   Goto Loop;
Continue:
   Index := Index - 1;
   CursOn;
End;


Procedure Writefile;                (Writes index file to disk)
Begin
   ClrScr;
   Write('Please wait ....          WRITING FILE');
   Delay(1000);
   Assign(Results,Str);
   Rewrite(Results);
   For Count := 1 to Index Do Writeln(Results,Entry[Count],',  ',Page[Count]);
   Close(Results);
   GOTOXY(54,1);
   Write('... Finished writing file.');
   GOTOXY(1,20);
End;


Procedure ReadFile;                 (Reads index file from disk if exists)
Label Skip;
Begin
   Count := 0;
   ClrScr;
   Write('Please wait ....          CHECKING FOR FILE');
   Delay(1000);
   Assign(Results,STR);
($I-)
   Reset(Results);
($I+)
   FileFound := (IOresult = 0);
   If Not FileFound then
      Begin
         GOTOXY(30,5);
         Write('NEW FILE');
         Delay(1000);
         Goto Skip;
      End;
   While (Not EOF(Results)) Do

      Begin
         Letter := ' ';
         Count := Count + 1;
         Entry[Count] := '';
         While (Letter <> ',') Do
            Begin
               Read(Results,Letter);
               If (Letter <> ',') then Entry[Count] := Entry[Count] + Letter;
            End;
         Read(Results,Letter);
         Readln(Results,Page[Count]);
      End;
   Close(Results);
   Index := Count;
Skip:
End;

( ==================== Main Program Logic Starts Here ==================== )

Begin
   If ParamCount = 0 then           (if no file name, there's a problem)
      Begin
         ClrScr;
         Writeln('No FILE name given, ABORTING...');
         SoundError;
         Goto finish;
      End;
   STR := ParamStr(1) + '.NDX';     (file name is first parameter)
   Index := 0;
   ReadFile;
   SoundUp;
   GetEntries;
   SoundDown;
   Window(1,1,80,25);
   GOTOXY(1,25);
   Write('                    Please WAIT, Sorting File ...');
   ShakerSort;
   GOTOXY(1,25);
   Write('                    Total Records Entered = ',Index,'   (Press <RET>)');
   Read(Right);
   WriteFile;
   Write;
Finish:
   SoundDown;
   ClrScr;
End.

          ****************************************
```

An example of an index for Issue #21 of TCJ is shown below:

A
ADC Systems, 19
ADC Systems - Applications, 21
ADC-1 Data Acquisition System - Ad, 4
APROTEK 1000 - EPROM Programmer - Ad, 26
Advertiser's Index , 50
Affordable Engineering Software - Ad, 12
Alliance Computers - Ad, 44
Analog Data Acquisition and Control - Ar, 19
Analog Fundamentals, 19
Applications - ADC Systems, 21

B
BDS C Compiler - Ad, 7
BERSEARCH Information Svcs. - Ad, 33
BMON - In-Circuit Emulator - Ad, 28
Back Issues Available - Feature, 48
Basic-52 Computer/Controller, 43
Blankenship BASIC - Ad, 21
Books of Interest - Feature, 32
Build the Circuit Designer 1MPB - Article, 35

C
CD-1 MPB - Construction of, 38
CD-1 MPB - Parts Diagram, 39
CD-1 MPB - Programming, 35
CD-1 MPB Parts List, 40
CD-1 Memory Board Schematic, 41
CTM - Ad, 39
Classified Ads - Feature, 49
Clockworks - Ad, 43
Construction of CD-1 MPB, 38

D
DIP Components - Removing, 14
Data Acquisition Software - Ad, 50
Desgn & Applicat of Sm Std Components, 32

E
Echelon, Inc. - Ad, 45
Extend-50, 43
Extending Turbo Pascal - Article, 2

F
Functions - Pascal, 2

G
GEMINI Robot - Ad, 17

H
HD64180, 52
HD64180 Communications Software, 50
HISPEED.PAS - Program Listing, 25
HISPEED.PAS - Program Listing, 29
Helping Hands - Soldering Tools, 17
Higher Sampling Speed for ADC-1, 42

I
Intellicomp - S-100 Board - Ad, 18

J
JERRYCO, Inc. - Ad, 44

K
Kaypro 4-84 Screen Attributes - Table, 5

L
LOGGER.BAS - Program Listing, 26
Lawson Labs, Inc., 50
Linear Optimizer - Acme Computer Company, 42

M
MICROMOTION - Ad, 47
MMS FORTH - Ad, 22
MPB Block Diagram - Circuit Designer 1, 36
MTBASIC - Used for ADC operation, 31
MTBASIC Compiler - Article, 10
MasterFORTH - Ad, 47
Microcomputer and Interfaces - Ad, 31
Microport 32, 43
Mid-Level Languages, 1

N
NGS Forth - Ad, 20
NO LIMIT - Ad, 47
New Products - Feature, 42

P
Pascal - Variable Storage, 3
Pascal vs. BASIC, 2
Portapac, 51
Printed Circuit Boards - Unsoldering, 33
Procedures - Pascal, 2
Programmable Microcontroller - Basicon, 42
Programming the CD-1 MPB, 35
Public Domain Software Center, 39
Publisher's Statement, 1

R
RAM 80e - Ad, 43
ROMable MTBASIC, 42
Real World Computing, 19
Resolution - Analog/Digital, 20

S
S-C Macro Assembler - Ad, 34
Solder Bridge - Removing, 11
Solder Suckers, 9
Solder Wick, 11
Soldering - Theory, 8
Southern Pacific Limited - Ad, 37
Surplus Parts Resource - Ad, 44

T
TBASIC, 50
Technical Engineering BASIC, 50
Televideo TPC-I, 51
The Computer Corner - Column, 52
Turbo Pascal - Include Files, 4
Turbo Pascal - Tutorial, 2

U
Universal MAC INKER, 43
Unsoldering: The Arcane Art - Article, 8

V
Vendors - ADC Systems, 23
Vendors - Soldering Tools, Boards, 34

W
Wabash - Diskettes - Ad, 30
Write-Hand-Man - Ad, 26

Z
Z System - Ad, 45
Z-System - Ad, 13
Z80 Plus, 52
Z80ASM - Assembler - Ad, 15

For a magazine so full of technical reference material as The Computer Journal, it's well worth the 30 minutes or so that it takes to index an issue. If EACH issue is indexed as it arrives, then the same file can be used to make a master index that spans more than one issue. If my example had been part of a larger plan of that type, each page would have been entered with the issue number first, such as :

Z-System - Ad, 21-13
Z80 Plus, 21-52
Z80ASM - Assembler - Ad, 21-15

Any small errors that are made while entering the subjects and page numbers can safely be ignored, as you'll have the opportunity to make corrections when you go back with the word processor to format the file into a printable index. If you notice too late that you've flubbed an entry, just make a note of it and re-enter it correctly. When you edit the sorted file you'll be able to make the correction easily.

Summary and Conclusion

INDEXER is a program that finds use in many different applications. It can keep track of technical articles, catalog small electronic parts in numbered bins or drawers, keep track of bottles in a wine cellar, or catalog a butterfly collection in alphabetic order for easy reference.

Since the file that's produced is a standard character text file, it can easily be formatted into a printed index or used as the input file for a program to search for records and display them to the computer screen.

INDEXER could even be used to keep track of program and file names, and the number of the diskettes that they're on, for anyone who doesn't already use a system like MCAT.

GetStr Function for Keyboard Input

Finally, as advertised eariler, here's an example of the GetStr function that I use in many programs. It provides a way to obtain data in string information, and to be able to read the IBM function keys at the same time. If a function key is pressed while the string is being entered, then the string that's returned will contain only 'F1' through 'F10'to represent the function key that was pressed, regardless of what was being typed up to that point.

This means that every section of a program that gets keyboard input from the user can have a 'bail-out' key, such as F10 or a HELP key, such as F1. Even if another entry has been started, the function will still return the value of any function key that's pressed.

If there are other characters that you would like your version of GetSTr to recognize, be sure to add them to the case structure as shown. In your main program, don't forget to declare a

```
TYPE   STR=String[30] (or other length)
```

so that a string can be returned by this function to the calling logic.

Using GetStr in a program is as easy as assigning to a string variable the value of GetStr. To input a name—but be ready to leave the procedure if an F10 key is pressed, you might code a couple of statements as shown in Listing 2. ■

The source code is available on MS/PS-Dos, Kaypro 2 SSSD, or AMPRO 5.25 inch DSDD for $10 postpaid. Inquire about other formats. We are planning on having our RBBS on line in May, and these files will available on the board at that time.

(Listing 2 on page 43)

The AMPRO Little Board Column

by C. Thomas Hilton

Well, after months of learning the ins, and outs, of the AMPRO® 1A CPU card I received the newer 1B system for development. While the 1A is still available, it is available only in OEM quantities. About the only place you can get the ORIGINAL LITTLE BOARD is through some of the mail order houses. The 1A CPU does offer a cost advantage, but the 1B is far more powerful for industrial work.

The 1B CPU has the SCSI bus on card, instead of requiring an adapter option, plus an 8 bit input port. The system ROM may be expanded in capacity, and there are a number of jumper options for various system attributes. I swapped the 1A CPU card in my Series 100 with the new card, and that will be one of the projects we will discuss this issue.

Making the Change

The first thing to do is to unplug ALL the cables from the rear of the enclosure. There are some nasty voltages exposed in there, and the packaging is tight enough where it just isn't worth the risk to have the power line connected. I always feel better when I can look at the back of a chair and see both ends of the power cable. I don't lose as many readers that way.

□ Remove the four screws on the long sides of the enclosure, on the bottom of the device. They are the ones set in the slots where the top folds over the bottom plate.

□ There are three more screws on the rear face of the system, remove them as well. Put the screws in a cup so you don't lose any of them. The case metal is thin, so we have to be careful not to put the screws in too tight, when we put it back together.

□ The top just slides off toward the back of the system. Remove it now.

□ See the two drives on the right, as the system faces you? Find the "L" flange where the power supply is mounted on the left, and the 1A CPU card is mounted on the right, next to drive "A."

□ Don't try to remove the CPU mounting screws, there are two on the bottom edge as well as on the top edge. We'll have to take the drives out to get at everything. See that "P" shaped aluminum bar on the top of the drives? That holds the sliding case cover in the proper position. Take out the two screws that hold the "P" bar to the two disk drives, and save them in the cup as well.

□ Lift up the case and locate the two screws in a line under each disk drive. Remove them, but only one set at a time, starting with drive "B."

□ Mark the cables with a felt marker on all connectors. Mark the cable side and the board side of each connection. The next time you will know what goes where, but the first time, let's mark them.

□ Now then, drive "B," with all the cables removed just slides straight out the front. It sometimes jams. Just pull it straight out, but don't force it. If it gets caught, just push it back in and try again. Place the drive out of the way, but so we can tell which drive is which. Yes, it does matter which drive is which. They each have different jumper settings, though we don't need to worry about jumpers for this project. On the lower right hand side of the drive you will see several white boxes. There are numbers where the boxes are. One box is near the number "2" on the circuit card, so this is indeed drive "B."

□ Take out drive "A" in the same way we removed drive "B."

□ Fine, now see the four screws around the edges that hold the CPU card in place? Don't remove the card yet, I want to show you something. Look at the flywheel side of the drive. The holes on the CPU card and the disk drive match! The CPU card is designed to mount to a standard 5¼ inch drive. The circuit board sets right down in the recess. It adds only about a half inch to the drive's profile, though we always design for at least ¾ inch clearance from the card or flange, when it is above the card.

□ Now mark all of the cables, to be sure we get them all back in the same place, and remove the mounting screws. Be careful as the card can be damaged by static electricity, even the amount on our bodies. Just handle the card as you would a phonograph record, along the edges. Good PC designers always put the power and ground bus structures around the edges. Lay the old card out of the way, for now.

□ The new CPU comes in a brown plastic bag, with a yellow caution seal. If the seal has been broken there is no way of telling if the board is good or not. AMPRO tests the boards and seals the package before they ship them. The bag is conductive, so there is protection from static, and handling by untrained people. Now open the bag by ripping the seal, and remove the board, touching only the edges.

□ Put the new card in the machine in the same way we took the old one out. Put all the mounting screws back in, snug, but don't screw them in tight, just snug. If you tighten them down too far you may damage the card.

□ Put all the cables back on the card. We marked them so we'd know where they went. AMPRO made the new card with all the connectors in exactly the same places, nice of them to do that eh?

□ Get drive "A," reconnect the cables, and set it in place. Lift up the device and replace the two bottom screws. Don't tighten them down as we will have to adjust them for proper placement later. Don't put in the top screws yet.

□ Now replace drive "B" in the same manner, making sure the cables are all in the right places, and that the two bottom screws are not tightened fully.

□ At this point we want to assure that the drives are set properly in relation to the case front. I leave an "ease" space of about 1/8 inch in the center between the drives. This lets us use that little plastic flange on the drive bezel to position each drive on the outer edges of the case opening. Now tighten the two bottom screws to set the drive in place.

□ Locate the "P" shaped positioning bar. Mount it with the thin edge to the left, the wide portion over the drives. What this piece does is position the outer case in relation to the heavy front panel. We want to set this bar as close to the front of the enclosure as possible, and make sure there is an equal space on each side of the bar. This will keep the sliding case top properly positioned. Just to be sure you can slide the case back on, and move it back and forth, with the "P" bar loose, to see what I mean. Once positioned, tighten the bar mounting screws.

□ Slide the case back on, and replace all of the case mounting screws. BE CAREFUL! The metal is very soft and will strip easily if you try to tighten the screws too tight.

Now power up the system and assure that everything works properly.

KTERM Update

I really should get a terminal I guess, but the Kaypro® does the job until I see what I will be doing in the 16 bit developments.

The original version of KTERM, written in TURBO PASCAL® , served to interface the Kaypro 4-84, and later Kaypro versions, to the AMPRO Series 100 as a "dumb terminal." The original KTERM was suitable for use in all applications except the use of WordStar® . The overhead of the TURBO runtime library, coupled with the internal functioning of WordStar, required faster data processing by the Kayp.o. The patching of WordStar's internal delay tables did nothing to improve the situation. This CROWE assembler version of KTERM is fast enough to handle all known commercial, and public domain software applicable to either machine. This version has replaced the TURBO PASCAL version for all tasks.

The primary difference to the user, other than an 8K reduction in program size, is that the "^_" character, (control/underline), is now the escape character. The null character is used, in the DOS direct console function request, to indicate a lack of keyboard activity.

```
LISTING ONE
-----------------------------------------------------------------
;
;                               Hermit Software's
;                       Crowe Assembler Source Code File
;
;                         (c) 1985 C. Thomas Hilton
;       Program: KTERM.COM        (WordStar Version)
;       Function:
;               The original version of KTERM, written in TURBO PASCAL, was
;       found to be incompatible with WordStar. This version of KTERM, written
;       in the CROWE assembler, corrects the defects of the original version.
;
;       Author: C. Thomas Hilton 12/16/85
;
;       Index:  KTERM.COM
;               KTERM.CRW
;
        LIST
        TITLE 'KTERM.CRW '
        NLIST
;
;       SIO A Equates
;
TXRDY   EQU     4               ;SIO transmitter ready mask
RXRDY   EQU     1               ;SIO data receiver ready mask
DATPRT  EQU     4               ;SIO data port
STATUS  EQU     6               ;SIO status and control register
DTRWT   EQU     68H             ;SIO data terminal busy data mask
DTRRDY  EQU     ØE8H            ;SIO data terminal ready data mask
;
;       System Equates
;
DOS     EQU     5               ;BDOS jump vector
        ORG     1ØØH
;
;       Initialize The Kaypro "Serial Data Port" to:
;
;       8 bit data word
;       1 stop bit
;       Even Parity
;
;       As well as preserving the functioning of the standard keyboard channel
;
        LD      A,18H
        OUT     (STATUS),A
        LD      A,1
        OUT     (STATUS),A
        LD      A,Ø
        OUT     (STATUS),A
        LD      A,3
        OUT     (STATUS),A
        LD      A,ØE1H
        OUT     (STATUS),A
        LD      A,4
        OUT     (STATUS),A
        LD      A,47H
        OUT     (STATUS),A
        LD      A,5
        OUT     (STATUS),A
        LD      A,ØE8H
        OUT     (STATUS),A
;
        LD      SP,STACK+16             ;define a local stack for DOS calls
;
;       Print Herald On Kaypro Screen
;
        LD      DE,HERALD
        LD      BC,9
        CALL    DOS
        JR      MAIN                    ;jump over herald print string
;
HERALD: BYTE    26                      ;clear the screen
        DATA    '                       '
        DATA    'Hermit Software''s KTERM  Dumb Terminal'
        BYTE    1Ø ! BYTE 13 ! BYTE 1Ø
        DATA    '                       '
        DATA    '             Press ^_ To Abort'
        BYTE    13 ! BYTE 1Ø ! BYTE 1Ø ! BYTE 1Ø ! BYTE '$'
;
;       Enter Primary Function Loop
;
;       The only operator control is ^_ as an abort character. A null character
;        cannot be used as it is trapped by the keyboard input scan and used to
;        indicate that no character is ready from the keyboard.
;
MAIN:
        IN      A,(STATUS)      ;get the receiver status byte
        AND     RXRDY           ;mask to see if an character received
        JR      NZ,RDCHR        ;nonzero means yes
        LD      BC,6            ;else check console status for keyboard
        LD      DE,ØFFH         ;character using function 6, FFH is input flag
        CALL    DOS             ;make the call
        OR      A               ;zero value (null) means no character ready
        JR      NZ,SNDCHR       ;nonzero means we have one to send, no echo
        JR      MAIN            ;repeat loop
;
;       Print An Abort Message Before Passing Control Back To Kaypro
;
ABORT:  LD      DE,BYE
        LD      BC,9
        CALL    DOS
        JP      Ø               ;return to operating system
;
BYE:    BYTE    13 ! BYTE 1Ø ! BYTE 1Ø
        DATA    'Exiting KTERM'
        BYTE    '$'
;
;       Primary Output To Computer Vector
;
;       Also checks for an abort character, which never gets sent to computer
;        if detected. ^_ is abort character.
;
SNDCHR:
        CP      1FH             ;is character ^_; or escape character?
        JR      Z,ABORT         ;yes, abort
        PUSH    AF              ;else save it
GCHR1:  IN      A,(STATUS)      ;is the transmitter empty so we can send?
        AND     TXRDY           ;mask status byte to find out
        JR      Z,GCHR1         ;zero means transmitter is busy sending
        POP     AF              ;yes so get character back
        OUT     (DATPRT),A      ;and send it
        JR      MAIN            ;hurry back to the main loop
```

```
;
;          If We Get Here We Have A Character From Computer
;
RDCHR:
           LD        A,5              ;select the proper SIO register for controls
           OUT       (STATUS),A
           LD        A,DTRWT          ;set the wait flag
           OUT       (STATUS),A
           IN        A,(DATPRT)       ;get and print incoming
           LD        E,A              ;character
           LD        BC,2             ;through standard DOS call so Kaypro can keep
           CALL      DOS              ;track of cursor position then
           LD        A,5              ;clear the wait flag to tell computer we are
           OUT       (6),A            ;ready for next character
           LD        A,DTRRDY
           OUT       (6),A
           JR        MAIN             ;and hurry back to main loop
;
STACK:     RSRV      16               ;a 16 byte local stack
           END
```

The source code for this simple program is shown in listing one. If you are following my "NEW-DOS" project, you should have a copy of the CROWE assembler. This assembler is available from the new TCJ Public Domain Library, and upon the NEW-DOS project disk.

Designing Turnkey Systems With MENU.COM

While awaiting the manual for ZCPR3, I have been working with the ZCPR3 "shell" MENU.COM. This is a very powerful system level menu program. To support AMPRO users I have developed a number of "turnkey systems," based upon this utility.

All the turnkey systems I am working on are prepared using the stock AMPRO ZCPR3 CP/M operating system, revision C, AMPRO Part Number A60101, on a dual floppy Series 100 using the 1B Little Board. MENU 3.6 is a true ZCPR3 utility, and you must assure that you have either properly installed the utility, or are using the AMPRO system noted. AMPRO has installed a number of utilities on the revision "C" system disk. All that is required is a good text editor to create powerful turnkey systems.

If you acquire one of the systems I have configured, several steps are required to make the disk operational.

Using your normal system working diskette, format and sysgen a blank disk. No operating system is supplied on any TKS or TCJ User Library Disk. Transfer the entire contents of this disk onto your freshly prepared disk with your favorite copy utility. PLACE NO OTHER FILES UPON THIS DISK AT THIS TIME!

With the new working copy of the TKS disk in drive "A" go to the CP/M prompt and enter:

CRC <RETURN>

your TKS disk will then report upon the status of all files by checking the CRC value of each file now on the disk, with the CRC values of the files at the time the disk was created. If you have made an accurate copy, all CRC values will match. In the case of a mismatch, recopy the file until all values do match.

Now transfer your terminal attribute file, (MYTERM.Z3T), to the working disk. It is assumed that your sysgen image has properly set your operating environment to match your terminal. Transfer also, at this time, the following AMPRO Utilities from your system working disk onto the TKS disk:

LDR.COM
MENU.COM

Using the AMPRO CONFIG utility, modify the Autocommand, (option 5) to "TKSTART." Now place the TKS disk in the system drive, and reset the computer. If the menu does not properly appear on your screen, repeat all of the above steps. If this fails, consult your AMPRO User Manual.

When the menu appears, press the "H" key to enter the on-line help system. The first screen of this system will describe any other programs you may need to complete the system. While I have attempted to build turnkey systems using only public domain software, there are cases where you will have to install a commercial program of choice, such as a text editor. The help sysstem will describe what is needed, and how to install the program of concern.

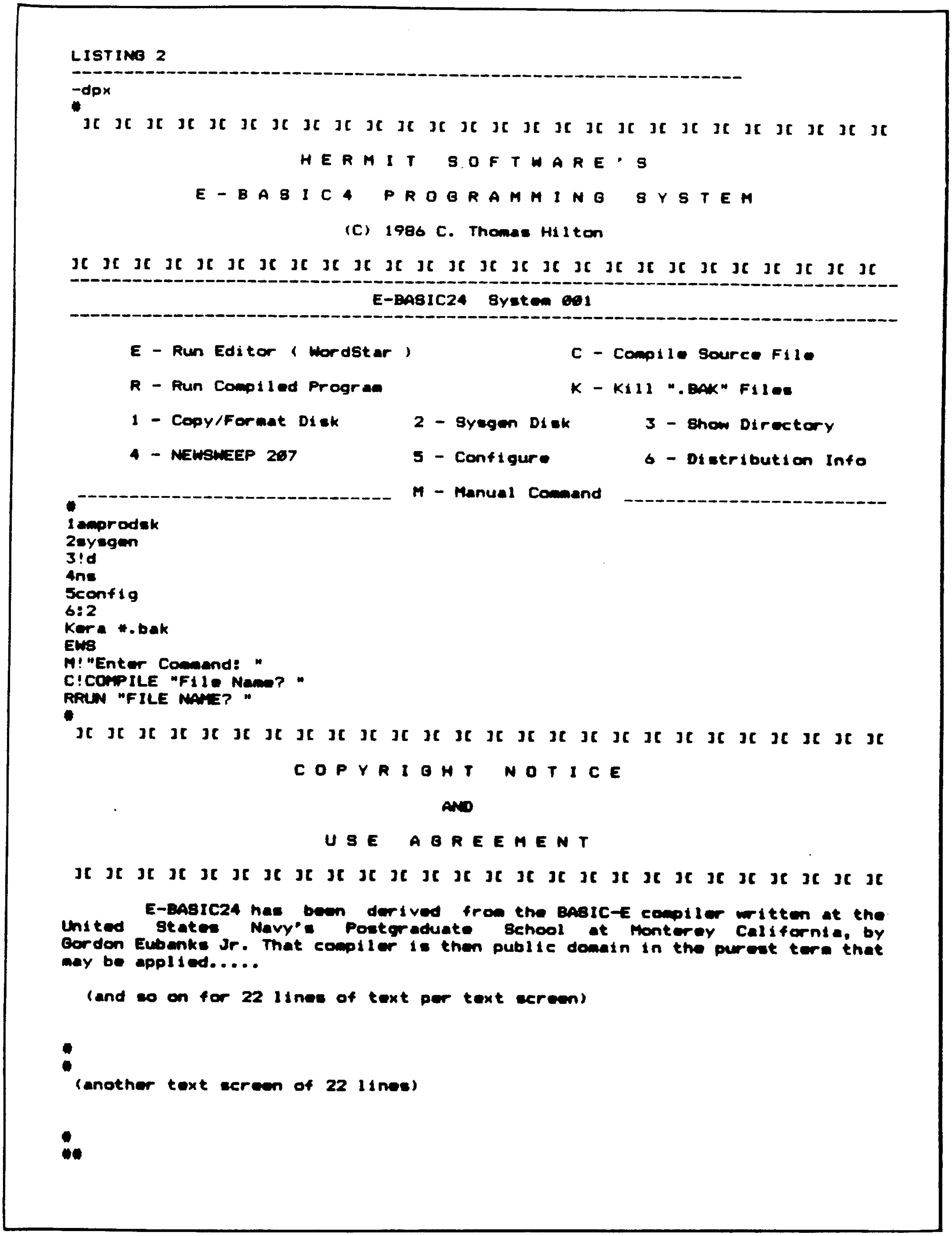
If you have a public domain program that will do the job of any commercial program required for a TKS disk, send it to me and I will modify the TKS disks concerned to use your program. In this way you can help to do your part to support the people who are supporting you.

Developing your own turnkey system is not difficult. A simple turnkey system is shown in Listing 2.

All MENU files, (MENU.MNU) begin with a statement beginning with a dash. This defines the global options that are to be used in the operation of the MENU system. In the example the file begins with:

```
-dpx
#
 )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )( )(

     The statement "-dpx" states that MENU is to display the
screen which follows, scroll the screen before displaying it, and
allow a  C access to the CP/M command line.  The options
available at the opening global declaration are:

     d -  Display text screen which follows.  A text screen is 22
lines of text.

     p - Scroll the screen before displaying text screen.

     x - Allow exiting MENU to the CP/M command line with a ^C,
(control-C).

     c - Display command lines as the program runs.  This is a
debugging option that is used when you get a "Structure Error."


     A text screen is bracketed with the pound sign, ("#").  The
proper format for a text screen is:

# (appearing as first character)

               22 lines of text

# (and another pound sign as a terminator)
```

The text screen may contain nearly any character, except, of course, the pound sign. If your terminal is capable of reduced video or other highlighting, and you have installed the command in your 'MYTERM.Z3T" attribute file, you may insert the command for these attributes in the text file. These attributes are indicated by the characters:



```
LISTING 2
-----------------------------------------------------------------
-dpx
#
 ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C
                         H E R M I T   S O F T W A R E ' S
                E - B A S I C 4   P R O G R A M M I N G   S Y S T E M
                           (C) 1986 C. Thomas Hilton
]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C
--------------------------------------------------------------------------------
                             E-BASIC24  System ØØ1
--------------------------------------------------------------------------------

      E - Run Editor ( WordStar )                C - Compile Source File

      R - Run Compiled Program                   K - Kill ".BAK" Files

      1 - Copy/Format Disk      2 - Sysgen Disk       3 - Show Directory

      4 - NEWSWEEP 2Ø7          5 - Configure         6 - Distribution Info

 ------------------------------ M - Manual Command ------------------------
#
1amprodsk
2sysgen
3!d
4ns
5config
6:2
Kera *.bak
EWS
M!"Enter Command: "
C!COMPILE "File Name? "
RRUN "FILE NAME? "
#
 ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C
                        C O P Y R I G H T   N O T I C E
                                      AND
                            U S E   A G R E E M E N T
 ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C ]C

        E-BASIC24 has  been  derived  from the BASIC-E compiler written at the
United   States   Navy's   Postgraduate   School   at  Monterey  California, by
Gordon Eubanks Jr. That compiler is then public domain in the purest term that
may be applied.....

  (and so on for 22 lines of text per text screen)


#
#
 (another text screen of 22 lines)


#
##
```

```
^A - turn on attribute
^B - turn off attribute
```

These attributes are entered into the file as normal text. WordStar users are familiar with the technique of inserting printer commands in their text. To implement these codes the WordStar user would enter:

```
^PA - turn on attribute
^PB - turn off attribute
```

How control characters are entered into a text file with your favorite text editor may be found in that program's manual.

The text screen of 22 characters is closed with another pound sign. What follows this, 'end of screen marker," should be the commands to be acted upon by single key press commands, or additional menu screens. The example below presents examples of MENU command lines.

```
1amprodsk
2sysgen
3!d
4ns
5config
6:2
Kera *.bak
EWS
M!'Enter Command: "
C!COMPILE "File Name? "
RRUN "FILE NAME? "
```

Each line following the screen terminator represents a valid ZCPR3 command line. Commands are entered as they would be entered at the "A0>" prompt, but must have the reference indicator noted in the first text screen. Additionally, functional options may be inserted into the command structure.

In command lines "1" and "2" no MENU command line options are present. When the "1" key is pressed MENU will load and run AMPRODSK, passing control of the system to that program. When the program has terminated system control will be passed back to MENU. This is the Ampro definition of what a "shell" program does.

In MENU command "2" the SYSGEN program would be run in a manner similar to that of the AMPRODSK program. This simplistic SYSGEN command line may be expanded for users of SYSGEN, Version 3. To place the system image on drive "B," from drive "A" the command line could read:

```
2sysgen /a,b,,
```

Command line "3" however, has a MENU option installed. When an exclamation follows a reference character, MENU will pause after completing the command sequence for a key press from the operator and display the message:

```
MENU 3.6 - Strike Any Key -
```

The use of this option is desirable when displaying a directory, for instance. Without it the directory screen produced by DIR.COM would be displayed, and then immediately overwritten by the MENU text screen. By implementing the option:

```
3!d
```

(which is a sorted directory program similar to DIR.COM) the directory of the current drive would be displayed, and the system would wait to redisplay the menu until a key was pressed. Again, the command line may be expanded to include more complex directory calls.

```
3!DIR *.BAK;ERA *.BAK;DIR;DIR B:*.BAK;ERA
B:*.BAK;DIR B:*.*
```

Any command line that may be entered from the ZCPR3 command line may be entered from the MENU command line. Command lines may be up to 200 characters, or the line length determined by your implementation of ZCPR3.

As noted below there are other command structures as well.

```
6:2
Kera *.bak
EWS
M!'Enter Command: "
C!COMPILE "File Name? "
RRUN "FILE NAME? "
```

In the example "6:2" the pressing of key "6" will cause the second text screen to be displayed. The colon, in this application means, "GOTO SCREEN NUMBER TWO." In the example you will note that the command lines for text screen one are terminated by another pound sign. MENU assumes the text which follows to be another valid text screen. The user must assure that this is the case. At the end of that text screen another set of command lines may be installed, up to 255 text screens and command line sections!!! With this much power complete online help systems are very easy to develop.

In the next set of examples file names and supplemental command lines are input to the MENU command line.

```
M!"Enter Command: "
C!COMPILE "File Name? "
RRUN "FILE NAME? "
```

The first of these examples cause the system to wait for a key press when they have completed their task, as indicated by the second character being an exclamation mark. The command:

```
M!'Enter Command: '
```

allows a manual command to be entered as if at the ZCPR3 "A0>" prompt. The general rule is that the string following a prompt enclosed in quotes will fill the vacant portion of a command line structure as shown in the next examples.

```
C!COMPILE "File Name? "
RRUN "FILE NAME? "
```

In my version of E-BASIC the compiler is named COMPILE.COM. Not an original name, but a fitting one. Too many people want to complicate simple tasks. The compiler needs the name of the text file it is to compile. MENU will ask for this file name and place it into the command line in proper format for the compile program to receive. In my E-BASIC the runtime interpreter is named, "RUN," of course. The same concept used to COMPILE a program is applied to the runtime interpreter.

This insertion of a file name into the command line is a truly universal concept. The following concept is also available, as might be applied to my Modified Crowe Assembler:

```
X!CROWE "File Name? ".BBZ
```

which would insert the file name and add the assembly toggles as if the command were entered at the command line as:

```
A: SYSTEM>CROWE CCP.BBZ
```

This series of text screens and command lines may go on nearly forever, and whose format is nearly "free form." The basic structure of the MENU file is as follows:

```
-dp (global display options)
#
text screen number one
```

```
#
command lines
#
text screen (if text it is screen 2)
#
text screens or command lines for text screen 2
#
more screens and/or command lines
#
## (end of file marker)
```

The only real defect I have found in using this shell is the limitation of a single key command. While this is a normal, and very powerful system, I have found several cases where a two character command would have made life a great deal easier.

There are other MENU commands which reference ZCPR3 command formats, which are loosely referred to as "variables," by the sparse documentation in the Ampro User Manual.

$D - Current Disk
$U - Current User
$FN - A reference to a ZCPR3 system file name and type number "N"
$Nn - A reference to a ZCPR3 system file name number "n"
$Tn - A reference to a ZCPR3 system file type number "n"
$$ - Place a $ in a command line

I may be retarded, but I have never been able to make the '$" reference, or "variable" work on the basic 60K ZCPR3 implementation.

In multiple text screen files there are also commands for moving about within the menu system.

<RETURN> - Rewrites the current menu on the screen
^C - Control C exits to system if "x" option is installed in opening option list, (-dpx).
* - Jump to first text screen, (main menu).
< or , - Jump to previous menu. This command allows the comma to be shifted, ('<") or entered unshifted as these characters are generally found on the same key.
> or . - Jump to next menu. This command allows the period to be shifted, ('>') or entered unshifted as these characters are generally found on the same key.
$ - Jump to the "System Menu," where a password is supposed to be required to use this function. The Ampro manual documentation makes no mention as to how to implement this function, and I have yet to make it work on my system.

In the near future we will take a close look at ZRDOS. I have used this operating system to implement the AVS Voice computing system for the visually impaired. There are several critical tests I wish to perform upon ZRDOS before rendering a final evaluation. When I am given a product to review, or evaluate, I use it for a reasonable amount of time, testing it fully. I feel these are things that should not be rushed.

This brief introduction to MENU.COM should, however, allow you to begin designing your own turn key systems.

AMPRO User Disk Library

By way of introducing the library, allow me to give a bit of the history of this library, and credit those who have made it possible. In the beginning there was TCJ. I met TCJ while working to develop technologies for the disabled concerning the Ampro Series 100 microcomputers. I needed a place to distribute my wares, most of which are released into the public domain. TCJ needed a user disk library. TCJ donated the disks, I contacted the librarian of the Charlottesville Virginia Kaypro User Group, (CKUG), who filled the disks with what they had. And it came to pass that no one cared much for the way most PD

THE LIBRARY CORE

IMPORTANT NOTE: To pack the largest number of files upon each disk like files are placed in "Libraries." The best approach to a Library File, if you are unfamiliar with them is to order, "TKS.001, THE LIBRARY CARD" which is the system I use to handle the library. NULU.COM is the library utility used to create all libraries, and is included upon each disk containing a library file. Read it's "HELP" screens, and always be sure to copy your TCJ disk onto a working file.

To assure you get a good copy enter:

```
A0>CRC<RETURN>
```

and the disk will report any files which did not copy properly. In the case of a bad file, just recopy it.

-- UTILITY.001 386K --

```
-UTIL    .001   0k | CRC      .COM   4k | CRCKLIST.CRC   2k | DESPOOL .LBR  34k
DIF/SSED.LBR  30k | DIRLABEL.LBR  16k | DISKPRAM.LBR  12k | EX14    .LBR  40k
FIND    .LBR  10k | FIX      .LBR  30k | HELP20  .LBR  42k | MAKE20  .LBR  16k
NSWP200 .LBR  40k | NULU     .COM  16k | PASSWORD.LBR  12k | SD-ASM  .LBR   4k
SUPERSUB.LBR  30k | UNERA19 .LBR  18k | UNSPOOL .LBR  24k | XERA    .LBR   6k
```

NEW SWEEP file management, SUBMIT type utilities, a complete HELP system, UNERAse for recovering erased files, PASSWORDing files, print spoolers, MAKE which moves files to other user areas without making copies, a neat disk label program, and more.

-- UTILITY.002 380K --

```
-UTIL    .002   0k | BU       .LBR  40k | CRC      .COM   4k | CRCKLIST.CRC   2k
DU      .LBR  72k | FINDBAD .LBR  32k | LB-TIME .LBR  94k | NULU    .COM  16k
QWIKKEY .LBR  18k | SCRAMBLE.LBR   8k | SD-78   .LBR  60k | SQUEEZE .LBR  28k
TYPE109 .LBR   6k
```

Massive Disk Utility programs, file Back-Up utilities, Bad sector lockout, a pseudo-realtime clock BIOS enhancement for the 1A Little Board, QWIK Key, a macro keyboard program, file security by scambling a text file, all the sorted directories you could ever need, and a better TYPE command.

-- UTILITY.003 366K --

```
-UTIL    .003   0k | CPMPOWER.LBR  10k | CRC      .COM   4k | CRCKLIST.CRC   2k
D-DUMP  .LBR  16k | LOCK     .LBR  48k | M-DUMP11.LBR  14k | NULU    .COM  16k
SD-88   .LBR  74k | SETDRU   .LBR  16k | SORT-DIR.LBR  90k | SWAP    .LBR  12k
UNERASE .LBR  12k | UNSPOOL .LBR  12k | VAX-VMS .LBR  30k | WYSE    .LBR  10k
```

General disk utilities, a library to allow a CP/M micro to communicate with a VAX mainframe computer, and WYSE terminal utility.

-- UTILITY.004 372K --

```
-UTIL    .004   0k | CATS     .LBR  40k | COMPARE .COM   2k | CRC      .COM   4k
CRCKLIST.CRC   2k | LABEL    .COM   2k | LIBRARY .LBR 158k | NULU    .COM  16k
NULU    .DOC  26k | NULU11   .LBR  54k | SORT     .LBR  32k | WHATSNEW.COM   2k
WHATSNEW.DOC   2k | YANC     .COM  32k
```

Library and Catalogue systems. Just about everything you'd ever need for file archiving.

-- UTILITY.005 376K --

```
-UTIL    .005   0k | ALLOC    .COM   2k | BAN      .AQM   4k | BISHOW  .COM   2k
BS      .COM   2k | CALL-CPM.COM  16k | CALL-CPM.DOC   6k | CHEK    .COM   4k
CLEANUP .COM   2k | CLEANUP .DOC   2k | CRC      .COM   4k | CRC     .DOC   4k
CRCKLIST.CRC   4k | DISPLAY .COM   4k | DISPLAY .DOC   2k | DUPUSR  .LBR   4k
DUSR    .LBR   4k | EP       .COM   2k | INDEX    .COM   4k | INDEX   .DOC   2k
L       .COM   2k | L        .DOC   6k | MKEY     .COM   2k | MKEY    .WPF   6k
NSWEEP  .DOC  24k | NSWP207 .COM  12k | NULU     .COM  16k | PROBE   .LBR  38k
READ    .COM   2k | RENAME  .COM   4k | RENAME   .DOC   2k | RPIP    .COM   8k
RPIP    .DOC   2k | RPNC     .COM  34k | RPNC     .DOC  20k | RX800SET .COM  2k
SAP     .COM   2k | SCROLL  .COM   2k | SCROLL  .DOC   2k | SD-92   .LBR   8k
STATUS  .COM   2k | SYNONYM .COM   2k | TPA     .COM   2k | TURN    .COM   6k
TYPEL31 .LBR  18k | TYPER   .LBR  38k | UNLOAD  .COM   2k | UNSQUEEZ.LBR   8k
VLIST   .COM   2k | VLIST   .DOC   2k | WASH    .COM   4k | WORM    .ASM   8k
XSUB    .COM   2k | XTYPE   .COM  10k | XTYPE   .DOC   2k
```

A wide selection of text and file management tools with SYNONYM, the "ALIAS" of standard CP/M. I use RPIP and INDEX constantly and recommend them. INDEX will even print directories on disk jackets!

-- UTILITY.006 382K --

```
-UTIL    .006   0k | CRC      .COM   4k | CRCKLIST.CRC   2k | NULU     .COM  16k
REPLACE .LBR  26k | SOURCE  .LBR 334k
```

This is the disk for those who just can't leave well enough alone. SOURCE is filled with utility program source listings that are on my "one of these days" list.

-- UTILITY.007 380K --

```
-UTIL    .007   0k | ALPHA    .COM   2k | CALENDAR.COM   6k | CALENDAR.DOC   2k
CRC     .COM   4k | CRCKLIST.CRC   2k | EDIT     .COM   2k | EDIT     .DOC  14k
FTNOTE13.LBR  30k | PROWRITE.LBR   8k | TESTPRGS.LBR  62k | TTYPE    .LBR  66k
TYPEWRIT.BQS   2k | TYPEWRIT.COM   4k | TYPEWRIT.COM   4k | TYPWRYT .COM   4k
WCOUNT  .COM   6k | WINDEX  .COM   2k | WINDEX  .DOC   6k | WORDSTAR.LBR  76k
WS0     .COM   2k | WS0/WS15.INF   2k | WS15    .COM   2k | WS330PT .DOC  12k
WSCMNDS .DOC  10k | WSDOCON .LBR   6k | WSLSTKP .AQM   2k | WSSOFTCR.LBR   6k
XREFT   .LBR  36k
```

Here we have tools for WordStar users, a typing tutor, and neat tools to keep handy in the office. Indexes and footnotes the way WS should have done them.

-- UTILITY.008 386K --

```
-UTIL    .008   0k | CAPITALS.LBR  10k | CRC      .COM   4k | CRCKLIST.CRC   2k
EDIT712 .COM  10k | EDIT712 .DOC   2k | EOFSTRIP.COM   6k | EPSON80 .LBR   6k
FOOD    .COM  32k | GOTHIC  .LBR  12k | MATH     .LBR  22k | MORSE   .COM   8k
MURPHY  .QQQ  22k | PROPOR  .LBR   4k | PSCREEN .COM   2k | PSCREEN .DOC   2k
PSCREEN .HEX   2k | PSCREEN .ZSM   4k | PW       .SYM   2k | PWBIND  .COM  22k
PWCONFIG.COM  34k | REFIND2 .LBR  32k | REFSRT   .COM   8k | TAXES-83.LBR 114k
WRTBIND .COM  24k
```

PERFECT WRITER tools, a simple minded database, tax preparation helps, and printer utilities. Of course MURPHY'S LAWS for every office employee.

-- COMMUNICATIONS.001 348K --

```
-COMMUN .001   0k | AMP-COM .LBR 326k | CRC      .COM   4k | CRCKLIST.CRC   2k
NULU    .COM  16k
```

Communications programs, and utilities for the Little Boards.

-- COMMUNICATIONS.002 366K --

```
-COMMUN .002   0k | CRC      .COM   4k | CRCKLIST.CRC   2k | MEX12    .LBR 146k
MEX12A  .LBR 122k | NULU     .COM  16k | XMODEM93.ASM  76k
```

Source files for MEX, (Modem Executive), and XMODEM, all you need to do it right, do it your way. If you are going to telecommute, or telecompute, you need these tools.

-- COMMUNICATIONS.003 300K --

```
-COMMUN .003   0k | AMODEM  .COM  18k | AMODEM  .DOC  28k | CRC      .COM   4k
CRCKLIST.CRC   2k | M7K     .LBR  22k | MDM712  .COM  18k | MDM712   .LBR  56k
MDM721  .LBR  32k | MDM740  .LBR  44k | MODEM   .DOC  34k | MODEM7   .LBR  16k
NULU    .COM  16k | PROTOCOL.DOC   8k | RBBS-USE.DOC  10k
```

Modem 7 programs and source code. For personal compterists needing access to the TCJ Board, here are your tools, and how to use them.

-- COMMUNICATIONS.ØØ4 374K --

```
-COMMUN .ØØ4    Øk | BBLIST   .       6k | BYE-KPRO.INF   4k | BYE-KPRO.LBR  72k
CONNECTG.COM   12k | CONNECTH.LBR   2Øk | CONNECTJ.LBR  24k | CONNECTL.LBR  4Øk
CRC     .COM    4k | CRCKLIST.CRC     2k | KP2/4BYE.ASM  28k | KPROTERM.DOC   2k
MDM716  .LBR 122k | NULU     .COM    16k | RBBS     .XQX  22k
```

A list of RBBS systems, how to use them some BBS utilitiy source files designed for the Kaypro, but can be modified for the Ampro, and some good files for working with mainframes, as used at the University of Virginia.

-- MBASIC.ØØ1 35ØK --

NOTE: MBASIC is a commercial BASIC sold by Microsoft. As this is a commercial BASIC we are offering these listings "as is" and will not support them further. The only BASICs that will be supported are languages we may freely distribute, without cost.

```
-MBASIC .ØØ1    Øk | ADVENTUR.LBR 172k | ART     .LBR 136k | CHASE   .LBR  2Øk
CRC     .COM    4k | CRCKLIST.CRC   2k | NULU    .COM  16k
```

This disk has the ORIGINAL ADVENTURE game, recently upgraded, and is the most cursed of all adventure games. 55Ø point version. ART.LBR has some neat computer pinups, that may be rated PG-13.

-- MBASIC.ØØ2 384K --

```
-MBASIC .ØØ2    Øk | BIO     .BAS    4k | CASTLE  .BAS  26k | CRC     .COM   4k
CRCKLIST.CRC    2k | GAMES1  .LBR   74k | GAMES2  .LBR 152k | GAMES3  .LBR  6Øk
LIFE    .BAS    2k | LUNAR1  .BAS    4k | NULU    .COM  16k | PACMAN  .COM  18k
PACMAN  .DOC    2k | STARWAR2.HQH   1Øk | STARWARS.BQS  1Øk
```

Games are not a priority and will not be supported. Good adventure games compatible with a voice output environment I will review, but they had better be good.

-- MABSIC.ØØ3 372K --

```
-MBASIC .ØØ3    Øk | AREACODE.LBR   2Øk | BACARRAT.BAS   4k | BIO     .COM  14k
BLKJCK  .BAS    6k | BLKJK   .BAS    8k | BOGGLE  .BAS   2k | CASTLE  .BAS  26k
CASTLE  .DOC   16k | CATCHUM .COM   3Øk | CHESS   .COM  26k | CRC     .COM   4k
CRCKLIST.CRC    2k | DUCK    .BAS    8k | HORSE   .BAS   6k | HURKLE  .BAS   2k
NULU    .COM   16k | SWORDS  .BAS   12k | UTIL-BAS.LBR 17Øk
```

More games. Those MBASIC programs of redeeming social value are in UTIL-BAS.

-- MBASIC.ØØ4 372L --

```
-MBASIC .ØØ4    Øk | CRC     .COM    4k | CRCKLIST.CRC   2k | GAMES4  .LBR 184k
INVASION.BAS   16k | MYSTERY .LBR   96k | NULU    .COM  16k | STARTREK.COM  54k
```

Mystery is a fair "Who Dunnit," but no good enough to capture my interest for very long.

-- LANGUAGES.ØØ1 21ØK --

```
-LANG   .ØØ1    Øk | ADDR    .CBL    2k | ALGOLM  .LBR  36k | CBL1    .CBL   2k
CBL2    .CBL    4k | CINTERP .COM   14k | COBOL   .COM  14k | COBOL   .DOC  38k
CRC     .COM    4k | CRCKLIST.CRC    2k | DEMO    .CBL   2k | EXEC    .COM   8k
FORTH   .LBR   36k | NULU    .COM   16k | PART2   .COM  14k | PILOT   .LBR  16k
SEQ     .CBL    2k
```

Fairly decent renditions of COBOL, PILOT, and FORTH, though the documentation on all these libraries was most frustrating. Anyone have any documentation on PILOT?

-- LANGUAGES.ØØ2 362K --

```
-LANG   .ØØ2    Øk | CRC     .COM    4k | CRCKLIST.CRC   2k | JPASCAL1.LBR 162k
JPASCAL2.LBR 178k | NULU    .COM   16k
```

This is the full $398 JRT, (not JDR!) Pascal system with manual. This is a very good Pascal implement and had a rather large following before the company went out of business. It is a FULL Pascal system, more suited for school use than TURBO, but not quite UCSD Pascal. I think it is worth the price several times over!! One of the best public domain Pascal systems!

-- LANGUAGES.ØØ3 314K --

```
-LANG   .ØØ3    Øk | CRC     .COM    4k | CRCKLIST.CRC   2k | CROWE   .COM  1Øk
CROWEA  .TXT   2Øk | CROWEB  .TXT   16k | CROWEC  .TXT  2Øk | README  .ZQØ   8k
XCCP    .COM    4k | XCCP    .DOC   14k | XCCP    .HLP   2k | XCCP-ERA.COM   2k
XCCP-REN.COM    2k | Z8Ø     .COM    8k | Z8Ø     .ZQØ   6k | Z8ØA    .ZQØ   6k
Z8ØB    .ZQØ   2Øk | Z8ØC1   .ZQØ   12k | Z8ØC2   .ZQØ  18k | Z8ØD    .ZQØ  14k
Z8ØE    .ZQØ   1Øk | Z8ØF    .ZQØ   1Øk | ZCPR    .ASM  54k | ZCPR    .DOC   6k
ZCPR    .MAN   46k
```

This package contains the original CP/M version of the CROWE assembler, as well as my modified version used for teaching assembly language, with source code. There are also several CCP programs for you to study along with the NEW-DOS PROJECT DISKS.

-- LANGUAGES.ØØ4 37ØK --

```
-LANG   .ØØ4    Øk | AN      .COM   12k | AZM     .COM  14k | AZM     .DOC  16k
CRC     .COM    4k | CRCKLIST.CRC    2k | CVERT   .AZM  38k | CVERT   .COM   6k
DASM    .COM   1Øk | DASM    .MQC   48k | DASMZLG .MQC  12k | DISK    .DOC  38k
PROG1   .AZM    2k | PROG2   .AZM    2k | PROG3   .AZM   2k | PROG4   .AZM   6k
RESOURCE.COM    6k | TDASM   .COM   1Øk | TEXT    .ØØ1  18k | TEXT    .ØØ2  2Øk
TEXT    .ØØ3   14k | TEXT    .ØØ4    6k | TEXT    .ØØ5  2Øk | TEXT    .ØØ6  1Øk
UNSQ    .COM    4k | UVMAC   .COM   18k | XLATE2  .COM   6k | XREFPRN2.COM   4k
Z8ØMR   .COM   14k | ZZSOURCE.COM    8k
```

This package has my AZM Macro Z-8Ø assembler as well as its parent version, 8Ø8Ø to Z-8Ø source code converters, a decent manual, and several Z-8Ø disassemblers. This is the tool kit I use when not working with the CROWE. Everything you need to program your Z-8Ø Little Board. My favourite programmer's tool kit! UVMAC is a Macro assembler written in "c" for which I have been meaning to write some documentation for. It assembles directly into a "COM" file. A nice program, but needs a manual badly.

-- LANGUAGES.ØØ5 356K --

```
-LANG   .ØØ5    Øk | BASIC/5 .ASM   76k | BASIC/5 .COM   8k | BASIC/5 .DOC   8k
CRC     .COM    4k | CRCKLIST.CRC    2k | LLLBASIC.ASM  86k | LLLBASIC.COM   8k
LLLBASIC.DOC    4k | LLLFP   .ASM   62k | LLLMON  .ASM   6k | STARTREK.DOC   2k
STARTREK.TBI    6k | TEST    .FIL    2k | TINIDISK.ASM  64k | TINIDISK.COM   4k
TINIDISK.DOC   14k
```

If you have ever wondered about how the BASIC language works, here is the disk for you. I started a book based upon these "tiny" and "mini" BASIC languages, but got side-tracked to other projects. These are 8Ø8Ø code files and are good for writing industrial process controllers. If there is enough interest in these languages, I may do a series on designing your own BASIC language.

-- LANGUAGES.ØØ6 28ØK --

```
-LANG   .ØØ6    Øk | AN      .COM   12k | AN      .DOC   2k | CRC     .COM   4k
CRCKLIST.CRC    2k | NULU    .COM   16k | SMALL-C .LBR 148k | SMALL-C1.LBR  96k
```

This is the "SMALL C" language folks. If you have the slightest interest in the use of the "c" programming system on a micro, this is the package to get. The documentation isn't very good, but there are several books written about this version of Small c. Take this one home even if you do not have an interest in "c" just now.

-- LANGUAGES.ØØ7 368K --

```
-LANG   .ØØ7    Øk | CRC     .COM    4k | NULU    .COM  16k | TARBEL  .LBR 348k
```

This is a full featured BASIC in 8Ø8Ø code. I also had a book started on this language until I found the REAL source for E-BASIC. Study this language after you have learned how the smaller BASIC's work. The IN/OUT module in this language is out where you can work on it, but make sure you save all flags and registers when writing your I/O code!! You have been warned!

★ ★ ★ See page 45 for ordering Library Disks. ★ ★ ★

disk libraries were organized. A disk was never released until it became filled with random selections. We felt that a library should be constructed like a library, and began sifting through the disks. I had in mind selecting files that could used on the AMPRO VOICE COMPUTING SYSTEM, for the blind, a time consuming task. And the publisher spake unto The Hermit, "Get The Damn Listing Done!" and The Hermit did comply, for when the publisher speaks, we poor galley slaves must listen.

I have not gone through each and every disk, nor fully tested each and every file. However, these disks are not just filled and forgotten. Each disk is in a category, and the files it contains will always be the same. As an example, 'UTILITY.004" will always contain library and catalogue programs. As long as there is room on the disk old versions will be placed in archive files, being replaced by new releases of the same class and type. New will replace old continuously. As I work my way through the library I will refine, sort, and adapt the disks. This is a living library, not a collection of files upon disks. When we get our BBS system on line then you will only have to refer to the category of disks and enter, "WHATSNEW" at the system prompt to see what files have been added, and what files have been replaced with newer versions. What follows is our 'BASIC LIBRARY CORE," the place were we begin building our library.

The files are tested using a Kaypro 4-84 as a terminal, and some files may, at this time, be intended for use upon only a Kaypro computer. As I work my way through the files I will attempt to convert the files for generic terminal use, and Ampro machines. These disks are sent to TCJ in Ampro DSDD 48 tpi format. Other disk formats may be had by contacting TCJ with your format request.

Special Library Services

Due to distribution restrictions between TCJ and myself, a duplicate library will be established at TCJ and at "THE HERMIT'S CAVE." Each issue of TCJ will have a listing showing the current state of the library.

In addition to the normal copying, shipping, and handling charge for these disks, you may request a custom disk containing only the files you have a need for. I will provide this service based upon demand, though all shipments will be routed through TCJ.

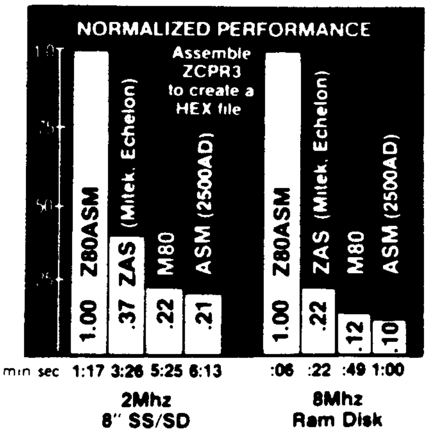
FREE Software, 2 for 1

Often I will issue a call for special PD software. These calls will be for materials I require for my Public Domain selections for the disabilities, which have a special listing. If you send in a disk containing the software we are looking for, with source code and means of assembling or compiling same, (i.e. so I can modify it for voice or other use), I will send you two free library disks of your choice.

If you convert a program to AMPRO format, update it, or just want to help your fellow computerists by donating a program of your own, I will send you two files for every usable file you send me. By usable I mean a file we do not already have, or one that is legally in the public domain, and can be used without cost by everyone. When donating original works be sure to send me a statement that you authorize its release into the public domain. Where there are duplicate submissions of a program type, preference will be given to works created with the languages contained in our library, (that we can distribute), or TURBO PASCAL.

The costs of this special exchange program will be born by myself, not TCJ, whose involvement is concerned only with the distribution of library material.

At this moment in time we need text editors, and good BASIC language source codes, so root around the disk graveyard, or just get creative! ■



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.

Z Corner

(Continued from page 5)

You can access any drive or user by specifying the Drive and User in the command. For example if you are in user area A5, and you want to transfer MYFILE.TXT which is in A3 to B6, and your file transfer program (the ZCPR3 file transfer program similar to PIP is MCOPY) is in A0, you would enter the command:

```
MCOPY B6:A3:MYFILE.TXT
```

It would take a lot of work and duplicating files in the various user areas to accomplish this from CP/M 2.2.

Once you become accustomed to how easy it is to access the user areas under ZCRP3, you can separate different types of files into different areas to make the directories less crowded. This is especially important when using a hard disk so that you don't have to look at directories with several hundred entries.

Conclusions

This first article only touched lightly on one aspect of ZCPR3, and I intend to continue with more detailed information on its utilities in the next issue. Please send in your questions, tips, and comments to share with others. ■

INDEXER Listing 2

(Continued from page 32)

```
        Name := GetStr;
        If Name = 'F10' then exit;
Here's the function:

Function GetStr : Str;
VAR    Stat   : Boolean;
       Key    : Char;
       Return : Str;
       Count  : Byte;
       StrVal : String[2];
Begin
   Return := '';
   Key := #0;
   Count := 0;
   Stat := False;
   Repeat
      If Keypressed then Stat := True;
      If Stat then
      Begin
         Read(kbd,Key);
         Case Key of
            #27 : Begin
                     If KeyPressed then Read(kbd,Key);
                        If Key In[#59..#68] then
                           Begin
                              Str(Ord(Key)-58,StrVAl);
                              GetStr := 'F' + StrVal;
                           Exit;
                        End
                     Else
                        Begin
                           GetStr := Key;
                           Exit;
                        End;
                  End;
            #13 : Begin
                     GetStr := Return;
                     Exit;
                  End;
            #8  : If (Count > 0) then
                     Begin
                        Return := Copy(Return,1,Length(Return) - 1);
                        gotoxy(wherex-1,wherey);
                        Write(' ');
                        gotoxy(wherex-1,wherey);
                        Count := Count - 1;
                     End;
         '.',' ',
         '-','+',
         'a'..'z',    ( <- put here all characters to be accepted normally)
         'A'..'Z',
         '0'..'9',
         ':'       : Begin
                        Return := Return + Key;
                        Count := Count + 1;
                        Write(Key);

                       End;
            End; (case)
         Stat := False;
      End; (if)
      Until(Key = #13);
   End;(function)
```

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 7 (Issue #11):
- *Putting the CP/M IOBYTE To Work*
- *Write Your Own Threaded Language, Part Three: Secondary Words*
- *Interfacing Tips and Troubles: Noise Problems, Part Two*
- *Build a 68008 CPU Board For the S-100 Bus*
- *Writing and Evaluating Documentation*
- *Electronic Dial Indicator: A Reader Design Project*

Sold Out

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*

Disks Available

The following disks are available for $10 each postpaid. Other formats may be available on special request, and many of these files will be available on our RBBS which should be on-line sometime in May.

CP/M-80
TCJ User Disk Library Core-AMPRO 5¼" DSDD; NEW-DOS CPP source file, Crowe Assembler, and text files-AMPRO 5¼" DSDD; Houston's ISAM and INDEXER source files-AMPRO 5¼" DSDD or Kaypro 2 SSSD.

MD-DOS
Houston's ISAM and INDEXER source files.

NEW-DOS

An AMPRO format 5¼ DSDD with the files for the Crowe assembler and the CCP is available from The Computer Journal for $10 postpaid. Inquire about other formats.

Additional disks with the BDOS and BIOS portions of NEW-DOS will be made available when these portions are published. Anyone making extentions to NEW-DOS or implementing it for other systems are urged to send their material to TCJ so that it can be shared with others.

Back issues are $3.25 each in the U.S., Canada, and Mexico. All other countries are $4.75 each for surface mail. All funds must be in U.S. dollars on a U.S. bank.

ORDER FORM

Enter my subscription to The Computer Journal for the period checked. Payment in U.S. funds is enclosed.

□ one year (6 issues) $14 in U.S. □ two years (12 issues) $24 in U.S. □ new subscription □ renewal

Disk name and # ______________________________ Amount __________

Back Issues #'s ______________________________ @$3.25 ea. __________

□ Check enclosed □ VISA □ MasterCard Card # ______________________________

Expiration date __________ Signature ______________________________

Name ______________________________

Address ______________________________

City ______________________________ State __________ ZIP __________

The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

Computer Corner

(Continued from page 51)

1991 NEWS

A newspaper report from the year 1991...

"IBM LAW FIRM COSTING MILLIONS... Recent reports indicate that law firms representing IBM have made millions in the last few years. Between IBM's attack on unauthorized copying and the attacks against IBM's products, several law firms have collected multimillion dollar fees. Most of the action started in the mid eighties, and have involved unfulfilled promises, changes in product lines, and delays in production of new items. " An attorney in the actions was heard commenting on the cases; "it all started when management thought they had squeezed all the little guys out, could force users of old systems to buy new products, they started increasing cost excessively, figuring they had a captive audience that would buy at any price. Unfortunately they paid little attention to quality and the promises of their sales force..."

In the same paper.....

"APPLE SELLS 1 BILLIONTH APPLE II.... A celebration was held today as the 1 Billionth Apple II came off the assembly line. Although the company has improved its sales of Macintosh like units, the major seller is still the Apple II. The mid eighties introduction of Macintosh support cards for the II has guaranteed the product well into the mid nineties."

Also..

"AT&T DROPS COMPUTER LINES... AT&T dropped their seventh computer line and vows to stay out of computer manufacturing. Despite the company's standing with UNIX, it never has been able to penetrate the user arena and sell complete systems. UNIX which was once considered the software of the future has since become almost unheard of. Some vendors have hidden UNIX under other operating systems, and thus became user friendly, a problem with UNIX from the start....."

Lastly....

"Small Business Administration Announces Largest Mom & Pop Business..Computer Support...... The administration today released figures which indicate that more 'mom & pop' businesses have developed to replace the large number of business failures of computer retail outlets. During the late eighties, a number of businesses went broke and left millions of products unsupported and opened up a considerable market for small operations. These businesses are one and two person operations providing either walk in or over the phone lines service. Support operations are restricted to usually one product and may have national distribution......."

A short note....

"USERS GROUP MAKES 1000th USER DISK. The original SIG/M users group which has supported CP/M80 systems from the beginning in 1975 has just released their one thousandth user disk. Despite some industry spokes people who have been saying CP/M is dead, this group has continued to provide new and free public domain software for CP/M80 users......"

What Will Come True

All these quick little teasers should have brought to mind, either fears or joys. Most of us users, especially those working directly or indirectly in the business, have most likely considered some of these ideas of late. With the industry in major upheaval and shakeouts happening daily, all the stories above are possible but only time can tell. Keeping these stories in mind let's now talk about buying that ideal system. For most people these days an ideal system is simply one that will last at least five more years. About the only thing we know for certain to happen over the next five years is the cost/performance ratio will continue to improve.

The cost performance ratio is a relationship between hardware ability and your cost. We have seen the cost of memory chips decrease as their size has increased. Competition has also dropped prices, but as we have seen lately, competition can also cause business failures. Those controlling manufacturing will add some stability to the pricing system in the next five years by limiting the production capacity. Too many companies got burned over the last year and will not allow that to happen again. I would predict that items in the future will reach a stable price shortly after industry wide acceptance. Items that have been replaced will go up in price as production is shifted to new items. For the user this translates to more memory for the same price as long as you continue to upgrade. System pricing will more likely be governed by use, than by hardware design.

The current level of hardware development is now mostly a "cook book" operation. The great GURU's of hardware have been replaced by VLSI designs and anybody can build a system. Manufacturing skills are becoming more automated, and automatic testers help companies crank out thousands of working units a day. To see this you need look no further than the PC clones which are so numerous. Prices are as low as $175 for a 128K PC compatible mother board. Complete systems can be built for less than $700, which has changed the whole way of looking at personal computers. If cost is the determining factor, then last year's business systems have become this year's home systems.

For many years the $1,000 limit marked the dividing line between personal and business systems. As most of us know, that figure is for a basic system and will likely double before a full system is put together. This last year has produced many new systems at this cost level (thanks to over production), and most of the advancements have been extra features for the same dollar. Some vendors have really jumped ahead by using newer chips (Mac) but found hard times instead of prosperity. This further supports the statement that it is software and not hardware that sells a system, and this will be true for the next five years.

New Software

When software has been developed for the home cost has always been low, typically under $100, often $19.95. For business use the opposite has been the rule, with prices typically $500 a package, and the really good stuff $1,000 and up. In truth the sale price seldom has any bearing on the actual cost of production or development. There are so many different ways that software can be created that price can not be used as a guideline for finding good software. One software company may steal a program, another may spend several years working on it, yet the market may only be willing to pay based on how slick the sales campaign is. Some of the best software in many cases has been free, mainly public domain programs.

To look at the future of software, we must see the changes that have happened over the past. Public domain software has been around from the beginning and will most likely stay for some time. These programs usually start as one person's work and are then put in public domain with the source code. The availability of the source code allows others to fix bugs, make improvements, customize the system and generally make the programs a mature product (some software houses could learn from this). As so many of us have found out, many software houses do almost as much to keep you from using the program as they do to let it be used properly. This fear of "pirating" the program has become such a problem that many programs now exist that cannot be used or backed up, which would make their cost excessive even if they were free.

If we exclude cost as a factor in determining software selection (along with hardware design), user friendliness and adaptability become major consideration. As with public domain programs, I rate availability of source code, support, and documentation all just as equally important. I have recently used a mouse type system (considered friendliest) and found some advantages, mostly however I have found short comings. This friendliness also costs in memory load times and space. Current hardware cost makes only loading time of any concern, as most new systems are over 500K of memory. This all makes me feel that the newer software will help the new user considerably (friendlier), but provide a less useful systems (mice are slow) to the old "hacker".

Choosing That New System

While considering what to cover, I got involved recently with an old and a new system. The old system was a collection of parts purchased at swap meets. The new is an Atari 520 ST with all the windows and mouse functions (user friendly). Many issues back I had talked about system integration, mostly about S-100 but also about systems in general. Choosing one's ideal system really is not much different from integrating a system. All the same concerns, fears, and problems are present in any type of system selection.

My friend's reason for selecting the used S-100 components was two fold. His hardware knowledge is low and he would like to improve it by some hands on "tinkering". He has not had much experience with CP/M 80 because his other systems have been TRS-80 and the COCO. We purchased a 520ST at work to use in emulating more expensive color units, with a look at some extra benefits. Neither of the two projects are intended to be "IDEAL SYSTEMS" but each can give us some help in meeting that goal.

In choosing a system, the first order of business is to decide why you need it, then what it will do, how often must it do it, who will support it, and where you will go for support. In both cases the reason for having the computer system started the process. Most often those who already have a system can loose track of "why" they got it, while new users may not be sure at all. My friend's choice of improving his hardware knowledge is a sound choice, while using the 520 for emulation may not be. Let's see how clarifiying this first choice effects us for the future.

Hardware will be changing some in the future, mostly with more VLSI circuits. Someone wanting to become better at hardware, either in computers or items using microprocessors, should then get involved with the newer components. My friend will learn about the S-100 standard, but will fall short on new technology. His system has cards which are already 7 and 8 years old, it can be upgraded, but he will likely end up replacing everything, including the noisy bus. So here is the first stumbling block—to learn fundamentals or to learn the newest items, an S-100 is a good choice, but make sure the equipment is capable of newer or different functions. Other choices might be the Heathkit Education system, which can be expanded into a full MSDOS system. He could also use his current COCO as a basis for building up small projects and interfaces, all using old and new components.

While software could be part of the emulation decisions, it is actually the hardware that makes a 520 a good choice for graphic emulation. The 520 is based on the 68000 by Motorola, and is my choice of the CPU of the next five years. A very important design advantage of the ST is the off-loading of functions from the CPU. The Mac suffers from an over worked CPU (it does everything) while the ST has three other VLSI (CPU's) devices. The 520's RGB output will drive many different types of monitors and this will allow us to upgrade monitors while still keeping the same interface hardware and software. Currently the color terminals are all one unit, with a rather poor monitor built in. The terminals are left on 24 hours a day and so we can expect to replace at least one monitor a year. The hardware part makes this a good choice, except that someone must write an emulation program for these terminals. Once done this will make upgrading faster (we hope) which is why the next stumbling block in choosing a system is "what is needed".

What Next???

The "what is next" kills many people after they bought that ideal system. I have heard too many stories about taking it home and then discovering how much extra is needed to make it run. When my friend called and said he had this S-100 stuff, I then spent 10 minutes telling him all he will need to do, just to bring the system up for the first time. To say he was stopped in his tracks a bit, would be a major understatement. It took me over 6 months to get my first system to run, and then only with the help of the original designer. A common problem of people who have used packaged systems, is their lack of familiarity with how much it took to get to that point, and worse, how much more to get to the next step. Another magazine has recently started covering the PC clone, more because they discovered how much cheaper and simpler it is to use a clone as the basis for a development system. My friend will spend months (and more money) before he has an operating system, and yet for the same or less money he could have a running system with room to expand (and build). Due to the lack of S-100 sales you will find only slight price drops (mostly memory), and fewer parts available on the used market (lots of PC parts). So if you want to tinker expect to use PC clones, this will be the situation probably, for the next five years.

The 520 is a new item, and has not been written about much yet. Our use has already hit a snag, as Atari only takes cash orders for their development support package. To write good software, and get a look at their schematics, you will need this package. Our company has a policy of buying only with purchasing orders, so at present we can not get the support package, a $300 dollar extra that is needed if you intend to do anything other than use canned software. This package also contains the C assembler and linking packages needed to write our emulation program. In the future however there will be plenty of SIG (special interest groups) support as well as books that could help with this problem. You only need to look at the speed at which the PC group has obtained well over 150 public domain disks to see how future support of any product can be enhanced by SIG's. The few 520 software packages are currently under $50, which is a big contrast to the PC. The "what is next" part of complete systems is becoming software and in the next five years will become the major selling (or breaking) point of any system. There are plenty of groups fighting for survival in software and with the Turbo Pascal success (cheap and good programs) you can expect to see plenty of under $100 software packs in the coming years.

How Often???

A common problem or question is how much use will the system get. My friend will be lucky to use the S-100 for 5 to 10 hours a week. This I feel is typical of a second system, but would it be the second system if it was a clone? A full running system, where you removed one expansion card and inserted your project card would most likely see much more overall use. However once you have established a software base, few will have the money to buy all new software for a second system. Our 520 however has a different problem with its use level—it will be on continuously. Can the 520 hardware take it? But then again could a clone? I feel that most of the current hardware, and most of the new hardware as well, will be more than adequate. It seems that most manufacturers are making everything of the same quality, while some are producing what is called industrial quality. Recently I saw IBM's version of their PC for industrial use. They said it had been beefed up some and was ready for rather harsh use. I felt that they had just made a tighter box, put in better filtering, and used a cheaper but more dust proof keyboard, all for twice their normal cost.

When looking for items that can be used continuously, the more expensive are not always better products. What you can do is separate out those components that have high usage or shorter life spans. Typically this amounts to keyboards and disk drives getting most of the abuse. The video monitor on the other hand will fail when the screen (filaments) life expectancy is exceeded. Either item can be looked at as replaceable items and should be inexpensive enough to allow frequent replacement. Cost in these cases is based on the number of suppliers, and systems using standard items (many manufacturers) will remain inexpensive. I have considered the emulation use to be very light on keyboards and drives, which is why we prefer a monitor that can be easily replaced. This will also prevent dealing with our original terminal supplier who has eliminated support for their older products.

Support

Our computer club was one of several groups asked by Digital Research Inc. if we would support CP/M. It seems that DRI is no longer supporting their products and is trying to find some alternate means. Both hardware and software companies have found out the nature of real support and especially the true cost. DRI who is no longer selling their CP/M 80 software, must still support it in some way because they have as yet to release the source code. One of our officers first comment on the idea was, " Will they give us the source code??". I ran into similar problems with the Superbrain, the source code was not available for their real BIOS, and the company would not release it even to the three support centers. This is all typical of the fears and concerns which companies have over their own software. The amount of technical support a company must supply is inversely proportional to the amount of source code they provide (free or otherwise). No code equals lots of calls. I would expect to see considerable changes in this respect over the next few years and by 1991 at least some changes in how support is handled.

Most companies have chosen to pass support to their OEMs or dealers. This idea was to make it easier and faster to get support. It has failed rather miserably due to the rather low skills level of most dealers. I have also found that many of the dealers have little more than the same manual you receive with the package. This poorly written document, usually does not answer anything but basic questions. This is especially true if you are dealing with old S-100 equipment. My friend's system is beyond support as most companies are no longer in business and the old manuals leave more questions unanswered than they answer. His support will be from other users and clubs. Our emulation project will be outside the normal use of the system, and the factory will be of little help, if at all. This all indicates that some other form of support is needed.

Where To Go For Support

To solve the major support problem will take lots of action and input from all users to effect any changes. If all clubs would respond to DRI's request by asking for the source code, they might realize that putting it in public domain frees them from any form of support. The public domain groups are already set up to handle support of software and distribution. This network of support and information distribution is far more successful and complete than any manufacturer could ever put together, what more it is free. My vision of the future contains as much hope as realistic insight that companies will put their old software and product information into public domain. However I strongly feel that expansion of the use of computers is currently hinged on this one aspect. Should companies continue to restrict their access to software and hardware details, I can only foresee continued declines in use and sales of systems. Those companies that make information available (even at a price) will on the other hand see an expansion of their use, the future is really up to them.

For hardware buffs I especially see a problem for the newer VLSI devices which may not become available to the average user. A point in mind is the Atari 520's video controller. This special chip will be needed if someone wants to create a 520 for the S-100 bus. This aspect of running one type of software in a different system is hampered by the use of non-standard VLSI circuits. However we can foresee that these devices are getting more intelligent each year and in five years it all may be a mute point. To explain this point, consider how intelligent peripherals have become. It should not take too much more time before the entire operating systems will be on the disk controller chip. Another example might be putting the entire GEM system on a single chip that takes mouse inputs and turns them into system commands (such as disk commands). This all leads to what I expect to see in 1991, a universal interface language, that allows different processors to talk among themselves and peripherals directly.

1991

When looking into my crystal ball for 1991, I see several things which one should keep in mind when purchasing equipment in 1986. We know for certain that there will be improvements in the nature and type of devices available. Their intelligence will continue to improve, making more advanced systems possible with less work on our part. I expect to see some older systems being used for intelligent interfaces to these newer items, thus keeping some parts of older investments useful, long past their design expectancy. There will be some newer chips which should lead the way to this newer technology. I expect most newer products will be based around the 68000, and one such product is the NOVIX by the Forth group. This 68K based unit runs Forth code directly and will find its way into industrial controls before the year is out.

The impact of optical disks is just beginning to come to people's attention,

Advertiser's Index

Alliance Computers	27
AMPRO Computers	14, 39
Apropos	29
BD Software	6
Bersearch	22
BV Engineering	2
C.C. Software	47
Computer Journal	44, 45
Computer Trader	23
Echelon, Inc.	24, 46
Jerryco	27
Miller Microcomputer Services	12
Next Generation Systems	11
Public Domain Software	43
Rockland Publishing	49
SLR Systems	42

and with the size and cost ratio improving it will not be long before systems based on their use start appearing. A system that I have been describing for several years now, will be based on the interactive abilities of the optical disk. I predict that by 1991 an educational system will be in most student's home. This system will most likely be built for the text book manufacturers, and sold to every family in America. The system will provide interactive and multimedia education similar to what is found in present day text books. The difference being that students will be asked questions after reading the text and should they answer a question wrong, the unit will return them to the correct place to reread the lesson. The interactive aspect of the system would allow the book makers to use video tape footage of actual history to replace pictures or descriptive text. The reason that the text book maker would be involved, is they already own and have all the material, and would only require moving it over to the new medium.

Conclusion

While writing this article I have tried to cover two views in answering the questions. Those views were what to look for in an IDEAL system and what changes to expect in the next five years. It was possible to see how your choices for a system may need some updating or more in depth research. The future system will be more powerful, but should the software not be adequate, you can waste that power. The use of older existing system will continue and may prove more useful than new ones. Newer system that shield the users from the system too heavily may in fact limit their use. The support problem will most likely get worse before help becomes easily available. Keeping that in mind, you should try and restrict your purchases to systems that you can repair and find present day support for. Many systems however will straddle the fence by providing somewhat more complex software than most can understand and yet have a standard enough system design such that support is a minimum aspect of the system (PC-clones).

The two systems we talked about will undoubtedly do their job. Our friend may learn considerably more than he had planned, and will most likely have a system of less quality than he needs. While on the other hand our 520 may prove useful only after more money and time is put into the project than we had hoped for. It is possible to see that computer systems require certain amounts of flexible thinking to be used fully. The next five years will continued to see things get smaller and become more intelligent. I expect that by 1991 hardware cost will be so low that truly a system in every home will be possible. That system however may not be a real computer as such but a complete learning system. That system will be used by a student from early school years well into college. The success of such a system will be mostly based on a standard interface and concept design which will come from outside the regular computer industry (a textbook company).

For myself, I expect to continue to use my Z80 S-100 system. The base of software and my understanding of how it works makes any major changes unrealistic. I will build and learn about other systems (68K), but my understanding and familiarity with my existing programs, makes it prohibitive to change completely. ■

★★★ Editor's comments ★★★

No-Nonsense Licenses

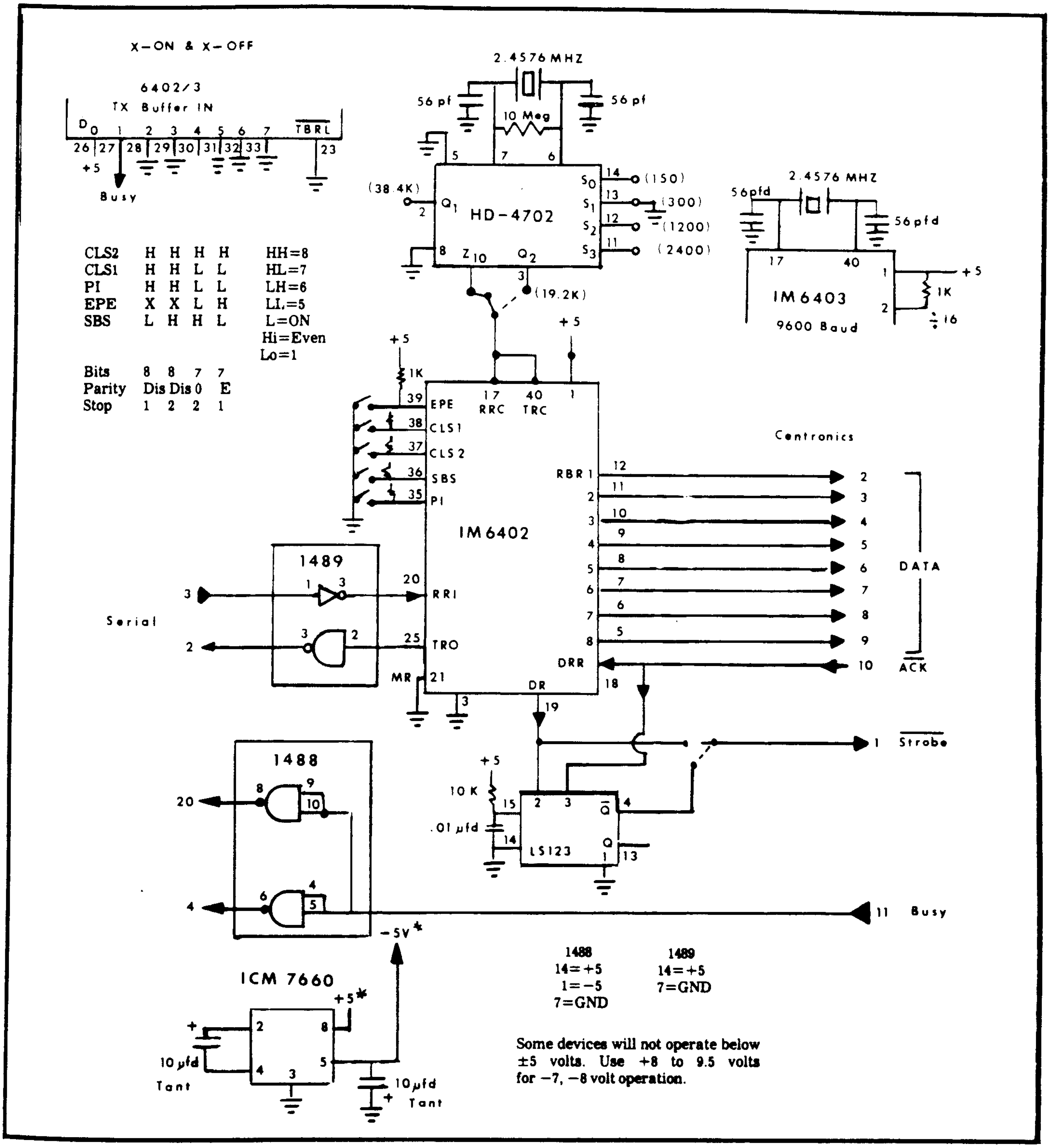
Most of the software copyrights and/or licenses have been overly restrictive, limiting the use to only one CPU. Some actually stated that if you replaced your computer, you would have to re-purchase or re-license your software in order to continue to use it on the new CPU. They also did not allow you to use the software on one computer in the office during the day and then take the software home to use on another machine in the evening.

The good news is that Borland and Datalight have adopted a very realistic attitude about licensing, and we want to encourage others to follow their lead.

Datalight's "No-nonsense License Agreement" for their C compiler (Datalight, 11557 8th Ave. N.E., Seattle, WA 98125) says that you should treat it like a book. It can be used by any number of people and freely moved from one computer location to another—just as long as there is **No Possibility** of it being used at one location at the same time that it is being used at another. There is also no license fee for distributed derivative programs.

Borland's "No-Nonsense License Statement" for their Turbo Pascal version 3.0 also says that you should treat it just like a book. In fact, the details read almost the same as Datalight's, and I'm not sure who adopted it first.

We should all encourage this development and let these companies know how much we appreciate their license, the fact that the programs are not copy protected, and the very reasonable cost. We should also recognize their rights and not pass out or accept copies which would violate their liberal agreement. If you know of other companies with these policies, send TCJ the details so that we can publish the information. ■



Some devices will not operate below ±5 volts. Use +8 to 9.5 volts for −7, −8 volt operation.

system on me, I got very disturbed. We haven't been able to determine if there are bugs in the system or if we were doing things wrong. They do have a checking routine but all that does is throw you back into the system if you make a bad call. We will be trying to load a true Forth-83 later but for now I would like to hear if others are using this program or not. We think this machine has some real possible uses and are looking for a good software package to implement it in. Forth would be ideal but not if we have to write our own true Forth-83.

Lastly for those other Forth-83 users, I am still working on the Forth in ROM project. I have purchased a used Godbout 68K CPU card and will be installing it in an S-100 system. My current plans are to do a Forth monitor/ROM for it and then try some operating stuff, (using the disk controller featured two issues back) all in Forth. I will keep you informed on how that is going, so till then...keep Forthing....

Selecting the IDEAL System

For those people who have been trying to find that IDEAL system let's talk some about what to look for and also what we can expect in the future. Let's start out by looking at some possible news articles, five years in the future.

(Continued on page 46)

THE COMPUTER CORNER

A Column by Bill Kibler

Well, this column will be longer than those in the last few issues, as many things are starting to happen. My term as editor of my local computer club's magazine will be over by the time you read this. Although it has been fun and interesting it has surely hampered my output in these pages. This month I have included a schematic of a serial to centronics converter and will discuss a rather interesting electronics show.

Serial to Centronics

I have had an MX80 printer for a rather long time now and from time to time I put it on a serial only computer (Superbrain). Currently I change the serial converter card by opening the MX80. This removing screws and trying not to break cables has gone on too long so I have undertaken this quick fix. The card in Figure 1 is based on the idea that you will install it in a system (get all the serial lines setup) and move the printer around. This is necessary, I found out, due to the many different forms of handshaking. The circuit is rather simple and I will not go into details other than possible problems you may encounter.

I built one unit and it worked perfectly at many baud rates. I could even use it without the single-shot (LS123). The next unit I built would not work without the single-shot and beefing up the −5V supply. I finally decided the problem is with the source of the RS-232 signal. Some drivers will work down to the (RS-232) specs lower limit of 3V, while others will not work properly below 5 or 6V on the negative side. I found that by using a five volt regulator for the other chips, and then supplying a separate 7 to 9.5 Volt source for the 1488 and 7660 devices, the unit could be made to work properly. I found a threshold of about 8.5V. The unit would drop a character below this, but run fine up to 9600 baud with voltage above it. Different systems have different requirements, so as usual experiment for best operation.

The main thing to keep in mind is the serial port must have handshaking. It needs to monitor a busy or not ready signal on the serial side. Should it be set up for X-on and X-off protocol you might try the hookup as shown. I haven't tried this as yet, but think it should work just fine. The software will need to check for X-on/X-off received data after each send, this is usually at 2400/4800 baud and so most computers will have enough time to check the status. You must make sure that the output data is being continuously output, I have indicated using the ACK signal as a strobe, but have realized that a busy would never be cleared, as the needed strobe to send the OK would not have happened till after a completed send. This means some form of free running single shot (or part of the baud rate generator) must be used to guarantee sending of status (X-on/X-off) regardless of printer condition.

You have two versions of the UART to use; a 6402 and a 6403. The only difference is a built in divider and crystal oscillator in the 6403 version. I bought a 6403 but then decided to use the 4702 baudrate generator when I had troubles at 9600 baud. Unless you intend to use 9600 buad, try the 6402. It is cheaper and the 4702 will give you more options. In any case you can build a simple converter with as few as 3 devices or worse case 6 chips.

WESCON

Once every two years the WESCON electronics show happens in San Francisco. This year the show only had electronic vendors, compared with two years ago when the mini-micro computer exhibit shared the spaces. What I saw this year was quite different from last time. The number of finished or complete system products were noticeably missing. Although several exhibitors had systems (like VME bus) most were component manufacturers looking for new OEM's to buy their stuff. A very noticeable addition was the number of foreign countries present. The slacking off of computer and electronic sales has left many foreign countries with excess production ability. Our country's strong dollar has also helped to make off shore manufacturing a strong competitor in supplying raw products.

We were looking for test equipment, mainly of the logic or emulation type. We found lots of products, most starting over $10,000 for a single type emulator. Some of the cheaper logic analyzers got almost as high when software emulation was added. One unit we saw and liked is the ORION work station. ORION (702 Marshall St. Suite 614, Redwood City, CA 94603 415-361-883) makes what they call a development lab. This unit works with either CP/M or MSDOS systems and can emulate or replace a rather large number of devices. We used it for several minutes and were quite impressed by the ease with which you could stop or find bugs in software. Its only drawback was true analyzer type functions. It is possible to monitor a number of lines at the clock rate and look for improper timing signals. Should your problem be spikes or minor timing variations this unit might not show them up. When you include this unit's ability to emulate ROMs at the same time it is looking at signals, the $4,000 price tag looks rather good. My favorite thing about the unit is that the utilities are all written in Forth. It was rather interesting to hear the sales person say how they don't mention Forth unless the user notices it. It seems they have gotten some rather strange reactions when they say Forth.

Forth and 520ST

I had a great idea the other day. We use $3,000 terminals at work, and they are all dying. Now replacing three of them at their current price didn't sound too good, but I reasoned that we could buy a new Atari 520 ST with color monitor and then emulate the old terminal. I further chose the only Forth package available to do it with. Now let me say I am really impressed about the 520 and may even buy one myself. The unit with the black and white monitor is OK but you will have to buy a color monitor to really enjoy the system. The color screen is really amazing, in fact their 80 by 24 color text is much better than some PC types I've seen and could be used all day long. There are not too many programs right now, but hang on because England has lots of people working on it. The operating system is a version of CP/M 68K and in fact you can read 5¼ inch MSDOS disks. I've seen the competition, AMIGA, and for the extra money they charge, I feel the ST is a better buy. Later on I will have more to say about the insides, as I will be opening it up and probing around.

The Forth system is 4xForth (four by Forth) by The Dragon Group. This was supposed to be a Forth-83 version, and based on other articles, I knew that a few things were different. Let me say that more than a few things are different—almost all of it is different. The ST takes about two minutes to boot up and when Forth kept resetting the