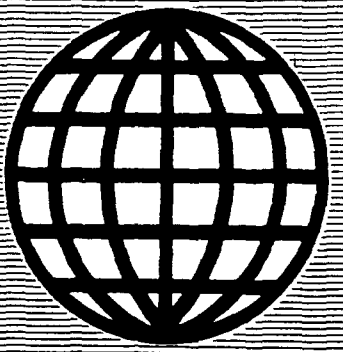


The COMPUTER JOURNAL®

Programming - User Support
Applications

Issue Number 29

$3.00

Better Software Filter Design
Writing Pipeable User Friendly Programs

MDISK
Add a One Megabyte RAM Disk to Ampro L.B.

Using the Hitachi HD64180
Embedded Processor Designs

The ZCPR3 Corner
Announcing ACPR33 plus Z-COM Customization

68000
Why Use a New OS & the 68000?

Detecting the 8087 Math Chip
Temperature Sensitive Software

Floppy Disk Track Structure
A Look at Disk Control Information & Data Capacity

ISSN # 0748-9331

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
Joe Bartel
C.Thomas Hilton
Donald Howes
Bill Kibler
Rick Lehrbaum
Frederick B. Maxwell
Jay Sage
Kenneth A. Taschner

Entire contents copyright© 1987 by The Computer Journal.

***Subscription rates**—$16 one year (6 issues), or $28 two years (12 issues) in the U.S., $22 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.*

Send subscriptions, renewals, or address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, Montana, 59912, or The Computer Journal, PO Box 1697, Kalispell, MT 59903.

Address all editorial and advertising inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 phone (406) 257-9119.

The COMPUTER JOURNAL

Features

Issue Number 29

Better Software Filter Design
A compromise that allows programs written under DOS to be pipeable, yet still user friendly
by Kevin Lacobie 5

MDISK
Add a one megabyte RAM Disk to your Ampro Little Board
by Terry Hazen & Jim Cole 13

Using The Hitachi HD64180
Coping with object code incompatibility, wandering I/O addresses, and using the ASCII ports.
by Kenneth A. Taschner & Frederick B. Maxwell 18

68000
An alternative to the PC for high performance systems
by Joe Bartel 31

Detecting the 8087 Math Chip
Routines which work on the XT don't always work on the AT
by E. Clay Buchanan III 36

Floppy Disk Track Structure
How track structure and sector size affect the formatted capacity
by Dr. Edwin Thall 38

Columns

Editorial 2

Reader's Feedback 3

ZCPR3 Corner
by Jay Sage 22

Book Reviews 46

Computer Corner
by Bill Kibler 48

Editor's Page

6800 Developments

IBM's recent announcements may (or may not) have a major impact on what's happening in the office appliance computer scene, but Hawthorne's progress with their 68000 Tiny Giant and their K-OS ONE operating system is much more exciting. While we may be forced to work with IBM's and compatibles because that's what our customers are now using, the '386 systems which will be designed for business will be about as much fun to work with as the old mainframes.

Just as the microcomputers using 8080s and 6502s developed into a separate market from the mainframes and the minis; the micro computer market will separate into the business machines which will largely replace the mainframes and the minis, and something else which is fun to work with. In the past, programmers and hardware hackers slaved on mainframes during the day and then went home to get personally involved with their computers — even average people banged away on Apples® because it was fun.

The large volume of sales is in business oriented machines, and as they become more 'main frame like' they will also become less interesting for individuals. People who want to get involved with the hardware and the operating system will have to turn to something else. Some possible contenders are the Apple IIGS® and the Atari® , but they lack the flavor of the old Apple II+ and CP/M systems. People will be looking for something which is more powerful than the eight bit systems, and yet friendly. Something where they can modify the operating system and design their own add on boards. The only answer I see right now is a 68000 system using Hawthorne's OS which comes with the source code.

Hawthorne has been making steady progress expanding their product line with items like their new Editor Toolkit, and they're working with people on a communications package, a Forth implementation, and possibly a BASIC, but what we need is the third party and hacker involvment we had with the early Apple II. If you want to develop code or hardware for a 16-bit system where everything including the operating system is under your control, you should write to Hawthorne for their information package. If enough of us get involved we can get a C compiler running and perhaps convince Joe to produce a board with slots for easy expansion. Write or call Joe Bartel at Hawthorne Technology, 8836 S.E. Stark, Portland, OR 97216 (503) 254-2005, to find out what's happening in the 68000 world. He'd really be interested in talking with anyone who wants to develop a hardware or software package for K-OS ONE.

If you'd like to write an article or a column on the 68000, I'd be real interested in talking with you (even short notes and letters will be appreciated).

Graphics Mania

Former CP/M users are complaining that too many PC/MS DOS utilities use fancy graphic displays which make them unprintable. CP/M users are used to using control-P to toggle on the printer during an assemble, compile, or debugging session in order to generate a complete hard copy history of a work session, and this does not work with PC programs having elaborate graphic displays. The question here is whether we have impressive graphics or useful information, and I'll vote for having the hard copy information I need instead of pretty displays which impress the sales people and non-technical users.

I fully agree that there are instances where bit-mapped graphics are necessary in order to display charts, graphs, illustrations, and other non-character type material; but I strongly object to being forced to look at some designer's idea of modern art while working with a character oriented programming utility — in fact I just won't use them! I fully agree with Joe Bartel (see his 68000 section) that if you need graphics it should be on a second monitor, and if you need multiple windows they should each be on their own monitor instead of many tiny windows on one screen.

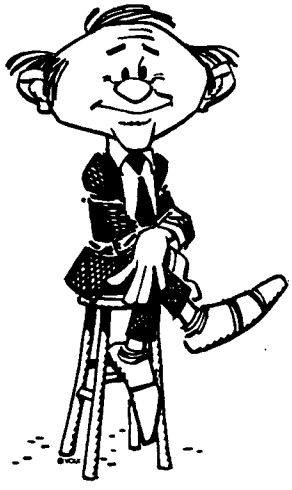
Compare these ideas with the former idea that everyone should work at a dumb terminal attached to the main frame — they couldn't understand why an individual would want the power of our current micros on their desk. Now we're saying that people will want multiple CPUs and multiple screens instead of multitasking one one CPU and windows on one screen.

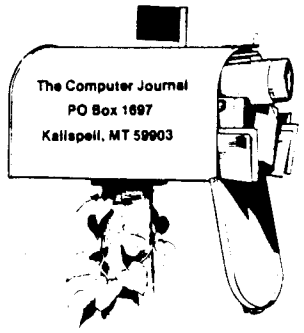
Everyone is welcome to their own opinion, and mine is that for character oriented work I want a high speed character oriented monitor with no fancy graphic borders or other distractions. At least the program designers could provide a toggle so that the user could select between a plain character and a fancy graphics interface.

This is probably a cultural issue. People who were raised watching color cartoons on the boob tube, and who are not comfortable reading a book, want to make their computer monitors look like the Saturday morning funnies. I like reading books, and I want my screen to look like a book — unless I need graphics for illustrations or CAD. Even books have illustrations where needed, but there is a difference between a good novel and a comic book!

If this doesn't get someone mad enough to write, I'll give up trying to get any response. The purpose of this Journal is to provide communications, so if you have something to say send it on disk or hard copy.

(Continued on page 44)





Reader's Feedback

Chicken-And-Egg Problem — Part One

I have been long concerned about the chicken-and-egg problem with new microprocessor chips, and I have a suggestion which I believe would aid the proliferation of the Series 32000 microprocessors. Now, I have much respect for the Herculean effort that National has made in developing the 32000, so please don't think that I would frivilously recommend additional work that National should do.

Thinking about the development of the 32000 makes me think of a marathon bicycle race. In approaching the last hill, the front-runners are usually in a pack. Suddenly, one cyclist will break from the pack and very strenuously apply everthing that he has in an attempt to be the first to reach the top of the hill. If he makes it to the top of the hill without croaking, he's the clear winner; he will sail to the finish line with no possibility of the others catching up. Since I see National selling those very inexpensive 32000 designer's kits, it seems to me that National is now attempting to break from the pack. When the cyclist is in the middle of his big effort to make good his break, it is likely that the legs will send messages to the brain indicating that something additional is needed. Likewise, in order for the 32000 to really decisively break from the pack, something additional is needed.

Consider the early days of the 8080. There were very many people who were regular engineers by day and 8080 experimenters by night. When the manufacturers began to build products using microprocessors, the industry turned into a big vacuum cleaner trying to suck up everyone who had experience with microprocessors. The only people who could answer the job ads were those who had experimented with microprocessors at home at their own expense. I disregard the engineers that the various firms extract from one another because there is no gain in this kind of exchange. Because of the availability of these self-taught microprocessor users, the 8080/Z80 boom was spectacular. Intel is still riding on the 8080 boom that was given to them by amateur experimenters.

Consider the early days of the 68000. For the experimenter, bootstrapping his own operating system and assembler was too much of an obstacle. Therefore, the amateurs stuck to their Z80's although they had great lust for the 68000. Motorola did nothing to cultivate self-taught users; they catered only to OEM's. The OEM's built nothing but white elephants because there was no supply of amateur 68000 users to give the OEM's the kind of expertise that they needed. If there had been a supply of self-taught users of the 68000, IBM could have used the 68000 in the first PC. If Motorola had issued a simple public domain operating system and assembler for the 68000 the same day that they introduced the chip, it is possible and even probable that we would not see any Intel segmented microprocessors in any personal computers today. For the first time in the life of the 68000, there is now an economical ($50) operating system with an assembler available to allow amateurs to begin work with the 68000. It's not from Motorola; it's from Hawthorne Technology.

Consider the present days of the 32000. National sells some very nice development tools, but they are all for OEM's. There is nothing for amateurs. The OEM's are producing some exceptionally wonderful white elephants, but nothing that is apt to proliferate enough to give the 32000 the widespread usage that it deserves. There is no supply of experienced 32000 enthusiasts to give the OEM's the kind of expertise that they need. The designer's kits go a long way toward putting the 32000 in the hands of passionate experimenters, but in this day and age the chip set is not enough. There is a grass roots 32000 interest group germinating in the readership of Micro Cornucopia Magazine, and I have corresponded with a number of amateurs who are fabricating their own experimental 32000 board. It is excruciatingly painful for me to think of the great duplication of effort as each of these experimenters attempts to bootstrap his own operating system and assembler. The assembler in the Tiny Development System that comes with the designer's kit is nice but is not sufficient for the task of bootstrapping an operating system. I have been attempting to popularize the 32000 by giving away all the 32000 code that I write. I'm in the middle of writing 32HL (32000 hacker's language), which is a stand alone combination of operating system, assembler, high level language, and editor. I plan to give it away to the public domain even though I am not gainfully employed. I have temporarily given up this effort because the cross assembler that I purchased from 2500AD was no good.

Here is my suggestion: National should issue a public domain simple single-user operating system (with assembler) for the 32000 which any arbitrary experimenter can port to his own unique 32000 board. Such an operating system should be at least as simple as CPM 2.2 and must have a customizable BIOS similar to that of CPM 2.2. Note that the customizable nature of CPM is in part responsible for the 8080/Z80 boom. Each experimenter can develop his own BIOS using the Tiny Development System. With this kind of system in the hands of 32000-loving experimenters, there is no way that the 32000 could avoid being a smash hit.

Neil R. Koozer

Chicken-And-Egg Problem — Part Two

I think I've finally figured out how to solve the chicken-and-egg problems with the 32000. It involves re-writing my 432 system with a different foundation. The new version will start out as an assembler so that it will be able to compile itself in one quick pass at any stage of its development. It should progress something lke this:

1. Create an NS32 native version of the cross assembler Z32. It will call Z80

CP/M for file I/O.

2. Add high-level keywords such as IF, ELSE, REPEAT, UNTIL, etc. At this point it's a language compiler that accepts either high-level syntax or assembly syntax in any statement. After this, any new features can be written in high level language, and pre-existing features could be changed to high level language.

3. Add an outer loop which accepts keyboard input, compiles it, and executes it. This loop creates the features of command processor, high-level language interpreter, and assembly-language interpreter. It will return even after stack-altering statements are executed. The DOS commands (PIP, DIR, etc.) will be in the language vocabulary, so they can be executed from the keyboard or from programs. Therefore, no batch processor will be needed. Nothing like ARGC or ARGV will be needed because the keyboard invocation of a program will be treated like a normal function call with a normal parameter list.

4. Since an assembly-language interpreter is almost a debug monitor, simply add the few remaining commands to make it a full debug monitor. These will include disassembler, one-key single stepping, one-key trace-through-call, and auto-updating display of registers and chosen memory locations.

5. Allow string variables to be compiled the same as include files so that string variables could be used as keyboard macros, small batch files, etc.

6. Integrate a screen-oriented editor into the outer loop. The same editor would be used to edit command lines and source files. The command line would be stored so it could be re-edited and re-executed any number of times.

7. Add or change enough features to allow any arbitrary experimenter to port this code to his unique 32000 computer.

Because of the speed of this compiler there may be no need to store any compiled programs, but just store the source code. At any time, an old version of the (compiled) system would be in EPROM. Upon power-up or reset, it would check for an 'autoexec' file, which could have instructions to compile the current version of the operating system and to execute it. This operation would be finished long before the CRT lights up. To install a new version of any module, simply put the new source code for that module on the boot disk and reset. All library modules for the high-level language could be included as well since replacement is so easy. Whenever the EPROM version becomes inadequate for compiling the current version, a new EPROM is burned.

At the end of any compilation, the symbol table would contain the symbols corresponding to any global variables or procedures so that they can be used manually or used by any subsequently compiled program. With this scheme, any program could make symbolic reference to functions in the BIOS, BDOS, and library without the use of jump tables, link tables, or external addressing mode. A BIOS jump table would be temporarily used in a distribution copy until the user has the system running.

Neil R. Koozer

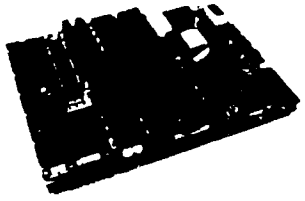
HD64180

In issue #27 you published a article about the HD64180. Since then Hitachi has released, or is about to release, the Z mask version which is more tolerant of Z80 peripherials.

I am running a Xerox 820-II with ZC-PR-2. My Z80 is resetting the system once in a short while after a cold start. Rather than buying a new Z80 I would like to

(Continued on page 35)

MDISK



A fast 1 megabyte RAM disk for your AMPRO Z80 Little Board!

Fast RAM workspace greatly speeds up disk-intensive operations like wordprocessing, database access, and program development.

- 5.75" x 5.15" printed circuit board plugs into your AMPRO Z80 socket
- Add standard 256K RAM chips for up to 1 megabyte of extended RAM
- BIOS resident MDISK driver software enables the extended RAM to be used as a solid state disk drive, complete with system track for instant warm boots
- Driver software supplied as ready-to-install BIOS hex files for several common system configurations. Source code for BIOS driver inserts also included
- Includes utilities for boot-time configuration and extended RAM testing
- Requires 5vdc at .60 amp via standard disk drive power connector
- Little Board must be modified to replace the 64k RAM chips with sockets and to add one jumper. Complete instructions included

MDISK (0k RAM supplied), including complete manual and software disk, only $149 plus $5 shipping and handling. California residents add 6% sales tax. Checks, COD, MO accepted.

n/SYSTEMS

21460 Bear Creek Road, Los Gatos, CA 95030

We have CP/M software

NewWord — the better word processor includes WORD plus by Oasis!
CP/M 80..... $69

SuperSort — original sorting program
CP/M or MSDOS..... $85

Turbo Pascal — version 3.0
CP/M or PC/MS DOS..... $45

Turbo Toolbox for CP/M or MS DOS..... $39

The above items are limited to stock on hand.

C/80 Compiler..... $45

DateStamper — adds date and time to your CP/M directory..... **$45**

8" SSSD disks, per box of 10..... $5

Hardware and software for Heath/Zenith computers.
Add $4 per order for shipping and handling.
Terms: Check or Money Order — Visa/MC — COD.
California residents add 6% tax.

ANAPRO **805/688-0826**
213 Teri Sue Lane **Buellton, CA 93427**

Better Software Filter Design

Writing Pipeable User Friendly Programs

by Kevin Lacobie, University of Texas Medical Branch

Summary: One of the more powerful features of the UNIX operating system is piping, which allows the output of one program to be immediately sent as the input to another program. While MS-DOS has the piping feature, most programs written in this system are not directly compatible with piping. The reason is because of a conflict between terse 'pipeable' and friendly 'readable' output, and MS-DOS has chosen a more readable output, for the perceived casual user of its personal computer operating system. A compromise is offered that allows programs written under DOS to be 'pipeable', yet still 'user friendly'.

In computer operating systems supporting input and output redirection and piping, the use of filter programs has become popular. Output redirection refers to sending output from a program to a device or file other than the default, which, for most interactive systems, is the user's console. Input redirection is similar, taking input for a program from something other than the interactive device, the keyboard. Piping refers to the process of having the output of one program be the input of another program. Filter programs are "programs that read some input, perform a simple transformation, and write some output."(1) The UNIX® operating system has become the most noted for its use of redirection, piping, and filters.

For piping and filter programs to be robust, all programs in the system must conform to some standards. Namely:

> "The output produced by UNIX programs is in a format understood as input by other programs. Filterable files contain lines of text, free of decorative headers, trailers, or blank lines. Each line is an object of interest — a filename, a word, a description ... When more information is present for each object, the file is still line-by-line, but columnated into fields separated by blanks or tabs ..."(2)

Since each program conforms to this, multiple programs can be piped together, to transform some input into some interesting new output. Given a range of robust filter programs, many different interesting results can be derived by properly piecing these programs together. "This powerful technique allows programs to be written as small, compact, one-purpose-only packages which can be used in conjunction with other programs to form more complex packages."(3) Thus, this often results in higher productivity, for each time a new result is needed from some input, a new program need not be written.

Interestingly, the MS-DOS® (Microsoft Disk Operating System) found on most personal computers supports redirection and piping. However, many of its commands do not comply with the above standards. This renders filter programs less functional, even useless. MS-DOS has not whole-heartedly embraced the advantages of the above standards. Interpreting this treatment toward the functionality of piping and program interaction leads to understanding some of the deficiencies in an operating system environment based entirely on the above standards.

MS-DOS has been largely created from UNIX and CP/M, an earlier operating system for microcomputers. Being for personal computers, the outputs of most commands are designed for usability, or 'user friendliness.' Thus, decorative headers, trailers, and blank lines are used. This not only makes for a more attractive display, but gives readable information to the user. For example, the DIR command gives a list of all files in one subdirectory on a disk:

```
 Volume in drive B is KEVIN

 Directory of  B:\

ARTICLE2         4138  11-14-86   1:06p
ARTICLE          2408  11-13-86   1:33p
WORDS    USE       66  12-01-86   5:33p
WORDS       <DIR>     11-13-86  11:58a
ARTICLE3         1621  11-17-86  10:42a
UNIXPC           3265  12-18-86   3:15p
CAPS     SCR      335  12-01-86   2:12p
CAPS     COM       56  12-01-86   2:12p
COMMITTE    <DIR>     12-18-86   3:20p
FILTERS          6662  12-19-86  12:12p
       11 File(s)    222208 bytes free
```

The output from DIR is indeed readable. However, it is nearly useless for filter programs. A user may want to sort the information by date, for example. A sort program could sort each line from the output of DIR, but it would sort the header lines and the summary lines in its output, thus not producing the expected output.

A typical UNIX command, when invoked, gives only a terse response. As pointed out above, this terse response is excellent for the standard, for another program could use its response as input. However, for a user, most "UNIX tools tend to be terse to the point of unhelpfulness."(4) For example, WC is the word count program. It will count the number of words in a normal text file. Yet, when invoked, it will respond with a line similar to:

```
59   199   2029   file
```

The output's meaning is unintuitive to the user. Only if the user remembers this command, or if header information is printed, will it be known that 'file' contains 59 lines, 199 words, and 2029 characters.

A typical DOS command might give such header information. This has proved very helpful for users, who on personal computers tend to be casual users. Thus, very few programs in MS-DOS have been written to the standard of filter programs. It can be argued that this has resulted in the loss of some functionality, as a new program must be written for every new task. In summary, DOS has given up the advantages of the pipe, in favor of more user readable output.

Certainly, some compromise between the standards for good filter programs, which leads to well-integrated tools, and more friendly output is desired. In "The Critique of UNIX," such a compromise has been described. A high level construct should be available for a program to know whether "it is being called as 'stand-alone' or part of a chain of programs."(5) With this construct, programs can, without much effort, take on a very different shape. If the input was not piped in, then the program can prompt the user for needed information. If the programs's output is not to be piped, then header information, more proper spacing, and summary information can be printed. However, if in a pipe, a program can output in the terse, single line per object format famous in UNIX programs. Now, the 'best of both worlds' is permitted.

A simple example below shows how a program would be written with this construct:

```
extern int standalone() ;
main()
{
int _standalone,sumt ;

_standalone = standalone() ;
if (_standalone)
  printf("PROCESSING HEADER\n") ;
doProcessing() ;
if (_standalone)
  printf("SUMMARY DATA: %d\n",sumt) ;
}
```

The only responsibility of DoProcessing() is to output information one line per object. Only if in a standalone mode will header and summary information be printed, and this is handled by main().

Standalone can be easily added to MS-DOS. Since version 2, a DOS function call is available which allows the program to interrogate device information. With this function call, standard output can be interrogated to see if it is a character or block device. Character devices are such entities as printers and consoles, while block devices are for storage of files. Since piping is done through temporary files, it suffices to determine if the standard output is a character device or not, to determine if it is part of a pipe or not.

Standalone can be created as a function utilizing this method. Below is shown the function in 'C', with a standard library call to make a DOS call.

```
#include <dos.h>
int standalone()
{
  REGS regs ;
  int flag ;

  regs.ax = 0x4400 ; /* DOS funct# for IOCTL */
  regs.bx = 1 ;  /* DOS # for standard output */
  flag = intdos(&regs,&regs) ;
/*check Bit 7 to determine if character device*/
  regs.dx &= 0x80 ;
  return (regs.dx == 0x80) ;
}

/*
   See latest version of DOS Technical Reference
Manual, under the chapter "DOS Interrupts and
Function Calls", for the description of function
call 44 Hex, "I/O Control for Devices (IOCTL)"
```

Similarly, InAlone() can be written to test whether the program's input is coming from the standard input device, or was redirected or piped. InAlone can be used to determine the necessity of prompts to the user. The only difference in the above code for 'inalone' is <C>regs.bx = 0<C>, to signify the DOS number for standard input.

With these constructs, programs should conform to some new standards. Namely, the filter program standards mentioned above should be used, but when 'standalone' (and only when 'standalone'), a program should print a header describing the output, appropriate blank lines and pagination, a header describing each column of information, and summaries giving totals or other relevant information about the collection of objects that the program processes.

With this in mind, LS.C (Listing 1) has been written to give an example of the use of 'standalone'. LS is also a useful program for DOS systems. First, it can display **all** that DOS knows about a file. Secondly, it is able to list files in all subdirectories in the hierarchical file structure that DOS has had since version 2.

The syntax for LS is:

```
LS [searchpattern] [/A{A|R|D|H}] [/F] [/D] [/T] [/S]
```

'Searchpattern' may contain drive and directory information. Without any flags, LS will display only file names. Adding flags /A (Archive), /F (File size), /D (Date), and/or /T (Time) will display more information about each file. Qualifying the /A flag will list only files with the qualified attribute(s) (A for Archive, R for Read-only, D for Directories, and H for Hidden (and System) files). Adding /S will allow LS to display files under all subdirectories of the specified search pattern.

If standalone, LS will output in the following format:

```
                DOS FILE INFORMATION

        DIRECTORY: d:\path

          FILE1  sss dd/mm/yy hh:mm Attr
          FILE2   "     "       "    "

        DIRECTORY: d:\path\subpath

            FILE3 "     "       "    "
            ...
```

If in a pipe, however, the output is only one line per file, in the following format:

```
d:\path\FILE1  sss dd/mm/yy hh:mm Attr
d:\path\FILE2   "     "       "    "
...
```

The latter output is more useful for other programs to perform sorting, selecting, or selective file processing, and be able to access files, regardless of which subdirectories they are in.

As another example, InAlone() is used in the the ParseCommand() procedure (Listing 2), which can be used to add flexibility to normal filter programs. ParseCommand will look on the command line, or prompt the user, whenever it determines that the input is not being redirected. This allows a program to be called in one of three ways: either as a filter, with syntax: program <infile >outfile; as a command, with syntax: program infile outfile; or as a prompted command by entering just program, and the program will prompt the user. The trick is in the fact that 'stdin', the standard input, can be closed and reopened to a program sup-

Listing 1

```
/*
  Program to list DOS files.  Similar to DIR except that only
  information wanted is displayed, file attribute information
  can be included, and it can list files in all subdirectories

  When the user includes specific attributes on the command line,
  LS will only list files containing those attributes.
*/

#include <dos.h>
#include <ctype.h>
#include "myfind.c"
#include "alone.c"
#define NULL 0
#define ON 1
#define OFF 0
#define MaxIndent 60
#define IndentAmount 2
#define LF '\n'
#define BACKSLASH '\\'
#define USAGE "Usage: LS [searchpattern] [/A{A|R|D|H|}] [/F] [/D] [/T] [/S]"
#define HELP1 "  /A to list attributes\n  /F to list file size"
#define HELP2 "  /D to list file date\n  /T to list file date & time"
#define HELP3 "  /S to list files in all subdirectories also\n"

  struct
  {
    unsigned dir : 1 ;
    unsigned time : 1 ;
    unsigned date : 1 ;
    unsigned size : 1 ;
    unsigned att : 1 ;
  } flag ;
  int level = 0,alone = 0, attr;
  char pattern[80] ;
  struct FINDAREA *MyFindFirst(),*MyFindNext() ;

/*  Program logic:
    1.  Parse command line, setting appropriate flags and creating the
        specified search pattern.
    2.  Call Search, which will print all specified files.
*/

main(argc,argv)
  int argc;
  char *argv[] ;
{

  StrCpy(pattern,"*.*") ;
  attr = 0 ;
  flag.date = flag.time = flag.size = flag.att = flag.dir = OFF ;
  ParseCommand(argc,argv) ;

  alone = StandAlone() ;
  if (alone)
    PrintHead() ;
  Search(pattern,attr) ;
}

/*  Logic of SEARCH:
    1.  Set LEVEL.
    2.  With the specified search pattern, call the DOS function FIND FIRST.
    3.  If no file found, try adding a wildcard at end of search pattern, to
        see if the pattern only a dirve specifier or directory.
    4.  If 3 fails, return.
    5.  Else, print this file information, and loop through DOS FIND NEXT,
        until no more files exist.
    6.  If FLAG.DIR is set, then user wants to have LS display files in all
        subdirectories.  So, search for any subdirectories in the current
        directory.
    7.  For each subdirectory, splice the directory name string to the search
        pattern, and call Search for this NEW pattern.
*/

  Search(pattern,attr)
  char *pattern ;
  int attr ;
  {
    char path[65],name_ext[12],new[80] ;
    int first ;
    volatile struct FINDAREA area, *info;
    /*  AREA is needed for the DOS Functions Find First & Find Next to work  */

    first = ON ;
    level++ ;
    if ((info = MyFindFirst(pattern,attr,area)) == NULL)
    {
      StrCat(pattern,"\\*.*") ;          /*  See if its a directory  */
      info = MyFindFirst(pattern,attr,area) ;
    }
    SplitPattern(pattern,path,name_ext) ;
    while (info != NULL)
    {
      if (first)
      {
        first = OFF ;
        if (alone)
          Printf("Directory: %s\n",path) ;
      }
      PrintLine(path,area) ;
      info = MyFindNext(attr,area) ;
    }
    if (flag.dir)   /*  Search for all directories  */
    {
      strcpy(new,path) ;
      strcat(new,"*.*") ;
      info = MyFindFirst(new,DIRECTORY,area) ; /*  Get first Directory Name  */
      while (info != NULL)
      {
        StrCpy(new,path) ;
```

LS.C

```
        StrCat(new,info->name) ;  /*  Splice in Directory Name as part of  */
        StrCat(new,"\\") ;        /*   search pattern                      */
        StrCat(new,name_ext) ;
        Search(new,attr) ;        /*  Search all files in this directory   */
        info = MyFindNext(DIRECTORY,area) ;  /*  Get next Directory Name   */
        level-- ;
      }
    }
  }

/*  Splits search pattern to prefixed pathname and suffixed file name/extension.
*/

  SplitPattern(pattern,path,fileext)
    char *pattern,*path,*fileext ;
   /*  Pattern will be in the form: 'd:path\path\filename'.
       By searching from the end of the string, go until finding the first
       backslash or colon.  Then, the first part of the string will be
       'D:path\path\', while the remainder will be 'filename'.
   */
  {
   int i ;

   i = StrLen(pattern) ;
   i-- ;

   while ((i >= 0) && (pattern[i] != BACKSLASH) && (pattern[i] != ':'))
     i-- ;
   if (i == 0)
   {
     path[0] = '\0' ;
     StrCpy(fileext,pattern) ;
   }
   else
   {
     i++ ;
     StrnCpy(path,pattern,i) ;
     path[i] = '\0' ;
     strcpy(fileext,pattern+i) ;
   }
  }

/*  Logic of PRINTLINE:
    1.  Print indentation if ALONE.
    2.  Else, print path string.
    3.  Print File Name.
    4.  For each Flag, print related file information, converting
        file date and time to displayable format, if necessary.
*/
  PrintLine(path,info)
    char *path ;
    struct FINDAREA *info ;

  {
    int mm,dd,yy,minute,hour ;
    char pm[3] ;
    char *charattr() ;

    if (alone)
      Indent(level) ;
    else
      Printf("%s",path) ;

    Printf("%-12s",info->name) ;

    if (flag.size)
      Printf(" %6ld",info->size) ;
    if (flag.date)
    {
      dd = getbits(info->date,4,5) ;        /*  See DOS Technical Reference  */
      mm = getbits(info->date,8,4) ;        /*  Manual, 'DOS Disk Directory' */
      yy = getbits(info->date,0x0F,7) ;     /*  on how DOS stores Date and   */
      Printf(" %2d-%02d-%02d",mm,dd,yy) ;   /*  time information             */
    }

    if (flag.time)
    {
      pm[1] = 'm' ;
      pm[2] = '\0' ;
      minute = getbits(info->time,0x0A,6) ;
      hour = getbits(info->time,0x0F,5) ;
      if (hour > 12)
      {
        pm[0] = 'p' ;
        hour -= 12 ;
      }
      else
        pm[0] = 'a' ;
      Printf(" %2d:%02d%2s",hour,minute,pm) ;
    }
    if (flag.att)
      Printf(" %s",charattr(info->attribute)) ;
    putchar(LF) ;
  }


  PrintHead()
  {
    Puts("               DOS File Information\n") ;
  }


/* Translate the integer ATTRIBUTE into displayable characters */
  char *CharAttr(att)
    int att ;
  {
    static char field[5] ;
    int i ;

    for (i=0;i<5;i++)
```

```
    field[i] = ' ' ;
  i = 0 ;
  if (att & RO)
    field[i++] = 'R' ;
  if (att & HIDDEN)
    field[i++] = 'H' ;
  if (att & SYSTEM)
    field[i++] = 'S' ;
  if (att & DIRECTORY)
    field[i++] = 'D' ;
  if (att & ARCHIVE)
    field[i++] = 'A' ;
  return field ;
}

/* Indent, but only up to a certain amount  */
Indent(n)
  int n;
{
  int i ;

  n *= IndentAmount ;

  if (n > MaxIndent)
    n = MaxIndent ;
  for (i=0; i < n; i++)
    putchar(' ') ;
}

getbits(x,p,n)  /* get n bits from position p */
  unsigned x,p,n ;
{
  return ((x >> (p+1-n)) & ~(~0 <<n)) ;
}

ParseCommand(argc,argv)
  int argc ;
  char *argv[] ;

{

  int c,i,j ;

  for (i=1; i<argc; i++)
  if ((argv[i][0] == '-') || (argv[i][0] == '/'))
  {
    c = ToUpper(argv[i][1]) ;
    switch (c)
    {
      case 'A' :
                flag.att = ON ;
                for (j=2; j <strlen(argv[i]);j++)
                /*   read all attribute bits specified after the '/A' on
                     the command line   */
                {
                    c = ToUpper(argv[i][j]) ;
                    switch(c)
                    {
                       case 'A' : attr |= ARCHIVE ;
                                  break ;
                       case 'R' : attr |= RO ;
                                  break ;
                       case 'D' : attr |= DIRECTORY ;
                                  break ;
                       case 'H' : attr |= HIDDEN | SYSTEM ;
                                  break ;
                    }
                }
                break ;
      case 'S' : flag.dir = ON ;
                 break ;
      case 'F' : flag.size = ON ;
                 break ;
      case 'D' : flag.date = ON ;
                 break ;
      case 'T' : flag.date = flag.time = ON ;
                 break ;
      case 'H' :
                 puts(USAGE) ;
                 puts(HELP1) ;
                 puts(HELP2) ;
                 puts(HELP3) ;
                 Exit(0) ;
                 break ;
      default  :
                 Fputs(USAGE,stderr) ;
                 Exit(1) ;
                 break ;
    }
  }
  else
    StrCpy(pattern,argv[i]) ;
}
```

MYFIND.C

for Better Software Filter Design

```
/* The two functions below are provided because the DOS function
   call to FIND files is lacking in two areas:

   1.  If you want to get files only with certain attributes, DOS
       will return files with that attribute, and all normal files;
   2.  It will not easily allow recursive calls to sub-directories.

   These two functions remedy this, by passing the FIND structure as part of
   the call, which DOS uses to store file information, and information needed
   for the FIND next.  Notice that MYFINDNEXT should only be called after
   MYFINDFIRST, and only with the same attribute setting.
```

```
*/
#include <stdio.h>
#define RO 1
#define HIDDEN 2
#define SYSTEM 4
#define DIRECTORY 16
#define ARCHIVE 32
#define NULL 0
/*   See DOS Technical Reference Manual, section on DOS function
     call 4EH, FIND FIRST, for documentation of FINDAREA structure.
*/

struct FINDAREA
{
  char reserved[21] ;
  char attribute ;
  unsigned time, date ;
  unsigned long size ;
  char name[13] ;
};

/*  Program Logic of MYFINDFIRST:

    1.  Set DOS Disk Transfer Area to the AREA structure.  DTA is needed for
        the DOS FIND function.
    2.  Call DOS FIND FIRST.  If error, set AREA to NULL.
    3.  If found file not of the proper attribute, call DOS FIND NEXT, until a
        proper file is found (also, ignore DOT and Double DOT files).
*/

struct FINDAREA *MyFindfirst(name,attr,area)
  char *name ;
  int attr ;
  struct FINDAREA *area ;
{
  REGS regs ;
  Bdos(0x1a,area) ;
  regs.ax = 0x4e00 ;
  regs.dx = (int) name ;
  regs.cx = attr ;
  if (IntDos(&regs,&regs) &1)
    area = (struct FINDAREA *) NULL ;
  while ((area != NULL) && (!(area->attribute & attr) && attr)
        || (area->name[0] == '.'))
  {
    Bdos(0x1a,area) ;
    regs.ax = 0x4f00 ;
    if (IntDos(&regs,&regs) &1)
      area = (struct FINDAREA *) NULL ;
  }
  return area ;
}

/*  Program logic of MYFINDNEXT:
    1.  Set DOS DTA to AREA.
    2.  Call DOS FINDNEXT until a file with the proper attributes is found.
*/

struct FINDAREA *MyFindNext(attr,area)
  int attr ;
  struct FINDAREA *area ;
{
  REGS regs ;
  do {
     Bdos(0x1a,area) ;
     regs.ax = 0x4F00 ;
     if (IntDos(&regs,&regs) &1)
       area = (struct FINDAREA *) NULL ;
     }
  while ((area != NULL) && (!(area->attribute & attr) && attr)
        || (area->name[0] == '.')) ;
  return area ;
}
```

ALONE.C

```
/*
  Function to determine if the calling program's output is the console, or is
  in a filter
*/

#include <dos.h>
standalone()
{
REGS regs ;
int flag ;

regs.ax = 0x4400 ;          /* DOS function number for IOCTL */
regs.bx = 1    ;           /* standard output */
flag = intdos(&regs,&regs) ;
regs.dx &= 0x80 ;          /* check Bit 7 */
return (regs.dx == 0x80) ;
}
```

Listing 2 For Better Software Filter Design

```
#include <stdio.h>

  ParseCommand(argc,argv)
     int argc ;
     char *argv[] ;
  {
        char filename[80],file2[80] ;

     /*  Normally expects the standard input, but user can specify
         file name on command line, or have program prompt for
         filename.  */
     if (InAlone())  /* Then no redirected input  */
     {
        switch(argc)
        {
        case 3 :
              strcpy(file2,argv[2]) ;
```

plied filename. With ParseCommand(), a programmer can write the main program operating on only the standard input and output, and not worry at all about the I/O. For example:

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

{
  char c ;

  ParseCommand(argc,argv) ;
  while (gets(str) != EOF)
  {
/*  To be whatever the filter processes  */
    Process(str) ;
    puts(str) ;
  }
```

Adding ParseCommand to a filter program adds three user interfaces to the same program. First, it can be a filter program, to be used in pipes. Second, it performs like a command, with the user specifying files on the command line. Last, it presents a prompt interface, for the more casual users who only know the name of the program. ParseCommand can be different for each application, as it may need to parse flags or options on the command line, or give additional prompts to the user, depending upon the application. ParseCommand could easily present a forms interface, or a 'point-and-shoot' method to choose files. The point is that the main application need not deal with these details. All the main applications needs to operate on is what it thinks to be the standard input and output.

The UNIX system, popularized filter programs. It is even conjectured that the redirection and piping features are the central features of UNIX. While MS-DOS has taken many of the same features from UNIX, its built-in commands and programs do not conform to the same standards, rendering piping less functionality. DOS does this to obtain the advantage of more readable, thus more useful to the casual user, output from commands. The simple functions **standalone** and **inalone**, which use only a single DOS call, solve the problem of filter programs being too terse, while maintaining their functionality. Users writing their own programs, and programmers creating public domain and commercial software are encouraged to incorporate these functions in their programs, so their programs can be used in piping **and** for casual users.

References

(1) Brian W. Kernighan & Rob Pike, *Unix Programming Environment,* (Prentice-Hall, New Jersey: 1984), p.101

(2),*ibid.*, p.130.

(3) Gordon Blair, Jon Malone, & John Mariani, "A Critique of Unix" *Software- Practice and Experience*, V.15, #12 (December 1985), p.1132.

(4) *ibid.*, p.1135.

(5) *ibid.*, p.1133.

Listing 2 Continued

```
    case 2 :
          strcpy(filename,argv[1]) ;
          break ;
    case 1 :
          fprintf(stderr,"Which file to convert? ") ;
          if (strlen(gets(filename)) == 0)
             exit(1) ;
          if (StandAlone())
          {
            pfrintf(stderr,"Output to where? ") ;
            gets(file2) ;
          }
          break ;
    default :    /* argc > 3 */
          fprintf(stderr,"ERROR. Usage: %s file newfile\n",progname);
          fprintf(stderr,"       or     %s <file >newfile\n",progname;
          exit(1) ;
    }
    if (freopen(filename,"r",stdin) == NULL)
    {
       frpintf(stderr,"Cannot open '%s'\n",filename) ;
       exit(1) ;
    }
    else
       if (strlen(file2) != 0)
          freopen(file2,"w",stdout) ;
  }
}
```

■ ■ ■

Own The Source Code

We have C source code for public domain versions of most UNIX programming tools.

Control your environment wherever you work!

Over 100 volumes...

Quarterly newsletter.

C and UNIX Books.

200 page indexed source code directory.

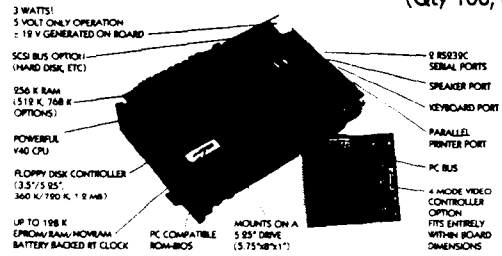
Write or call
C Users' Group
PO Box 97
(316) 241-1065

UNIX ... of AT&T Bell Laboratories

Reliable, Cost Effective Solutions for Computerization

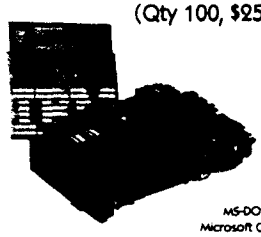
A Motherboard and 4 Expansion Cards in the Space of a Half-Height 5-1/4" Disk Drive!

from **$329**
(Qty 100, $252)



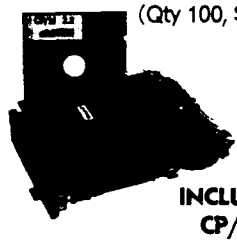
Little Board™/PC
World's smallest PC — and CMOS too!

$329
(Qty 100, $252)



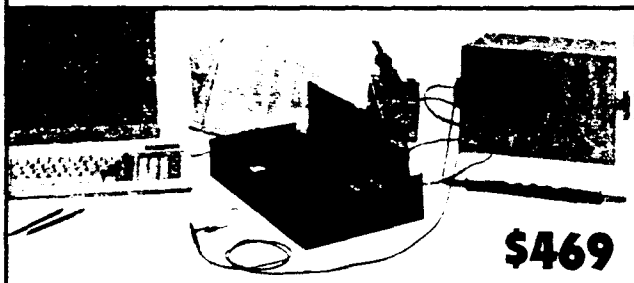
Little Board™/186
High performance single board MS-DOS system.

$159
(Qty 100, $124)



INCLUDES CP/M!™ DRI

Little Board™
World's least expensive single board system.



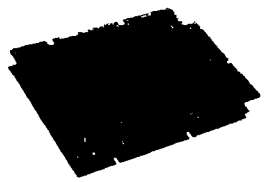
$469

Development Chassis/PC™
"Known Good" PC bus project development environment for Little Board/PC (not included).

$99

Project Board/186™
Prototype adapter for 80186 based projects and products.

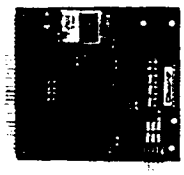
$89



Project Board/80™
Prototype adapter for Z80 based projects and products.

Z80 ™ ZILOG

$159

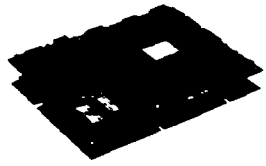


CMOS Video Controller
4-mode CMOS video controller for Little Board/PC.

from **$179**

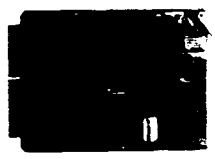
SCSI Memory Controller
SCSI controller for fixed and removable volatile and non-volatile semiconductor memory.

from **$149**



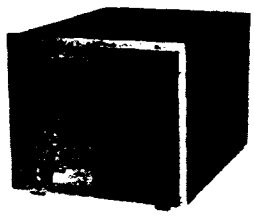
Expansion/186™
Multi-function expansion for Little Board/186. I/O, Serial, RAM, and Math Options.

$129



SCSI/IOP™
STD bus I/O expansion adapter for any SCSI host system.

from **$495**



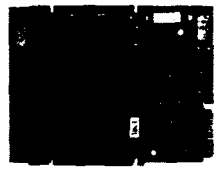
Bookshelf™
System Modules

- PC compatible Little Board/PC systems.
- Single and Multi-User Little Board/186 systems.
- Little Board/Plus CP/M systems.
- SCSI disk and tape expansion modules.
- Floppy drive expansion modules.

$395

Concurrent PC-DOS™ DRI
Multi-user, multi-tasking operating system for Little Board/186 supports up to four users.

$99



SCSI Z80 Host Adapter
SCSI host adapter for any Z80 system.
Plugs into Z80 socket.

Distributors • Argentina: Factorial, S.A. 41-0018 • **Australia:** Current Solutions (03) 227-5959 • **Brazil:** Computadores Compuleader (41) 262-4866 • **Canada:** Tri-M (604) 438-0028 • **Denmark:** Danbit (03) 66 20 20 • **Finland:** Symmetric OY 358-0-585-322 • **France:** Egal Plus (1) 4502-1800 • **Germany, West:** IST-Elektronik Vertribes GmbH 089-611-6151 • **Israel:** Alpha Terminals, Ltd. (03) 49-16-95 • **Spain:** Hardware & Software 204-2099 • **Sweden:** AB Akta (08) 54-20-20 • **UK:** Ambar Systems, Ltd. 0296 435511 • **USA:** Contact Ampro

AMPRO
COMPUTERS, INCORPORATED

67 East Evelyn Avenue • Post Office Box 390427, Mountain View, CA 94039 • (415) 962-0230 • TLX 49-0302

MDISK

Add a One Megabyte RAM Disk to Ampro L.B.

by Terry Hazen & Jim Cole

Introduction

MDISK is a printed circuit board, containing sockets for up to 1 megabyte of 256k RAM chips, that plugs into the Z80 socket on your AMPRO Z80 Little Board. The MDISK software drivers, when added to the AMPRO BIOS source code, enable the Little Board to use the additional RAM as a solid state disk drive, complete with a system track. MDISK utilizes the additional memory by swapping 32k pages in and out of the lower 32k of system memory while the upper 32k, containing the operating system along with the MDISK drivers, remains a stable "global" memory segment.

AMPRO's SYSGEN utility can be used to copy the system track from your AMPRO boot disk to the system track on MDISK, and the SWAP utility will allow you to make MDISK your logical drive A so that very fast warm boots can be made from the MDISK system track. The ZCPR3 PATH utility can be used to optimize your file search paths for use with MDISK.

In part one of this series, we will discuss the modifications to the Little Board that are required in order to allow the MDISK hardware to address and refresh the extended RAM. In part two we will cover the software driver routines that are added to the AMPRO BIOS in order to enable the extended RAM to emulate a disk drive.

Although MDISK was designed specifically for the Little Board, it could probably be used with many other Z80 computers, depending on their specific hardware design and the structure of their BIOS. At this time, however, MDISK has not been tested with machines other than the AMPRO Z80 Little Board.

Why use a RAM Disk?

Since Z80 computers only have a 64k memory space to work with, many programs use the disk drive a lot to swap program overlays in and out of memory. This process can be quite slow, especially if you are using floppy drives. The major advantage of using a RAM disk as your operating workspace is its greatly increased speed when you are performing operations requiring a lot of disk access, like word processing, program development, or database manipulation. Files resident on a RAM disk are accessed at memory speed, while accessing files on floppy or hard disks is relatively much slower.

For example, using a 4 mhz Little Board and a 96tpi floppy disk drive with a 3ms head stepping rate, the ACOPY file copy utility can copy and verify a 100k file to another filename on the same disk in about 91 seconds. The same operation on the MDISK RAM disk takes about 18 seconds, almost one-fifth the time. On the floppy disk drive, WordStar and the same 100k file can be loaded and the file ready to edit in about 26 seconds, while on MDISK they are ready to go in under 4 seconds!

Your computer work habits will probably change a lot when you have this kind of fast response at your fingertips. When doing assembly language programming, for instance, you can modify the source code, assemble and test it, and go back and repeat the cycle again so quickly that it becomes almost interactive. You can also use the ZCPR3 utilities VALIAS and SAK to set up a recursive ALIAS (that is, an ALIAS that calls itself) to create such a cycle, making the process even easier and faster.

Programs seem to run so fast that it becomes very fast and easy, for example, to exit WordStar, run another program or look up material in another text file, then go back into WordStar almost as quickly as you can write about it. The Little Board isn't running any faster, you just aren't waiting around for the disk drives to do their work anymore. It is very hard to go back to a floppy disk system after using MDISK for a while, and the system that seemed perfectly adequate before now seems horribly slow and inconvienient.

The Price of Progress

Everything has its price, however, and MDISK is no exception. Only the later model 1B Little Boards have a provision for a high speed bus, the SCSI port. SCSI extended RAM boards are offered by AMPRO and while they are undoubtedly the easiest and most elegant way to obtain Z80 extended RAM, they are relatively expensive. We have chosen instead to access the Z80 directly. Our approach is to remove the Z80 from its socket and plug in a printed circuit board containing the required interface circuitry, the extended RAM chips, and a new socket for the Z80. This approach requires a little hands-on modification of the Little Board, but is considerably less expensive and can be used with all Z80 Little Boards, even those without SCSI capabilities.

Since MDISK will contain a number of 256k dynamic RAM chips that will replace the 64k chips originally resident on the Little Board, you will need to remove the 64k chips from the Little Board, replace at least one of them with a socket, and you will need to add one jumper. A 16 pin ribbon DIP cable from this Little Board RAM socket to the MDISK board will allow access to the normal 'RAM BUS' signals on the Little Board so that we don't have to duplicate the circuitry that provides them. If all 8 RAM chip positions are socketed, the Little Board will operate normally when the 8 64k RAM chips and the Z80 are present.

Another part of the price that must be paid is the susceptability of the data contained on the RAM disk to problems on the AC power line. Since MDISK becomes an integral extension of the Little Board, the whole Little Board/MDISK combination, or at least the 5 volt power to both boards, must be battery backed-up to avoid power outage problems. We have not taken that approach at this time, choosing instead to knowingly use the MDISK RAM disk as a volatile workspace and making use of ALIAS files, archiving (under ZRDOS), and ACOPY to help us back up the working material often during the work sessions. This approach has worked quite well in practice.

Ampro Little Board Modifications

With care, removing the 8 64k RAM chips on your Little Board and replacing them with sockets is not really difficult. If you don't have experience with soldering and desoldering on printed circuit boards, you might wish to review James O'Connor's article, "Repairing and Modifying Printed Circuits" and Bill Kibler's column "The Computer Corner," which deals in part with removing and replacing ICs. Both articles appear in issue #25 of TCJ.

Removing the RAM Chips

The first step is to identify the 8 chips you need to remove. They are the 64k RAM chips, and are marked "4164." On the Little Board 1A (the original Little Board), they are chips U21-24 and U30-33. On the newer Little Board 1B (with provisions for SCSI), they are chips U19-21 and U30-34. Remove these chips by cutting off each chip lead close to the PCB with close cutting snips. You can then use a good solder sucker to remove the lead and the solder at the same time, provided that the lead is straight and the solder sucker is kept clean and clear. When all the leads have been successfully removed, check all the holes and clean them out with a solder sucker if necessary.

Adding Sockets

You will need to add one socket to the Little Board in order to access some of the signals needed for RAM control, but as long as you need to add one socket, you might as well socket all 8 positions so that you can have your Little Board work as it did originally if necessary. If you wish to only add the one required socket, the prefered position is U21 on both the model 1A and model 1B.

Use good quality 16 pin sockets. Low profile, double-wiping sockets that make contact on both sides of the IC leads are recommended. Tack all the sockets in place first, soldering only the two diagonal corner leads on each socket. Then you can apply finger pressure to the socket and re-melt one corner of each socket at a time, which will release the socket to seat itself fully. Then you can carefully solder the remaining leads. Check for good connections and any solder bridges. Clip off the leads close to the board if they are too long.

Adding the Jumper

Add a jumper on the back of the Little Board from pin 1 of the RAM socket that you will use for your 16 pin ribbon DIP jumper cable (for example, U21 on either the model 1A or 1B) to pin 1 of either of the two 74LS157 multiplexer chips (U20,29 on model 1A, U22,23 on model 1B), whichever is closest. This jumper will extend the multiplexer select line to the MDISK board, where it will be used to help generate the extended chip addressing.

Flux Removal

Use a liberal quantity of flux remover spray to remove all residual flux from back of the Little Board, following the directions on the can. To avoid problems with the fumes, work outdoors if at all possible. When you are done, there should be no traces of flux on the back of the board at all. Try to keep the remover off the front of the board, as you can't get under the sockets and other components well enough.

Testing the Modified Little Board

Check to make sure your Little Board still works properly by inserting a new set of 8 64k RAM chips (200ns is fine) in the sockets. Put the Little Board back in its case, connect everything up, and boot a disk. If all is well, everything will run normally. Try running some programs you are familiar with to be sure everything is OK.

The MDISK Printed Circuit Board

The MDISK board consists of seven basic functional sections:

1. Z80
2. I/O port decoder and latch
3. Refresh generator
4. Memory chip selection
5. A15 address generation
6. A16/A17 address generation
7. 1 to 4 banks of 256k RAM chips and associated buffers

Let's look at each section, and then see how they are tied together to function as a single unit.

The Z80

The Z80 is removed from its socket on the Little Board and plugged into a socket on the MDISK board. The MDISK board has a 40 pin board-to-board connector that plugs into the Z80 socket on the Little Board. In this way, we can intercept the Z80 signals that we need to modify in order to create extended RAM addressing, and then pass on the modified signals to the Little Board.

The I/O Port Decoder and Latch

The Z80 processor has 256 possible I/O ports. The AMPRO Little Board uses several of these for various functions, such as the floppy disk controller, serial ports, printer port, and on the later boards, the SCSI port. We chose to use port 30H as the MDISK control port as it is not used by the Little Board hardware.

In operation, the Z80 processor executes an OUT 30 instruction and puts the contents of the A register on the data bus. Two comparators, U1 and U2, monitor the address lines, and NOR gate U3c monitors the Z80 WR* and IORQ* lines, watching for the proper port address and generates a high pulse whenever an OUT 30 instruction is executed. This pulse will cause 8 bit latch U4 to latch and hold the A register contents from the data lines. This data will not change until the next OUT 30 instruction is executed or the reset button is pressed.

These 8 data bits are used to generate the extra address bits needed for the 256K RAM chips and to control the MDISK drive select LED. D0 is used as A15 when addressing the lower 32K of the Z80 address space. D1 and D2 are used to generate A16 and A17 for the RAM chips. D3 and D4 generate the RAM chip selection for Bank 1, 2, 3 or 4. D7 is used to turn on and off the MDISK "drive select" LED, which is under software control and is used only to help the operator monitor disk operations in progress. The D5 and D6 bits are not currently used.

Editor's Note:

The IEEE standard uses a new notation for logical and electrical state relationships which is convenient with word processors. The overbar is no longer used to indicate an active low signal state because it is difficult for many word processors and phototypesetters to set it. The suffix "" is used instead to designate that an electrical signal is active low, as in the Z 80 WR* above.*

The Refresh Generator

The Z80 was designed to work with dynamic RAM chips and is able to generate 7 bits of refresh internally and gate them onto the address bus at the proper times. This will support the full Z80 address space of 64K, which is normally adequate. 256K dynamic RAM chips, however, require 8 bits of refresh to retain their contents. While the Z80 takes care of generating the refresh signals for A0-A6, the MDISK refresh generator circuit intercepts the Z80 A7 line and 8 bit counter U10 counts the refresh cycles, setting its output high for 128 cycles and then low for 128 cycles. During refresh cycles, U9 adds the output from U10 to the Z80 A7 signal in order to provide the 8th bit refresh signal for the RAM chips. During non-refresh cycles, U9 passes the Z80 A7 signal on unchanged to the address bus.

RAM Chip Selection

The chip select logic uses the CAS* signal required by the dynamic RAM chips. If a RAM chip receives a normal read cycle (except for the CAS* signal), it accepts it as a refresh cycle. If the CAS* signal is present, a memory read/write cycle is processed. The CAS* signal is generated by logic on the AMPRO Little Board and gated to the selected MDISK RAM chip.

Address bit A15 is used to differentiate between the 32k global page of RAM in high memory and the various banks that MDISK swaps in and out of the lower 32k of Z80 address space. If A15 is low, bits D3 and D4 in latch U4 are gated to quad AND gate U5, selecting the bank to be swapped into the lower 32k. If A15 is high, addressing the upper 32k, the outputs of U5 are all forced low, making the high bank always Bank 1 Page 2 of memory. This provides a stable global base in the upper 32k of the Z80 64K address space for the operating system and RAM disk control.

The outputs of U5 are fed into 2-4 decoder U7 as shown in Figure 1. The outputs from U7 are fed to quad OR gate U8 along with the CAS* signal from the Little Board. Notice that there will always be a logical 0 on one of the U8 OR gates. That gate will send the AMPRO CAS* signal to the selected bank of 256K RAM chips for the read/write operation. The other banks will process the cycle as a refresh.

Figure 1

(U7 Input) Pins: 2 3	(U7 Output) 4 5 6 7
0 0	0 1 1 1
0 1	1 0 1 1
1 0	1 1 0 1
1 1	1 1 1 0

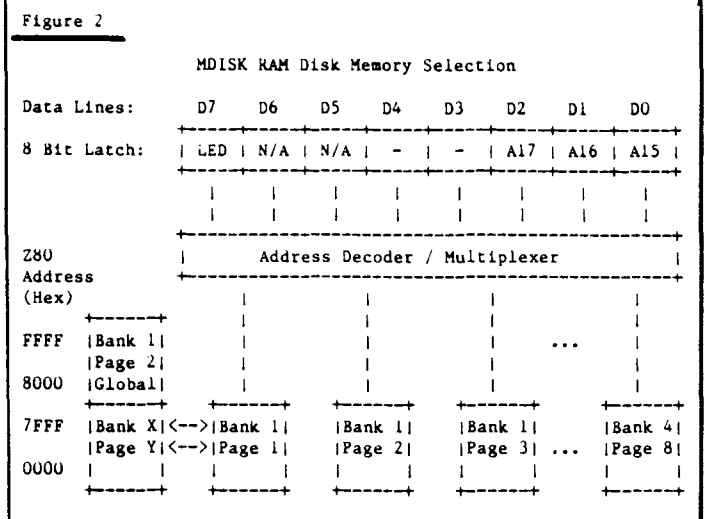
A15 Generation

The Z80 A15 line is intercepted and regenerated by U3 and the D0 output of latch U4 in order to control all of the extra extended RAM chip addressing. When it is high, the A16/A17 and the bank select logic are all forced to a logical 0 state, selecting the upper 32k of the Z80 address space, Bank 1 Page 2. All of the control software is executed from this area, simplifying the software and management of the memory banks. The system of memory banks is shown in figure 2.

A16 and A17 Generation

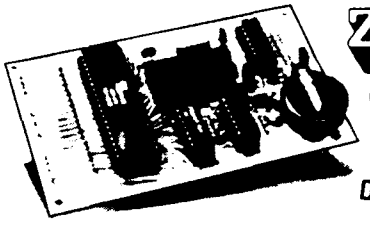
The A16 and A17 address lines are generated from data bits stored in latch U4. D1 and D2 become the new address bits for the extra RAM chips, provided A15 is low. These two data bits are gated through the other two sections of AND gate U5 to multiplexer U6.

Figure 2



Bank X and Page Y are selected by the value in the 8 bit latch. Any bank can be selected, including the global Bank 1 Page 2.

Dynamic RAM chips use a multiplexed address line to decrease the number of pins required to address a large memory space. The first half is called the row address strobe (RAS), and the second half is called the column address strobe (CAS). During the RAS cycle, U6 will output the A16 bit and during the CAS cycle, it will output the A17 bit, thus providing the 256k RAM chips with the required A8 signal. This is coordinated with the other address lines by using the multiplexer select signal from the



Ztime-I

CALENDAR/CLOCK

$69 KIT

NOW WITH FILE DATE STAMPING!

- Works with any Z-80 based computer.
- Currently being used in Ampro, Kaypro 2, 4 & 10, Morrow, Northstar, Osborne, Xerox, Zorba and many other computers.
- Piggybacks in Z80 socket.
- Uses National MM58167 clock chip, as featured in May '82 Byte.
- Battery backup keeps time with CPU power off!
- Optional software is available for file date stamping, screen time displays, etc.
- Specify computer type when ordering.
- Packages available:

Fully assembled and tested	$99.
Complete kit	$69.
Bare board and software	$29.
UPS ground shipping	$ 3.

MASTERCARD, VISA, PERSONAL CHECKS, MONEY ORDERS & C.O.D.'s ACCEPTED.

N.Y. STATE RESIDENTS ADD 8% SALES TAX

KCT

KENMORE COMPUTER TECHNOLOGIES

P.O. Box 635, Kenmore, New York 14217 (716) 877-0617

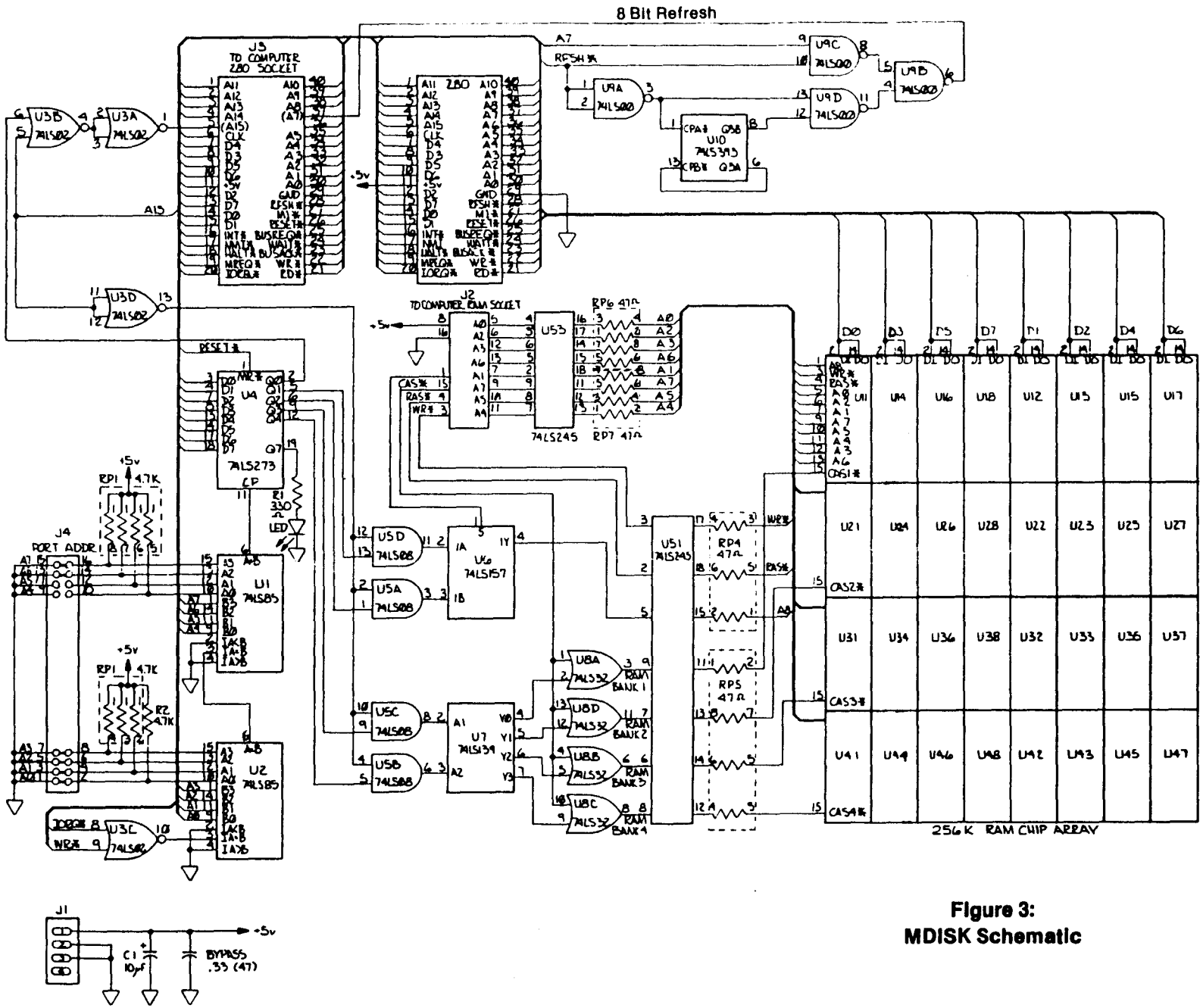


Figure 3:
MDISK Schematic

EVERYTHING YOU NEED... $279.00

Now it's easy to program the Heath-Zenith HERO-1® Robot with an Apple® II. HERO® Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics. The HERO® Macros come with 30 pages of documentation.

[For example, the line: 1130 > MVWRIM GRIP, OPEN, 60, FAST instructs the HERO® to open his gripper 60 units at fast speed. Motor position is expressed in base-10.]

Transfer to HERO® with ROBI... an affordable interface for the robotics experimenter... is simple.

- **ROBI is a complete package. No additional hardware required for Apple® or HERO®.**
- **ROBI installs quickly in an Apple® II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO®.**
- **With ROBI and the Cross Assembler, the programmer uses Apple®'s memory to write the program, and HERO®'s memory to run the program.**
- **Not "copy protected," archival copies may be made as needed.**
- **ROBI offers expansion potential.**

VISA and MasterCard accepted.

BERSEARCH Information Services

26160 Edelweiss Circle
Evergreen, Colorado 80439

The Cross Assembler with HERO® Macros sells for $100.00; the ROBI Interface sells for $199.00. Both as a package — $279.00.

To order, or for more information, call (303) 670-6137.

APPLE® is a trademark of Apple Computer HERO® is a trademark of Heath Electronics

Little Board, which we have jumpered to pin one of the Little Board RAM socket we are using for our RAM interconnect cable.

RAM Banks

The MDISK hardware will address up to four banks of 256k RAM chips located on the MDISK board. This will give us up to 1 Meg of additional RAM. Some of the Little Board signals required for RAM operation, such as WR*, RAS*, and the multiplexed address lines A0-A7, are obtained from the Little Board RAM socket via the 6 inch 16 pin ribbon DIP interconnect cable that is plugged into MDISK socket J2, as they are not directly available at the Z80 pins. The interconnect cable signals are buffered by U51 and U53 before being sent to the RAM chips.

Operation

We have now created a Z80 that has a "window" in the lower 32k of its 64k address space. This window can be filled with any 32k page of the extended RAM by loading the proper byte into the control port, as shown in figure 2. The upper 32k of memory is global and never changes, giving us the consistant work area we need for the operating system and the routines necessary to use the hardware as a RAM disk.

Connecting It All Up

Remove the Z80 from the Little Board and plug it into the Z80 socket on the MDISK board. Plug one end of the 6 inch 16 pin ribbon DIP cable into the Little Board RAM socket that you have wired the jumper to, with the cable tail extending toward the rear edge of the Little Board (the edge with all the connectors on it). Carefully plug the MDISK 40 pin board-to-board connector into the Little Board Z80 socket, plug the free end of the 16 pin ribbon DIP cable into J2 on the MDISK board. The MDISK board requires 5 volts at 0.60 amps, which is supplied through J1, a standard 5¼ inch disk drive power connector. This is the same type connector used by the Little Board. If your power supply doesn't have any extra plugs, a standard disk drive 'Y' power connector can be used to supply both the Little Board and MDISK boards from one power supply plug.

Next Time

In part two we will present the software drivers that are added to the AMPRO BIOS in order to enable the Little Board to use the new extended memory as a solid state disk drive. These drivers must be added to the AMPRO BIOS source code, which is available directly from AMPRO, and is a worthwhile investment for anyone interested in learning about the inner workings of their operating system or in customizing or optimizing their system. ■

Using the Hitachi HD64180

Embedded Processor Designs

By Kenneth A. Taschner & Frederick B. Maxwell

Electronic Technical Services

Jon Schneider gave us an introduction to the HD64180's features and programming in his two part series "The Hitachi HD64180: New Life for 8 Bit Systems," Computer Journal issues #27 & 28. We will try to build upon this by first covering the highlights of the part, followed some of the problems we must work around (virtually every complex piece of silicon or software has some bugs). In future articles, we will compare it to its competition, both 8 & 16 bit, for use in embedded processor design and we will cover the various versions of the 64180/Z180 and correct methods of interfacing to memory and peripherals.

The popularity of the Intel 8080/85 series, the Zilog Z80 series, and the newer National Semiconductor NSC-800 as embedded processors has established the viability and desirability of this basic, 8-bit architecture as a problem solving tool. While speed improvements have been made in both the Intel and Zilog processors since their introduction so many years ago, until the unveiling of the HD64180 by Hitachi, this powerful architecture had not been updated with the introduction of a compatible, big-brother processor with additional features and/or powerful, on-board peripherals. The introduction of the Hitachi HD64180 has provided the stepping stone that designers have been looking for and that, up to now, has been unavailable. With this processor, Hitachi has given new life to many older systems. The HD64180 allows engineers to upgrade existing designs without a major software re-write, to add additional features that require more memory addressing or speed than the older 8-bit processors were capable of, and, most importantly, allows the engineer to design in a processor that has an architecture and instruction set that is familiar to his software staff.

The HD64180 is a desirable, supported part with many features that will cause it to be a major force in the embedded processor market. Being a CMOS chip, its power consumption is an enviably low 75mw at 6mhz clock speed. Its two ASCI (Asynchronous Serial Communications Interface) ports with built in baud rate generators provide the communications to the outside world that many embedded processor systems require. In addition the HD64180 has a third serial port which is intended as a high-speed inter-processor communications channel for multiple processor designs. Lastly, its extremely low cost will make it a serious contender where cost is an important consideration.

A Family Affair

The HD64180 is not an orphan. Its basic design will be made available as a standard ASIC (Application Specific Integrated Circuit) cell. An ASIC is a semi-custom part which consists of building blocks (cells) that can be put together by a hardware engineer to produce an IC customized to his design's requirements. This allows a company to reduce the parts count, assembly cost, debugging cost, repair cost, and physical size of its product by getting a large percentage of the parts needed in a single IC package. Unlike a hybrid circuit, an ASIC is a single piece of silicon. This allows an engineer to prototype a design using an HD64180 and to reduce the design to an ASIC package with few extraneous parts.

The microcomputer market is not being ignored either, where Hitachi intends to package the HD64180 with RAM, ROM, EEPROM, I/O, and possibly A/D converters for sale as part of their growing family of single-chip microcomputers. This will allow an off-the-shelf part to be used in designs where an ASIC's initial cost cannot be justified and where individual components could consume too much board real estate (space) or could make the product too costly to produce.

ALMOST 100% Object Code Compatible

From an applications programmer's point of view, there isn't much to worry about when it comes to writing software, as most of the actual interfacing to the hardware has been done through the operating system or through BIOS calls. One problem that applications programmers must be aware of is a subtle difference between the Z80 and HD64180 when using the RLD or RRD nibble rotate instructions. The Z80 flags reflect the contents of the accumulator upon completion of the instruction, as we would expect, however, the HD64180's flags reflect the contents of the memory location pointed to by HL, an obvious error. The bug exists in all package styles in both the R0 & R1 masks of the HD64180. The Zilog equivalent to the Hitachi HD64180, the Z180, is still in ALPHA test and, at the time that this is being written, it is not known if this bug has been fixed. Fortunately, this instruction is used to manipulate BCD type data and rarely are the flags checked after its use. If you must make a decision based on the results of one of these rotate instructions then OR the accumulator with itself to correct the flags. Please don't make the mistake of depending on this bug of the current 64180 in your software, as it may be corrected in a future release of the part, and, it guarantees that your code will not run on the new Z280 or the older Z80 or NSC-800.

Most of the problems that will need to be dealt with will occur to the engineer designing hardware or to the systems level programmer writing the BIOS or, in the case of embedded processors, the low level device drivers.

Wandering I/O Addresses

The first issue we will cover will be the "64K" of I/O address space. One of the many enhancements of the 64180 over its 8 bit predecessors (8080,8085,Z80,NSC800 etc.) is the enlargement of its physical I/O address space from 256 I/O ports to 65536 I/O ports. This at first glance would appear to be heaven sent, especially since the onboard peripherals use 64 I/O ports, but great care must be used to actually reap any benefit from these new-found I/O ports. Where does the 64180 actually get its upper 8 bits of I/O address from? Herein lies the problem. The typical OUT (PORT),A can have disastrous effects in a 64180! Why, you

ask? Well, the designers of the 64180 use the contents of the accumulator as the upper 8 bits (A8-A15) on the OUT (PORT),A instruction. To illustrate the problem let's look at typical I/O sequences and and their effect on the 64180's enviroment.

```
        Typical Z80 I/O

DATA            EQU     010H
PORT            EQU     022H

        LD      A,DATA          ; Get data byte to output to
port.
        OUT     (PORT),A        ; ...now send it.
```

In the 64180, using this same code sequence the I/O port selected will depend on the contents of the register being output! Remember, A15-A8 is equal to the contents of the accummulator and A7-A0 is equal to the port specified so, DATA = 10H and PORT = 22H the effective port address is 1022H! Now let's just change DATA to 80H. Now the port address we output to is 8022H! As you can see, this fairly common code sequence is useless on a 64180 using 64K I/O addressing. Is there a "work-around" for this design flaw? Fortunately, the answer is "yes." The register indirect OUT (C),A is actually equivalent to OUT (BC),A, so, just load the (BC) register with the actual physical address of the port to be output to. This can be done, because on a register indirect instruction A8-A15 reflects the contents of the B register!

```
        A Correct Method for 64180 64K I/O

WORD_ADDR       EQU     01022H  ; Port address
DATA            EQU     010H

        LD      A,DATA          ; Byte to output.
        LD      BC,WORD_ADDR    ; Set port address in <BC>.
        OUT     (C),A           ; ...now send it to port 1022H
```

This problem also exists on block I/O as seen in the following sequence:

```
    Typical Z80 Block Output to Port

    LD      HL,BLOCK_START  ; String to output to port.
    LD      B,NUMBER_BYTES  ; ...string length to output.
    LD      C,PORT          ; Port to output string.
    OTIR                    ; Output and repeat till done.
```

Again the problem is that the OTIR instruction is using the B register as its repeat loop counter and the 64180 uses the B register as the upper address. Therefore, if B = 5 and PORT = 22H the bytes would be sent as follows. I/O addr. = 0522H, data = (HL) then I/O addr. = 0422H, data = (HL + 1), etc.

```
        A Correct Method for 64180 Block I/O:

WORD_ADDR       EQU     01022H  ; Port address

        LD      HL,BLOCK_START  ; String to be output to port.
        LD      BC,WORD_ADDR    ; Set port address in <BC>.
        LD      D,NUMBER_BYTES  ; ...string length to output.
LOOP:   LD      A,(HL)          ; Get byte.
        OUT     (C),A           ; Output byte.
        INC     HL              ; ...next byte.
        DEC     D               ; ...one less to do.
        JR      NZ,LOOP         ; Keep going until string is
complete.
```

There are hardware solutions to these problems, of course. One could set up an I/O port to act as an I/O page register for the upper address lines but the 64180 would not recognize this so we would lose 64 I/O addresses in each page to avoid conflict with the 64180's internal I/O ports. Another approach would be to memory map the I/O, but, as microcomputer manufacturers learned long ago, no matter how little space this memory mapped I/O uses, someone will think they need the memory. Fortunately, it is unusual for a system to need so much I/O address space that 256 ports are not enough, so designing for 256 ports and preserving the ability to use the Z80 and 64180 I/O instructions (yes you can use the new I/O instructions on any I/O port in the 256 I/O port address) is usually the way to go.

64180 - ASCI Ports - Goodbye DART!

The 64180 has 2 on-board, independent, full duplex ASCI ports with independant baud rate generators and control lines. Although both channels are very similar, there are some minor differences, primarily in the modem control lines. The ports share the following characteristics:

```
o Full Duplex Communication
o 7 or 8 bit data length
o 1 or 2 stop bits (see text about bug)
o 9th data bit for multidrop communications (hardware
   must be configured for party line communications)
o Odd, even, no parity
o Programmable baud rate generator or optional external
  clock
o Modem control signals:
      Port0 :DCD0 -Data Carrier detect -(see text about bug)
             CTS0 -Clear to Send
             RTS0 -Request to Send

      Port1 :CTS1 -Clear to send
o Programmable interrupts (see text on precautions)
o DMAC Operation - To be decussed in a future artical
o Double buffered receiver and transmitter
```

As Jon Schneider suggests in his articles, it saves a lot of typing to generate include files to define absolute I/O addresses and then to refer to them using the same symbolic form as Hitachi demonstrates in their data book. We use a slightly different method than Mr. Schneider does. See the example below:

```
;       ASCI - Asynchronous Serial Communications

CNTLA0  EQU     BASE+0          ; ASCI Control REG A Chan 0
CNTLA1  EQU     BASE+1          ; ASCI Control REG A Chan 1
etc.

Usage:

BASE    EQU     0               ; Location of 64180 I/O Ports for
this
                                ; design.
        INCLUDE 180PORTS.MAC
```

We use this approach so that the same include file is used for all of our designs regardless of where we locate the 64180's I/O ports.

Before we start discussing the serial port registers and simple programming of the ASCI channels under programmed I/O and interrupt let's cover the two problems on which we receive the most phone calls.

Stop Bits

Most commonly used UARTS on the market will allow the user to specify whether to use 1 or 2 stop bits. What stop bits specify, in reality, is the delay between characters. UARTS that support this feature, if programmed for 1 stop bit, will wait 1 bit time at the end of the character being transmitted before starting the next character transmission. If 2 stop bits are requested they will insert 2 bit times before sending the next character. Regardless of the number of stop bits specified, the vast majority of UARTS only require 1 stop bit on a received character. There is a not so obvious advantage to this: Most systems will use only 1 stop bit above 300 baud. This decreases the communications time by approximatly 10%. In many cases this savings isn't actually

realized due to proccessing overhead on one or both sides of the link. Some computers are fast enough to keep the UART full so there is no break in the transmission stream, but, many "intelligent" peripherals are not quick enough to get the character, process it, determine whether to try to stop the transmitter by toggling a handshake line, etc. These devices (most notably CRT's and PRINTERS) gain the extra time they need when the transmitter inserts an extra stop bit. Therefore, a common practice, with fast I/O processors, is to send 2 stop bits and receive 1, being compatable with virtually all peripherals.

The 64180 ASCI channel, although technically correct, requires 2 stop bits on receive if the transmitter is programmed for 2. If the ASCI port on the 64180 is programmed for 1 stop bit it will work with most devices whether they are programmed for 1 or 2 stop bits (preferred usage). The exceptions to this, being the aforementioned intelligent terminals and printers which fail to operate at higher baud rates (9600 and above) without delays. If the 64180 is programmed for 2 stop bits the device it communicates with MUST always send 2 stop bits or the 64180 will misinterpret some characters. Hitachi has indicated that future versions of the 64180 (available Spring of 1988) will conform to the same conventions used by other industry-standard UARTs so, until then, it is recommended to only use 1 stop bit.

Data Carrier Detect - Stop Interrupting Me!

DCD0* (Data Carrier Detect Channel 0 NOT) is a status line into ASCI channel 0. If this line is in the normal (logic 0) state, channel 0's receiver behaves normally. If the DCD0* changes to a disconnected (logic 1) state, ASCI channel 0's receiver is disabled. It will not be re-enabled until DCD0* returns to a logic 0 AND STAT0 is read. The first time STAT0 is read, it will indicate that the device is disconnected and will cause the ASCI to re-evaluate the DCD0* line condition. If DCD0* indicates that there is a device present, the receiver is immediately activated. When STAT0 is re-read, the actual status of the DCD0* line will be indicated, so remember, if DCD0* indicates that there is no device connected, re-read STAT0 if you need to know the actual status of DCD0*.

The reason that this operates in this manner is to prevent the processor from interpreting line noise as legitimate data when no device is connected and this works fine as long as the serial data is gathered as programmed I/O. Where a potential problem can arise is using interrupts with ASCI channel 0. Whenever DCD0* is high, an interrupt will be generated. This is the same interrupt generated by the channel 0 receiver, so, before a character can be read, it is the programmer's responsibility to read STAT0 and assure that the interrupt was not generated by a change in the DCD0* line. If it was caused by the DCD0* line and interrupts are re-enabled before the DCD0* line returns to a logic 0, we will be hit by another interrupt immediately. This will also happen if the interrupts are re-enabled before we read STAT0 and determine that the DCD0* problem has been corrected. This is because this is a level sensitive interrupt. It will interrupt whenever the level is a logic 1, not just once, as an edge sensitive interrupt would. Obviously, the stack will get blown away if we enable and get interrupts before returning from the one we're servicing! In a correctly designed RS-232C interface, line noise is not a serious problem, so the obvious solution to this problem is to simply ground DCD0*, preventing this disastrous problem from ever occurring. A much more intelligent design would have been to generate an interrupt on a change of state of DCD0* rather than on a level.

Future Installments

In upcoming articles, we'll discuss programming the serial ports, memory interfacing to the HD64180, clock speed vs. baud rate, and more! ■

Editor's Note: We want to expand this HD64180 section, so send any tips, ideas, questions, or answers. We would also be interested in hearing about what you are doing with the '180, especially any unusual applications.

North American One-Eighty Group and Electronic Technical Services

Hardware and Software support for Z80/HD64180/Z280 Computers

- **The DateStamper:** Time and date stamping facility for Z-System and CP/M 2.2 files, provides creation, access and modification time and date, also supports a hardware-independent Real Time Clock interface. Includes many support utilities, even works with SB180's "heartbeat" clock.
  From Plu*Perfect Systems, $49.95 complete plus $3 s&h
- **HI-Tech C Plus:** Professional C compiler produces ROMable, reentrant code, runs under Z-System or CP/M, interchangeable Z80- and HD64180-optimized libraries with source, single-precision floating point support, assembler/linker, conversion utility for use with M80 and SLR assemblers. Function prototyping and 31-character names supported.
  From ETS: $195 complete plus $3 s&h
- **The One-Eighty File:** "The chronicle of the 8-bit renaissance" — the monthly newsletter of North American One-Eighty Group (NAOG) and ZCPR Systems Interest Group (ZSIG) keeps you up-to-date on the latest PD and commercial software releases, advanced 8-bit hardware developments, inexpensive PD disks, hardware savings, tips for programmers and users.
  From NAOG/ZSIG: $15 for 12 issues ($25 outside North America)
- **ETS-180-IO +:** Add-on board for SB180 provides two 115.2 kbps serial ports, SCSI interface, battery-backed Real Time Clock, 24 bits of parallel I/O, XBIOS, DateStamper, full Z-System support included.
  By ETS, $299.95 complete plus $15 s&h (includes NAOG membership)
- **XBIOS:** Flexible operating system for SB180 and SB180FX has cached disk I/O, expanded TPA, menu-driven installation. Includes DateStamper, utilities, supports Z-System and both MicroMint and ETS add-on boards.
  By Xsystems Software, $74.95 complete plus $3 s&h
- **Backgrounder II (BGII):** Full task swapping for Z-System and CP/M 2.2 computers, suspend current task without losing your place, supports two independent TPAs plus many always-available internal commands, includes Print Spooler, screen and function key drivers, much more.
  From Plu*Perfect Systems, $74.95 complete plus $3 s&h

Call 215-443-9031 during East Coast business hours
MasterCard & VISA accepted

Big news
for the brave
few of you
who started this
whole thing.

WORDSTAR®

CP/M® Edition, Release 4.

Finally, all the "if only's." Over 100 truly useful improvements including undo, macros, on-screen boldface and underline, and multiple ruler lines stored with documents. Even something to help you get the most from your laser printer. Everything you need to be at the forefront of technology. Again.

System requirements: CP/M-80 2.1 or higher; 54K TPA w/Math (51K TPA without Math). Disk requirements: two 5-1/4" DD drives, or two 8" drives, or one DD floppy drive and a hard disk.

To order WordStar, CP/M Edition, Release 4, fill in this coupon and send your check or money order to: MicroPro Order Update Department, P.O. Box 7079, San Rafael, CA 94901-7079. **Or call toll-free 800-227-5609 Ext. 762.** Allow 3-4 weeks for delivery.

Name_____
Address_____
City_____ State_____ Zip_____
Company Name_____
Telephone ( )_____

Osborne™ format 5-1/4" disks_____
Kaypro® format 5-1/4" disks_____
Generic 8" disks_____
Apple® format 5-1/4" disks_____ (available October '87)
WordStar Serial No._____ or include title page of your WordStar Reference Manual.

CP/M Release 4	$89.00
Tax*	
Shipping/Handling	$5.00
Total	

**Only these states require sales tax: CA, GA, IL, MA, NJ, NY, OH, TX and VA.*

WordStar and MicroPro are registered trademarks of MicroPro International Corporation. CP/M is a registered trademark of Digital Research, Inc. All other product names and trademark information are listed for purposes of description only.

© 1987 MicroPro

The ZCPR3 Corner

Announcing ZCPR33 & Z-COM Customization

by Jay Sage, Echelon, Inc.

One of the problems with writing a column for a magazine that only appears every two months or so is that so many things can happen between when one column is written and the next one is published. (Of course, there would be other, insurmountable problems if I had to turn out these columns every month, so I am not complaining.) At the end of the last article, I mentioned that ZCPR33 would probably be out by the time that issue appeared. Indeed, that was true. What I did not write, because no public announcement had been made yet, was that I had joined the Echelon team and would be the author of that version. So I am now wearing two hats, one as ZSIG software librarian and one as the Echelon team member in charge of command processor development. Richard Conn has gone off to do more esoteric things. In recognition of this change, I have broadened the title of this column.

ZCPR Version 3.3

Since ZCPR33, or Z33 as I will call it for short, is too exciting a subject to pass up, I will say a few words about it before continuing the discussion we began last time of techniques for customizing Z-COM. I will not say too much, however, since a lot of effort already went into preparing the 60-page "ZCPR33 User Guide." It has all the details and is available from either Echelon or Sage Microsystems East for $15 plus shipping ($3 from SME). Since the code, as in the past, is still available free of charge for personal, noncommercial use, sale of the manual is the only way other than OEM sales that we get any compensation for the enormous amount of effort that went into Z33. I will talk this time only about the design goals for Z33, and perhaps in future columns I will talk about some of the new features and capabilities.

Design Goals

In developing Z33, I tried to achieve five things (the number keeps growing every time it think about it). (1) I have tried to maintain a very high level of compatibility; (2) I have tried to increase flexibility, control, and speed; (3) I have tried to make the code rigorous and reliable; (4) I have tried to make more information about the internal state of the command processor accessible to user programs; and (5) I have tried to make the code readable and educational.

To the greatest extent reasonable, I have maintained compatibility between Z30 and Z33. No change need be made to any part of the operating system other than the command processor; the Z33 command processor, as I hinted at in the last column, can simply be dropped in wherever the old command processor was, either on the system tracks of the boot disk or in the appropriate places in a Z-COM system. No changes need be made in the memory allocations, and the officially released system modules that worked with Z30 will work with Z33 as well, though new, more powerful RCP and FCP modules were released with Z33 (the 'H' command in the unofficial experimental RCP145, because it made direct references to internal addresses in the CPR, will not work). All application programs and almost all utility programs will work unchanged with Z33. New utility programs have been written to take advantage of some of the new features in Z33.

Many features of ZCPR3 — such as automatic path searching for COM files, extended command processing, and error handling — are very convenient but can significantly slow system response. With Z33, the user is given greater control over these features from the command line so that unnecessary operations can be bypassed to save disk activity and time. No longer does the path automatically include the current directory first. The user can now, at his option, omit the current directory or include it in any position in the path. This has a dramatic effect on system speed. A command entered with a leading slash ('/') is handled directly by the extended command processor without wasting time searching the path for a COM file. For systems that take increasing advantage of ARUNZ or other extended command processing, speed is, again, greatly improved. Programs with what is called a type-3 environment are automatically loaded and executed at addresses other than 100H. By loading error handlers, shells, and extended command processors high in memory, user programs at 100H are left intact and can be reinvoked using the GO command.

The code in Z33 was almost totally rewritten, taking only the basic functions from Z30. Quite a few bugs, some very serious, were corrected, and new algorithms were used for many of the functions. A great deal of effort was devoted to making the code rigorous. No longer can a command tail longer than 128 bytes overwrite the program code and crash the system. No longer can command lines in SUBMIT files write beyond the end of the command line buffer. The root path and minimum path features now work correctly, so that duplicated elements in the search path do not have to be searched more than once. Extended command processing functions reliably and in combination with error handling.

The Z33 command processor makes much more information available about its operation. In Z30, only COM files that could not be found would invoke error handling. Z33 traps many different kinds of errors, and it can report the nature of the error to the user. Some examples are TPA overflow, disk full, bad numerical expression, incorrect password, bad directory specification, or ambiguous file specification. Z33 even makes it possible for user programs and routines in the resident command package to invoke error handling and to report the type of error. When Z30 parsed a file expression that specified an invalid directory, it simply substituted the current directory, but the program had no way to tell that this had been done. Z33 sets a flag to indicate the error. With Z33, a program can tell where on the search path it was actually found. This can be valuable for shells and

error handlers that install themselves into the system. They can operate faster if they know where they are located.

Finally, the source code has been completely reorganized and very extensively commented. I did this not only to make it easier for me to maintain it but also so that others could read it to learn how the command processor works. One of the main reasons why many of us hobbyists remain involved in the 8-bit world is that a Z80 can be comprehended fairly easily and can thus be used to learn deeply about the operation of a computer. Z33 is designed to contribute to this.

Advanced Z-COM Customization

In our discussion last time we described what Z-COM is and how it works, and we presented a number of techniques for carrying out simple modifications that did not alter the basic structure of the Z-COM system. This time we will examine some much more far-reaching modifications, including those that involve changing the way Z-COM operates. The goal is to develop techniques that will permit us to use the basic principles behind Z-COM to build arbitrary systems of our choice. I do want to warn you: a good part of this discussion will be at an advanced technical level. Even I feel rather mentally exhausted after going through the process of developing it and writing it down. If any one section is getting too technical for you, please do not give up completely. Skip ahead to each new section and read the introductory philosophical comments. There is material there that I would really like everyone to see.

When Z-COM Will Not Work

Before launching into the heavy code patching, I would like to cover a topic that probably should have been included last time: why Z-COM will not work in all systems or will not work fully.

There are two circumstances that I have experienced in which Z-COM interferes with the proper operation of a system. One class of difficulties arises when the system uses utilities that make modifications to the operating system image in memory. Such utilities always invite disaster, but they are nevertheless quite common (partly because they are so useful). If the utilities calculate the addresses to change from the BIOS warmboot address at location 0001, then there will be trouble, because that address points to the virtual BIOS set up by Z-COM and not to the real BIOS.

The Ampro BIOS, for example, has a number of special structures in defined locations with respect to the beginning of the BIOS. Some of these, for example, support the various configuration options. Since these options are rarely changed except when the system is first assembled or when new hardware is added, one can overcome any problem by running the utilities when Z-COM is not in operation. Other BIOS structures are used to define the alternative disk formats. These one might want to change during a session at the computer. Although it is inconvenient, one could again exit from Z-COM using ZCX, change the disk format, and then reenter Z-COM. Of course, a conventionally installed ZCPR system is available for the Ampro so that Z-COM is not necessary. However, a number of other computers running standard CP/M 2.2 use similar techniques and have similar utilities.

The second class of difficulties arises when the BIOS warmboot code performs some indispensable function. Remember that when Z-COM is running, the warm boot is intercepted, and the warmboot code in the original BIOS does not run. My BigBoard I with the double-density upgrade automatically selects from a large number of disk formats. One simply puts the diskette with the new format into the drive and presses control-c. The warmboot code in the BIOS includes code for determining the format of the diskette. When Z-COM is running, I often experience problems when I try to log in a new drive for the first time or when I try changing disk formats. The trouble seems to have to do with the way the BIOS keeps track of what drives are logged in, and by using disk resets or control-c's from Z-COM, I can often get the system to work. But clearly there can be problems when the BIOS warmboot code is completely by-passed.

• Z Best Sellers •

Z80 Turbo Modula-2 (1 disk) **$89.95**
The best high-level language development system for your Z80-compatible computer. Created by a famous language developer. High performance, with many advanced features; includes editor, compiler, linker, 552 page manual, and more.

Z-COM (5 disks) **$119.00**
Easy auto-installation complete Z-System for virtually any Z80 computer presently running CP/M 2.2. In minutes you can be running ZCPR3 and ZRDOS on your machine, enjoying the vast benefits. Includes 80+ utility programs and **ZCPR3: The Manual**.

Z-Tools (4 disks) **$169.00**
A bundle of software tools individually priced at $260 total. Includes the ZAS Macro Assembler, ZDM debuggers, REVAS4 disassembler, and ITOZ/ZTOI source code converters. HD64180 support.

PUBLIC ZRDOS (1 disk) **$59.50**
If you have acquired ZCPR3 for your Z80-compatible system and want to upgrade to full Z-System, all you need is ZRDOS. ZRDOS features elimination of control-C after disk change, public directories, faster execution than CP/M, archive status for easy backup, and more!

DSD (1 disk) **$129.95**
The premier debugger for your 8080, Z80, or HD64180 systems. Full screen, with windows for RAM, code listing, registers, and stack. We feature ZCPR3 versions of this professional debugger.

Quick Task (3 disks) **$249.00**
Z80/HD64180 multitasking realtime executive for embedded computer applications. Full source code, no run time fees, site license for development. Comparable to systems from $2000 to $40,000! Request our free Q-T Demonstration Program.

Z-System OEM inquiries invited.
Visa/Mastercard accepted. Add $4.00 shipping/handling in North America, actual cost elsewhere. Specify disk format.

Echelon, Inc.
885 N. San Antonio Road • Los Altos, CA 94022
415/948-3820 (Order line and tech support) Telex 4931646

More Named Directories

To get our feet wet again with Z-COM patching, let's start with a relatively simple but very practical example. The most frequent request I get from users of Z-COM, especially those using it to make a remote access system, is for a way to increase the number of named directories.

To refresh our memories, I have reproduced in Figure 1 the memory map of our unmodified Z-COM system. Those of you who are really sharp (or have photographic memories or are cheating by actually looking at the last issue) will notice that this is not exactly the same as the map presented last time as Figure 2. The reason for this is that I recently was offered a deal that I simply could not refuse on a hard-disk Televideo 803H system (every household needs four complete computer systems, no?). Since I cannot bear to operate a computer without Z-System, I immediately implemented Z-COM on it and have been using it as the testbed for the techniques described here. It's BIOS is obviously even less compact than the one on my BigBoard and starts 200H lower in memory. I hope this address switch does not confuse you

too much, but since your system probably does not match mine anyway, you have to get used to translating addresses. The addresses in the ZC.COM image, of course, do not change.

System Component	ZC.COM Address	System Address
CPR	0200 - 09FF	BA00 - C1FF
ZRDOS	0A00 - 17FF	C200 - EFFF
Virtual BIOS	1800 - 19FF	D000 - D1FF
Named Directory Register	1A00 - 1AFF	D200 - D2FF
Shell Stack	1B00 - 1B7F	D300 - D47F
Z3 Message Buffer	1B80 - 1BCF	D380 - D4CF
External FCB	1BD0 - 1BF3	D3D0 - D4F3
PATH	1BF4 - 1BFE	D3F4 - D4FE
Wheel Byte	1BFF - 1BFF	D3FF - D4FF
Environment Descriptor	1C00 - 1C7F	D400 - D47F
TCAP	1C80 - 1CFF	D480 - D4FF
Multiple Command Line	1D00 - 1DCF	D500 - D5CF
External Stack	1DD0 - 1DFF	D5D0 - D5FF
Resident Command Package	1E00 - 25FF	D600 - DDFF
Flow Control Package	2600 - 27FF	DE00 - DFFF
I/O Package	2800 - 2DFF	E000 - E5FF

Figure 1. Addresses of system components in the ZC.COM file and in the example system for which it was generated (Televideo 803H).

Now, if we want to have room for more directory names, we have to find a way to allocate more memory to the NDR module. Where can we steal some memory? Unfortunately, the memory cannot be taken from either of the neighbors of the NDR. The virtual BIOS and shell stack are indispensable. That means that we will have to move the NDR or something else from its present position. The best target for our memory raid is that hulking 6-page, 1.5K IOP that often goes unused, especially on remote access systems. We will cut it down to 3 pages and use the top 3 pages for our new NDR, which will have a capacity for 42 names [(3 * 256 - 1) div 18 = 42]. The resulting memory map is shown in Figure 2.

System Component	ZC.COM Address	System Address
CPR	0200 - 09FF	BA00 - C1FF
ZRDOS	0A00 - 17FF	C200 - EFFF
Virtual BIOS	1800 - 19FF	D000 - D1FF
(unused space)	1A00 - 1AFF	D200 - D2FF
Shell Stack	1B00 - 1B7F	D300 - D47F
Z3 Message Buffer	1B80 - 1BCF	D380 - D4CF
External FCB	1BD0 - 1BF3	D300 - D4F3
PATH	1BF4 - 1BFE	D3F4 - D4FE
Wheel Byte	1BFF - 1BFF	D3FF - D4FF
Environment Descriptor	1C00 - 1C7F	D400 - D47F
TCAP	1C80 - 1CFF	D480 - D4FF
Multiple Command Line	1D00 - 1DCF	D500 - D5CF
External Stack	1DD0 - 1DFF	D5D0 - D5FF
Resident Command Package	1E00 - 25FF	D600 - DDFF
Flow Control Package	2600 - 27FF	DE00 - DFFF
I/O Package	2800 - 2AFF	E000 - E2FF
Named Directory Register	2B00 - 2DFF	E300 - E5FF

Figure 2. Addresses of system components in the ZC.COM file and in the target system as modified to support 42 named directories.

To implement this change in our Z-COM file we have to change only the ENV and CPR modules. The ENV has to know about the new memory map, and the CPR code has to know where the NDR module is located. Although the NDR module will be placed in a new location in the ZC.COM file, the NDR data are position-independent, so the module itself need not be changed (though we will presumably be adding many new names). The FCP and RCP are still in the same place doing the same thing.

One of the new features of Z33, by the way, is the ability to determine the locations of the NDR, FCP, and RCP from the environment descriptor. Thus if we are using ZCPR33 with this feature enabled, no change in the CPR code is required.

Only three changes in the Z3BASE.LIB file are required to reflect the new memory map. These are the definitions for the symbols IOPS (the number of 128-byte records allocated to the IOP), Z3NDIR (the address of the NDR), and Z3NDIRS (the number of names in the NDR). These changes are as follows:

symbol	old expression	new expression
IOPS	12 (0CH)	6 (06H)
Z3NDIR	z3env - 200H	z3env + 0E00H or iop + 300H
Z3NDIRS	18 (12H)	42 (2AH)

The new SYS.ENV file can be made either by assembling SYSENV.ASM with the modified Z3BASE.LIB, or it can be done by patching (either to the image imbedded in ZC.COM or to the standalone ENV file). I didn't have a copy of SYSENV handy, so I have been using ZPATCH (which is much more fun anyway). I find that I am constantly in need of the addresses of various items in the environment descriptor, so to make them easier to find, I took my copies of Richard Conn's "ZCPR3, The Manual" and put a 3M Post-It® (one of those wonderful little yellow semi-stick note sheets) on page 300 where the SYSENV module is described. Then I wrote the offsets shown in Figure 3 into the margin next to the symbols. It was thus very easy to determine that the addresses to patch in the ZC.COM image are 1C11H (IOPS), 1C15H (Z3NDIR), and 1C17H (Z3NDIRS).

offset	SYSENV code line	
09	dw	expath
0B	db	expaths
0C	dw	rcp
0E	db	rcps
0F	dw	iop
11	db	iops
12	dw	fcp
14	db	fcps
15	dw	z3ndir
17	db	z3ndirs
18	dw	z3cl
1A	db	z3cls
1B	dw	z3env
1D	db	z3envs
1E	dw	shstk
20	db	shstks
21	db	shsize
22	dw	z3msg
24	dw	extfcb
26	dw	extstk
28	db	[quiet flag -- no symbol]
29	dw	z3whl
2B	db	[cpu speed -- no symbol]
2C	db	[max drive (A=1) -- no symbol]
2D	db	[max user]

Figure 3. Offsets to various symbols and information in SYSENV.ASM, the environment descriptor module (see "ZCPR3, The Manual" p. 300ff).

The next steps were to assemble up a new version of the command processor, create the desired NDR file, and then put all the pieces into the ZC.COM file as described last time. While I was at it, I decided that it would be nice if this new version could co-exist on the system with the previous version, in case I ever wanted the full IOP space back temporarily. To achieve this, I made two additional changes in the image and saved it to a file called ZC1.COM instead of ZC.COM. These two changes were to the name of the startup alias and the name of the CPR file to be loaded from A15 by the warmboot code.

The startup command is stored in the multiple command line buffer, whose image begins at 1D00H. The standard ZC.COM has the following data there for my Televideo 803H with its real MCL at D500H:

<04> <D5> <CC> <04> S T R T <00>

The first two bytes are a pointer to the address D504, where the next command (in this case the only command) to be executed is stored. They do not have to be changed. The third byte, CCH, is the maximum number of characters that the command line can contain. It should not have to be changed, but in fact the value is

wrong. Fortunately, this mistake can only cause trouble in highly exceptional circumstances, but while we are at it we can put in the correct value of CBH = 203. The value of the symbol Z3CLS in Z3BASE.LIB should also be changed. The correct value is the maximum number of actual characters in the command line. As can be seen above, there are four bytes before the command string and one byte (the terminating null) after it. Hence the proper value for Z3CLS is five less than the total amount of memory allocated to the multiple command line buffer module.

The fourth byte is the number of text characters in the command line. This value is never used by the operating system, but the DOS line input function writes the count to that position, so we have to provide space for it. If you put a wrong value there, it will not make any difference, at least not for the operations performed here. I have heard that there was at least one utility program that used this value for some purpose. I do not recommend this practice, since some command-line-generator programs, I believe, do not update the value after they produce their command lines. I am also not sure whether or not the Z3LIB routines APPCL and PUTCL update the character count.

To make ZC1.COM use a startup alias called START1 (6 characters), we would change the MCL buffer to

<04> <05> <CB> <06> S T A R T 1 <00>

This can be done either with ZPATCH or a debugger. If there is a lot of garbage in the rest of the command line buffer, you can fill it with zeros out through address 1DCFh to make things look neater.

The file control block for the ZC.CP command processor image that is loaded from directory A15 starts at address 1944H. By changing the space character at address 1947H from 20H to 31H ('1' ASCII), the CPR image ZC1.CP will be loaded instead. You can also change the message at address 1901H to reflect the name of the CPR image file. There is room to squeeze two extra characters into that message, one by eliminating the leading space and one by omitting the ending period. After you're done with these changes, don't forget to put the CPR image file ZC1.CP in A15.

It seems wasteful with this configuration to leave unused the block of memory where the NDR used to be. When I implemented this version, I moved the multiple command line buffer there so that I could increase its size from 208 to a full 256. One might not enter such long commands by hand, but aliases and other command line generators occasionally overflow the 203 character limit in the usual configuration. For ZCPR34 I am considering some techniques for extending the length to a two-byte value so that the command line can be as large as one would like. I will not describe the extra changes required to move the command line buffer, since the next example will cover that and more.

Completely Revamping the Memory Model

We will now consider how we go about completely revamping the memory model, including moving the IOP. Moving any module except for the IOP can be accomplished using a straightforward extension of what we have already described. The IOP poses some special problems that we will now deal with.

Before turning to that subject, I would like to make some general comments about the memory allocation in a ZCPR3 system. With a fixed system — that is, any particular system that one will use at all times — it does not really matter how the modules are distributed in memory, just so long as they all fit somewhere.

As soon as one wants to be able to change from one system configuration to another, not all memory models are as good as others. I first noticed this when I wanted to run both Z3-DOT-COM and Z-COM on my system. I usually used Z3-DOT-COM because it left a larger TPA, but occasionally I wanted to make use of the IOP for redirecting console output to a disk file. At that point I discovered that Joe Wright's choice of memory models was a poor one. By adding the IOP at the top in Z-COM rather than at the bottom, the environment descriptor moved down to a lower address, and that meant that I either had to have two sets of utility programs or had to reinstall all the utilities every time I changed from one system to the other. This provided a powerful incentive to discover methods for modifying the memory maps. Of course, with ZCPR33 and its automatic installation of utilities, this is no longer a concern.

Even with ZCPR33 there are compelling reasons to choose some memory configurations over others. The addresses of most system components are hard-coded into even the ZCPR33 command processor. However, as we mentioned earlier, the addresses of the largest memory buffers — the NDR, FCP, and RCP — can determined dynamically from the environment descriptor in memory. As a result, if these buffers are placed adjacent to one another, the single block of memory allocated to the set of them can be reconfigured simply by loading a new environment descriptor. Since the command processor does not directly refer to the IOP, the IOP can also be included at the bottom of this single buffer space. The tradeoffs that become possible are illustrated in Figure 4, where a single memory block of 4.25K (the standard amount in Z-COM for these modules) is allocated in two different ways.

SAGE MICROSYSTEMS EAST

Selling & Supporting The Best In 8-Bit Software

- **Plu*Perfect Systems**
  - Backgrounder II: switch between two or three running tasks under CP/M ($75)
  - DateStamper: stamp your CP/M files with creation, modification, and access times ($49)
- **Echelon (Z-System Software)**
  - ZCPR33: full system $49, user guide $15
  - ZCOM: automatically installing full Z-System ($70 basic package, or $119 with all utilities on disk)
  - ZRDOS: enhanced disk operating system, automatic disk logging and backup ($59.50)
  - DSD: the incredible Dynamic Screen Debugger lets you really see programs run ($130)
- **SLR Systems (The Ultimate Assembly Language Tools)**
  - Assemblers: Z80ASM (Z80), SLR180 (HD64180), SLRMAC (8080), and SLR085 (8085)
  - Linker: SLRNK
  - Memory-based versions ($50)
  - Virtual memory versions ($195)
- **NightOwl (Advanced Telecommunications)**
  - MEX-Plus: automated modem operation ($60)
  - Terminal Emulators: VT100, TVI925, DG100 ($30)

Same-day shipping of most products with modem download and support available. Shipping and handling $4 per order. Specify format. Check, VISA, or MasterCard.

Sage Microsystems East
1435 Centre St., Newton, MA 02159
Voice: 617-965-3552 (9:00 a.m. - 11:15 p.m.)
Modem: 617-965-7259 (24 hr., 300/1200/2400 bps, password = DDT, on PC-Pursuit)

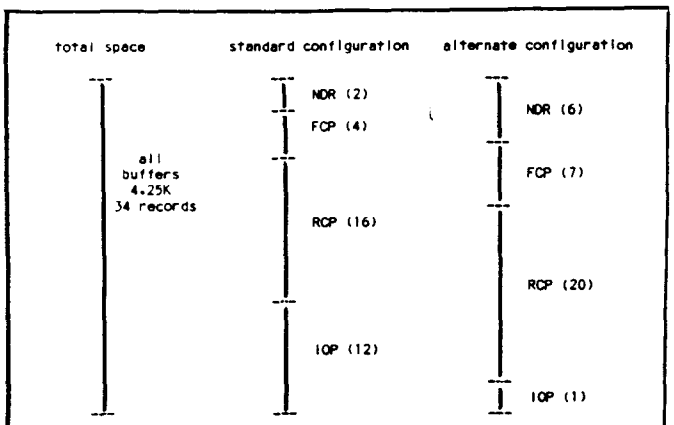


Figure 4. One illustration of how a single block of memory can be dynamically allocated among the IOP, RCP, FCP, and NDR buffers in a ZCPR33 system.

In the alternative memory map in Figure 4 the IOP buffer has been shrunk to a single record, a space just large enough to hold the dummy IOP that Z-COM comes with (we can't shrink it to zero without changing the VBIOS). The extra 11 records of memory are then distributed to the other modules. The NDR increases to 6 records, enough for 42 named directories. The FCP picks up 3 more records. At 7 records, it has enough space to implement a very large number of resident test options. The remaining 4 records go to the RCP, which can probably now include all options in Z33RCP.

Suppose the standard configuration is described by SYS.ENV and uses the modules SYS.NDR, SYS.FCP, and SYS.RCP, and that the alternative configuration is described by ALT.ENV with modules ALT.NDR, ALT.FCP, and ALT.RCP. Then we would change from the standard to the alternate configuration by entering the command

```
LDR ALT.ENV,ALT.NDR,ALT.FCP,ALT.RCP
```

Note that it is essential that the ENV module be listed, and therefore loaded, first. If it is not, LDR will not know the correct addresses for the other modules. Similarly, to change back to the standard configuration, one would use the command

```
LDR SYS.ENV,SYS.NDR,SYS.FCP,SYS.RCP
```

If one were switching back, for example, to use the NuKey IOP to provide keyboard macro capability, the line could read

```
LDR SYS.ENV,SYS.NDR,SYS.FCP,SYS.RCP,NUKEY.IOP
```

In this way one can make much more flexible use of system resources. One might choose to reduce the size of the overall buffer space to only 35 records, keeping only the IOP stub in the standard configuration. Then when an IOP like IOR (I/O recorder) or BPRINT (print spooler) is needed, the RCP and/or FCP would be contracted temporarily. Either one or both could even be eliminated (reduced to zero size). Aliases can be used to automate the process of switching configurations.

Moving the IOP

We will now build a Z-COM system of the form described above. The memory map for the new configuration is shown in Figure 5. Since the ENV (including TCAP) is a fixture in any system, I would have preferred to put it in the invariant position at the very top of memory. However, ZRDOS has to know where the ENV is in order to support wheel-locking of files. If you have purchased ZRDOS separately, you can generate a version to run at any address and to reference an ENV at any address. If you only have Z-COM, however, you do not have this freedom. Therefore, I have left the environment where it was.

System Component	ZC.COM Address	System Address
operating system modules		
CPR	0200 - 09FF	BA00 - C1FF
ZRDOS	0A00 - 17FF	C200 - EFFF
Virtual BIOS	1800 - 19FF	D000 - D1FF
large, variable buffers		
I/O Package	1A00 - 1FFF	D200 - D7FF
Resident Command Package	2000 - 25FF	D800 - DFFF
Flow Control Package	2800 - 27FF	E000 - E1FF
Named Directory Register	2A00 - 1AFF	E200 - E2FF
third page of buffers		
Shell Stack	2B00 - 1B7F	E300 - E47F
Z3 Message Buffer	2B80 - 1BCF	E380 - E4CF
External FCB	2BD0 - 1BF3	E3D0 - E4F3
PATH	2BF4 - 1BFE	E3F4 - E4FE
Wheel Byte	2BFF - 1BFF	E3FF - E3FF
second page of buffers		
Environment Descriptor	2C00 - 1C7F	E400 - E47F
TCAP	2C80 - 2DFF	E480 - E4FF
top page of buffers		
Multiple Command Line	2D00 - 1DCF	E500 - E5CF
External Stack	2D00 - 1DFF	E500 - E5FF

Figure 5. Memory map for a system designed for dynamic buffer reallocation under ZCPR33.

The first step, as usual, in making a new configuration is to prepare a new Z3BASE.LIB file. The important addresses in that file are shown in Figure 6. I have considered each page of memory to be a unit. The bottom of the page is referenced to the real BIOS address, and the other modules in the page are referenced to the base module for that page. Of course, you can express these addresses in many other equivalent ways, and you may well prefer to do it differently. In any case, with Z3BASE.LIB in hand, you can assemble up the CPR, RCP, FCP, and ENV modules (or you can make the latter by patching).

```
; Z3BASE.LIB
        rbios   equ     0e600h
; First page under BIOS
        z3cl    equ     rbios - 100h
        z3cls   equ     203
        extstk  equ     z3cl + 0d0h
; Second page under BIOS
        z3env   equ     rbios - 200h
        z3envs  equ     2
; Third page under BIOS
        shstk   equ     rbios - 300h
        shstks  equ     4
        shsize  equ     32
        z3msg   equ     shstk + 080h
        extfcb  equ     shstk + 0d0h
        expath  equ     shstk + 0f4h
        expaths equ     5
        z3whl   equ     shstk + 0ffh
; Variable modules
        z3ndirs equ     14
        z3ndir  equ     rbios - 0400h
        fcps    equ     4
        fcp     equ     z3ndir - fcps*80h
        rcps    equ     16
        rcp     equ     fcp - rcps*80h
        iops    equ     12
        iop     equ     rcp - iops*80h
; Operating system components
        vbios   equ     iop - 0200h
        dos     equ     vbios - 0e00h
        ccp     equ     dos - 0800h
```

Figure 6. Address equates for the Z3BASE.LIB file corresponding to the memory configuration shown in Figure 5.

We would be able to implement this configuration without any problem using the techniques we have already seen were it not for the IOP module. The IOP is really an extension of the BIOS, and the problem is that the virtual BIOS has vectors (jump instructions) going to the IOP. When the Z-COM system is built by the loader program ZCLD.COM, the addresses are calculated based on the assumed relative positions of the VBIOS and IOP. Since we are now going to change the spacing between them, we will have to perform some patching on the VBIOS module.

The table of jump vectors in the virtual BIOS is shown in Figure 7. There are two jumps (warm and cold boot) that are internal to the VBIOS, 7 jumps to addresses in the IOP (6 in one group and one extra), and 10 jumps to the real BIOS. Only the jumps to the IOP have to be changed.

```
(00)    VBIOS:  JP      VBIOS + 67H     ; coldboot
(03)            JP      VBIOS + 67H     ; warmboot
(06)            JP      IOP + 0CH       ; console status
(09)            JP      IOP + 0FH       ; console input
(0C)            JP      IOP + 12H       ; console out
(0F)            JP      IOP + 15H       ; list out
(12)            JP      IOP + 18H       ; punch out
(15)            JP      IOP + 1BH       ; reader in
(18)            JP      BIOS + 18H      ; home disk
(1B)            JP      BIOS + 1BH      ; select disk
(1E)            JP      BIOS + 1EH      ; set track
(21)            JP      BIOS + 21H      ; set sector
(24)            JP      BIOS + 24H      ; set DMA
(27)            JP      BIOS + 27H      ; read sector
(2A)            JP      BIOS + 2AH      ; write sector
(2D)            JP      IOP + 1EH       ; list status
(30)            JP      BIOS + 30H      ; sector translation
```

Figure 7. Structure of the jump table in the virtual BIOS.

You should begin by making a copy of ZC1.COM, which we generated earlier, giving it the name ZC2.COM. Then determine the absolute address of the IOP in your system. Next, go into ZC2.COM with ZPATCH or with a debugger and change the addresses. For my Televideo 803H system the IOP had been at E000H and is now at D200H. I would look for the seven jumps of the form "C3 ?? E0" and change them to "C3 ?? D2".

We also have to make some changes to the code in the dummy IOP. It has a total of 16 jump instructions, 7 of which refer to the real BIOS and 9 of which refer to internal addresses. The former need not be changed, but, since we are changing the address at which the IOP will execute, we have to change the latter addresses. The source code for the dummy IOP is shown in Figure 8.

```
(00)    IOP:    JP      IOP + 30H       ; status
(03)            JP      IOP + 30H       ; device select
(06)            JP      IOP + 30H       ; device name
(09)            JP      IOP + 30H       ; initialization
(0C)            JP      BIOS + 06H      ; console status
(0F)            JP      BIOS + 09H      ; console input
(12)            JP      BIOS + 0CH      ; console output
(15)            JP      BIOS + 0FH      ; list output
(18)            JP      BIOS + 12H      ; punch output
(15)            JP      BIOS + 15H      ; reader input
(18)            JP      BIOS + 2DH      ; list status
(1B)            JP      IOP + 30G       ; new I/O routine
(1E)            JP      IOP + 30H
(21)            JP      IOP + 30H
(24)            JP      IOP + 30H
(27)            JP      IOP + 30H
(30)            DB      'Z30IOPDUMMY    ' ; ID string
(30)            XOR     A               ; set zero value and flag
(3E)            RET                     ; return
```

Figure 8. Source code for the dummy IOP in Z-COM. All the internal routines simply return with the zero flag set, while the substantive functions are vectored off to the real BIOS.

The garbage that appears from address IOP + 3FH to the end of the IOP at IOP + 5FFH can be filled with zeros to make things look neater. Then the 9 internal vectors have to be modified in the same way those in the BIOS were to point to the new address of the IOP. Finally, the reconfigured IOP has to be moved from its present position at 2800H to its new place in ZC2.COM at 1A00H.

While we are in the debugger doing that, we can also change (1) the file control block to load the command processor image ZC2.CP (and also the 'not found' message) and (2) the multiple command line buffer to run START2. This initial multiple command line must also be moved from its old address at 1D00H to its new position at 2D00H. Make sure that the second byte in the command line buffer points to the page where the real multiple command line buffer will be.

You should then fill all the other buffers with zeros. The CPR, RCP, FCP, and ENV modules can be loaded into the image. Then the initial path should be set up at address 2BF4H, and the wheel byte at address 2BFFH should be set to FFH if you want it on. Finally, if you want an error handler to be loaded when Z-COM is booted, then patch in the error command line (for example, "A0:Z33VERR<0>") at Z3MSG + 10H (2B90H in ZC2.COM).

If everything has gone right, and I have not left something out or written something wrong, you should be ready to test it out. Don't forget to put the command processor image files in A15. I know I keep harping on this, but while working on this article, I forgot to do that on one occasion, and there are now a few dents in the table from my fist!

A Minimum System

If you ever meet me and feel like having a little fun at my expense, just casually make some comment like, "Oh, yeah, I hear ZCPR3 has some nice features but that it uses up much too much memory." It is a subject that has become a sore point for me lately and one that is almost guaranteed to provoke me. To be honest, however, the persistance of this false impression is to a large degree the result of poor education on our part. No one ever talks about minimum ZCPR3 systems; we only describe full-blown ones.

It is true that a full-blown ZCPR3 system uses quite a bit of memory. Z-COM takes 1600H = 5.5K bytes of memory out of your TPA, a pretty hefty chunk. I like the features that this big Z-System gives me, and for the kind of work I do, the smaller TPA almost never causes any problem. For me, the benefits far outweigh the cost.

For some people, however, especially those working with voracious memory hogs like database managers and C compilers, this is not the case. (It seems to me that someone like Steve Russell of SLR should write a good, virtual-memory C compiler like his virtual-memory assemblers to prevent this problem.) But the real answer is that a ZCPR3 system does not have to have every feature implemented. As with most things in this world, the 20/80 rule applies: for less than 20% of the cost, you can buy more than 80% of the features. If we eliminate all the variable buffers in the last example, we end up with a system that uses only 5 pages (1.25K) of memory, a very modest amount. The same system installed in the conventional way, with no space required for the virtual BIOS of Z-COM, would take only 0.75K away from the TPA, yet all of the following features would still be there:

automatic command search path
flexible access to user areas
multiple commands on a line
aliases
shells (including history shell and filer shells)
extended command processing (including ARUNZ command generator)
powerful error handlers (with command editing)
terminal-independent operation (TCAP)
flexible program loading (type-3 environment)
interprogram communication

The only things missing are named directories and flow control. Some simple flow-control-like features could be implemented even without the FCP. With all the security features and named-directory support removed from the command processor, there is room for many more CPR-resident commands, and the ones that won't fit can be implemented as virtual residents using the new type-3 environment. In summary, a very powerful system can be

built with a very small cost in TPA.

To prove this point (and because it will be useful on those rare occasions when I run my database manager), I decided to develop a version of Z-COM that would have only the three pages of core modules. The memory map is shown in Figure 9. It differs in two fundamental ways from all the other maps we have seen: (1) it is shorter and (2) the real VBIOS and ZRDOS run at different addresses than usual. These differences will require some new techniques, and we will cover them shortly.

System Component	ZC.COM Address	System Address
CPR	0200 - 09FF	C800 - D1FF
ZRDOS	0A00 - 17FF	D300 - E0FF
Virtual BIOS	1800 - 19FF	E100 - E2FF
Shell Stack	1A00 - 1A7F	E300 - E47F
Z3 Message Buffer	1A80 - 1ACF	E380 - E4CF
External FCB	1AD0 - 1AF3	E3D0 - E4F3
PATH	1AF4 - 1AFE	E3F4 - E4FE
Wheel Byte	1AFF - 1AFF	E3FF - E4FF
Environment Descriptor	1B00 - 1B7F	E400 - E47F
TCAP	1B80 - 1BFF	E480 - E4FF
Multiple Command Line	1C00 - 1CCF	E500 - E5CF
External Stack	1CD0 - 1CFF	E5D0 - E5FF

Figure 9. Addresses of system components in the ZC3.COM file and in the target system for a minimum configuration with no IOP, RCP, FCP, or NDR. The file is 2E00H - 1D00H = 1100H = 4.25K shorter than the other versions.

First we have to make Z3BASE.LIB, the equates for which are shown in Figure 10. The only subtle change is in the way the VBIOS address is defined. Make sure you do not define it in terms of any of the modules whose addresses have been set to zero to disable them or you will get extremely strange results.

```
; Z3BASE.LIB
        rbios   equ     0e600h
; First page under BIOS
        z3cl    equ     rbios - 100h
        z3cls   equ     203
        extstk  equ     z3cl + 0d0h
; Second page under BIOS
        z3env   equ     rbios - 200h
        z3envs  equ     2
; Third page under BIOS
        shstk   equ     rbios - 300h
        shstks  equ     4
        shsize  equ     32
        z3msg   equ     shstk + 080h
        extfcb  equ     shstk + 0d0h
        expath  equ     shstk + 0f4h
        expaths equ     5
        z3whl   equ     shstk + 0ffh
; Variable modules — all disabled
        z3ndirs equ     0
        z3ndir  equ     0
        fcps    equ     0
        fcp     equ     0
        rcps    equ     0
        rcp     equ     0
        iops    equ     0
        iop     equ     0
; Operating system components
        vbios   equ     shstk - 0200h
        dos     equ     vbios - 0e00h
        ccp     equ     dos - 0800h
```

Figure 10. The Z3BASE.LIB equates for a minimum system that takes only 0.75K for the ZCPR3 buffers and 0.5K for the virtual BIOS required for automatic installation.

The basic procedure for creating this system is very much like what we have done before. We edit the Z3BASE.LIB file and assemble up the CPR and ENV modules (or make the latter by patching). There are no FCP, NDR, or IOP modules to worry about. We set up the path, the wheel, and the error handler; we change the VBIOS file control block to load ZC3.CP for the CPR image file and change the 'not found' message to match; and we put in the ENV and CPR modules.

Now we have to face the new complications that arise because of the change in size of the file. First we will take up the simpler problem — changing the loader code in the first page of the file. It normally copies 2C00H bytes (from 200H to 2E00H) up to the run-time location in memory. Now the image part of the file is only 1B00H bytes long. If you follow through the loader code in a debugger, you should not have too much difficultly figuring out what is going on. The hardest parts to trace through are the places where the code prints out in-line strings. You will typically see a call instruction followed by very strange code. That strange code is the text to be printed. That coding technique is very convenient for the programmer (and I use it all the time), but it makes disassembly and single-stepping in a debugger much more difficult. When you encounter code like this, you have to use the dump ('D') display to locate the null (binary 0) that marks the end of the string. Then you can continue running using the command "G,addr", where 'addr' is the address just after the null.

Anyway, at address 181H you will find the key instruction: LD B.C.,2C00H. This must simply be changed to LD B.C.,1B00H. That's all there is to it — as far as the loader code is concerned. You might wonder why we don't have to change the starting address for the load, since it is not the same as before. The answer is that Z-COM derives it from the initial jump instruction in the CPR image. My first versions of Z33 used a relative jump, and I had to put the absolute jump back after one of my beta-testers pointed out that it would not work with Z-COM.

Now we have to face two much more difficult problems: both the VBIOS and the ZRDOS modules have to be relocated to a new address. How can we possibly do this without source code? Well, if you purchased a separate copy of ZRDOS, you have a file with a name like ZRDINS.COM with which you can create a binary image of ZRDOS that will run at any address and with any ENV address. But what if you don't have it? And what about the VBIOS part? The latter could conceivably be disassembled, since the code is not very long or very complex, but there is a better way, one that makes use of the built-in capabilities of the auto-install package.

If you had two computers with different BIOS entry addresses, you could perform a standard installation of Z-COM on each system. By taking the two ZC.COM files and subtracting them in a debugger, you could derive the relocation map. Now the question is how we can make two ZC.COMs using a single computer.

If you examine the beginning of the code in ZCLD.COM, you will see that ZCLD keys its system generation to the address of the BIOS warmboot vector at address 0001, and that is what we will base our strategy on. We will fool ZCLD into making us two versions of ZC.COM one page apart that we can subtract.

With my standard Z-COM system running on the Televideo 803H, the CPR is at BA00H and the virtual BIOS at D000H. Thus the vector at address 0001 points to D003H. By entering the following commands, we can make the BIOS look as though it were at B900:

```
POKE B903 C3 03 D0      ; Set up JP D003H at address B903H
POKE 2 B9               ; Change bios vector from D003 to B903
```

Of course, you must use addresses appropriate to your system. Any address below the CPR but above the memory used by ZCLD should work.

With this patch in place, everything will be fine so long as we do not try to run a program that does direct BIOS calls based on the address at 0001. (If we do? — the system will simply go up in flames!) In case you're worried, the next warmboot will delete our patch and restore things to normal.

Fortunately, ZCLD does not mind being fooled this way. All in all, the following sequence of commands will result in two files, ZCB8.COM and ZCB9.COM.

```
POKE B903 CE 03 D0;POKE 2 B9;ZCLD;REN ZCB9.COM=ZC.COM
POKE B803 CE 03 D0;POKE 2 B8;ZCLD;REN ZCB8.COM=ZC.COM
```

We can then load both files into a debugger using the commands

```
IZCB8.COM              ; set file to ZCB8.COM
R                      ; read in at 100H (100H-2DFFH)
IZCB9.COM              ; set file to ZCB9.COM
R3000                  ; read in at 3100H (3100H-5DFFH)
```

Now from right in the debugger we can assemble a little program at 3000H to subtract the ZRDOS and VBIOS images in memory block 0A00H-19FFH from the images in memory block 3A00H-49ffH to give us the relocation map we need. Here is how the entry of the assembly code proceeds:

```
-A3000
3000  LD DE,3A00        ; set up pointers to two files
3003  LD HL,A00
3006  LD B.C.,1000      ; number of bytes to subtract
3009  LD A,(DE)         ; get byte from higher image
300A  SUB (HL)          ; subtract byte from lower image
300B  LD (DE),A         ; put result (0 or 1) back
300C  INC HL            ; increment the pointers
300D  INC DE
300E  DEC B.C.          ; check count
300F  LD A,B
3010  OR C
3011  JR NZ,3009        ; loop through 1000H bytes
3013
```

Enter the command "G3000,3013" to run this routine with a breakpoint at 3013. If you look at memory from 3A00H TO 49FFH you will see a pattern of 1s and 0s. This is the relocation map. It indicates which bytes must be changed to shift the execution address of the code.

Now we have to relocate the ZRDOS to run at D300H and the VBIOS to run at E100H instead of the values 9400H and A200H as they are in the image in ZCB8.COM (determined by inspection with a debugger). Thus we have to add an offset of 3FH (E1 – A2) to all bytes where the relocation map has a one in it. So we assemble up another little program at 3000H as follows:

```
-A3000
3000  LD HL,3A00        ; point to relocation byte
3003  LD DE,1A00        ; point to ZRDOS/VBIOS we are creating
3006  LD B.C.,1000      ; number of bytes to cover
3009  LD A,(DE)         ; get current value of byte
300A  BIT 0,(HL)        ; see if relocation map has a 1 in it
300C  JR Z,3011         ; if not, skip the offset addition
300E  ADD A,3F          ; add the offset
3010  LD (DE),A         ; put back the corrected byte
3011  INC HL            ; increment the pointers
3012  INC DE
3013  DEC B.C.          ; check the count
3014  LD A,B
3015  OR C
3016  JR NZ,3009        ; loop through 1000h bytes
3018
```

Run this program with "G3000,3018" and you will have a ZRDOS and VBIOS that will run at the required address. Put them in a safe place temporarily while you load in ZC3.COM and then move them into the proper place in the image. This completes the generation of the minimum system except for one step described a little later.

Although all that work with the debugger took quite a lot of space to describe, it is really not that difficult to do. I particularly wanted to show it to you because it is a technique that is useful in many other circumstances as well (like using MOVCPM to relocate your system in one-page rather than 1K increments). Now, however, I will show you a much easier way to accomplish the same thing in the case of Z-COM.

If you load ZCLD.COM into the debugger and trace through the code with the 'L' command, you will see how it works. Before one gets to the main code, there are two places where checks are made. One is to see if the command was invoked as "ZCLD //" to request the built-in help screen. The second test is to make sure that you are not trying to run ZCLD from inside a running Z-COM system. We could drop out of Z-COM after making the changes I am about to describe, but why put up with that trouble? The code is testing for the presence of the letter 'Z' in the copyright notice inside the VBIOS at offset 42H. At address 249H there is a "JP NZ,29D" instruction that takes one to the system-building code if no 'Z' is detected. If we change this to an unconditional jump, we will effectively disable the test.

At 29DH, the code loads the address of the BIOS with the instruction "LD HL,(1)". We know that we want the system to be generated at an address 1100H higher than it would be normally because of all the modules we removed. Since the normal BIOS address was E600H, we want ZCLD to see a value of F700H. To accomplish this, we assemble in the instruction "LD HL,F700", a direct load instead of an indirect load to HL. We can then save away this modified version as ZCLD1.COM. When we run it, it very nicely produces a system with a ZRDOS and VBIOS at just the addresses we wanted.

There is one other change I have forgotten to mention. The jumps in the VBIOS that go to the IOP have to be replaced by jumps to the real BIOS at the very same offset as in the VBIOS. Thus the jump at offset 06, which was "JP IOP+0CH", would become "JP BIOS+06HG" .

Switching from One Version to Another

What if we are running one of our versions of Z-COM and want to change to another one. We can always run the ZCX command to get back to CP/M and then load the new Z-COM system. This seems unnecessarily tedious. Why can't we just run ZC2.COM from the ZC1 system? The answer is that Joe Wright did not want us to do this. Of course, he was thinking in terms of only single configurations, and then there would be no reason to run ZC.COM when Z-COM was already running. So, he put in some code to protect us from this mistake.

Now we want to do that very thing! How can we disable the safety code? Very simply. It is based on the same check we described with the ZCLD program. It looks for a 'Z' at byte 42H of the BIOS or VBIOS. If it is a VBIOS, the 'Z' will be there; if it is a real BIOS, it would be very unlikely that a 'Z' would be there. We could go into the code itself as I described with ZCLD and disable the test. Alternatively, we could do something much simpler (and reversible): go into the VBIOS and remove the 'Z' by changing it to something else. A third possibility, one I implemented for the fun of it, is to write a little utility program that finds that byte of the BIOS. If it is not a 'Z', it simply returns; if it is a 'Z', it sets it to zero. Then the next ZCx can load over it. If the utility is not run, then the system cannot be overlaid.

I will suggest one last modification to Z-COM to enhance it even further. That is a change to allow alternative versions of Z-COM to be loaded without interrupting the flow of commands. Thus your memory-hungry C compiler would be invoked with an alias the reads something like:

```
RUNC zc3;c $*;zc2
```

This alias would load the minimum Z-COM system to give the C compiler the most room to work, and then once the compilation was finished it would reload the full Z-COM configuration. I did this with my Z3-DOT-COM/Z-COM combination. The I/O Recorder IOP was invoked using an alias. The alias would first check to see which Z-System was running (that is what I put the $M parameter into ARUNZ for). If it was Z3-DOT-COM, then alias would load Z-COM before proceeding to load the IOP. I will not go through the method for accomplishing this, but I will give you a hint. You have to change the loader code in the first page of ZCx.COM so that the multiple command line buffer and some other system information is not overwritten by the load.

Plans for Next Time

Whew! That was quite a session of heavy technical material. I don't know yet exactly what I will do next time, but I certainly I will find a less technical subject. You need a rest, and I need a rest. If you have any questions or suggestions, please write or call. See the ad for Sage Microsystems East for the address and phone numbers. ■

Z sets you free!

Who we are

Echelon is a unique company, oriented exclusively toward your CP/M-compatible computer. Echelon offers top quality software at extremely low prices; customers are overwhelmed at the amount of software they recieve when buying our products. For example, the Z-Com product comes with approximately 92 utility programs; and our TERM III communications package runs to a full megabyte of files. This is real value for your software dollar.

ZCPR 3.3

Echelon is famous for our operating systems products. ZCPR3, our CP/M enhancement, was written by a software professional who wanted to add features normally found in minicomputer and mainframe operating systems to his home computer. He succeeded wonderfully, and ZCPR3 has become the environment of choice for "power" CP/M-compatible users. Add the fine-tuning and enhancements of the now-available ZCPR 3.3 to the original ZCPR 3.0, and the result is truly flexible modern software technology, surpassing any disk operating system on the market today. Get our catalog for more information - there's four pages of discussion regarding ZCPR3, explaining the benefits available to you by using it.

Z-System

Z-System is Echelon's complete disk operating system, which includes ZCPR3 and ZRDOS. It is a complete 100% compatible replacement for CP/M 2.2. ZRDOS adds even more utility programs, and has the nice feature of no need to warm boot (^C) after changing a disk. Hard disk users can take advantage of ZRDOS "archive" status file handling to make incremental backup fast and easy. Because ZRDOS is written to take full advantage of the Z80, it executes faster than ordinary CP/M and can improve your system's performance by up to 10%.

Installing ZCPR3/Z-System

Echelon offers ZCPR3/Z-System in many different forms. For $49 you get the complete source code to ZCPR3 and the installation files. However, this takes some experience with assembly language programming to get running, as you must perform the installation yourself.

For users who are not qualified in assembly language programming, Echelon offers our "auto-install" products. Z-Com is our 100% complete Z-System which even a monkey can install, because it installs itself. We offer a money-back guarantee if it doesn't install properly on your system. Z-Com includes many interesting utility programs, like UNERASE, MENU, VFILER, and much more.

Echelon also offers "bootable" disks for some CP/M computers, which require absolutely no installation, and are capable of reconfiguration to change ZCPR3's memory requirements. Bootable disks are available for Kaypro Z80 and Morrow MD3 computers.

Z80 Turbo Modula-2

We are proud to offer the finest high-level language programming environment available for CP/M-compatible machines. Our Turbo Modula-2 package was created by a famous language developer, and allows you to create your own programs using the latest technology in computer languages - Modula-2. This package includes full-screen editor, compiler, linker, menu shell, library manager, installation program, module library, the 552 page user's guide, and more. Everything needed to produce useful programs is included.

"Turbo Modula-2 is fast...[Sieve benchmark] runs almost three times as fast as the same program compiled by Turbo Pascal...Turbo Modula-2 is well documented...Turbo's librarian is excellent". - Micro Cornucopia #35

BGii (Backgrounder 2)

BGii adds a new dimension to your Z-System or CP/M 2.2 computer system by creating a "non-concurrent multitasking extension" to your operating system. This means that you can actually have two programs active in your machine, one or both "suspended", and one currently executing. You may then swap back and forth between tasks as you see fit. For example, you can suspend your telecommunications session with a remote computer to compose a message with your full-screen editor. Or suspend your spreadsheet to look up information in your database. This is very handy in an office environment, where constant interruption of your work is to be expected. It's a significant enhancement to Z-System and an enormous enhancement to CP/M.

BGii adds much more than this swap capability. There's a background print spooler, keyboard "macro key" generator, built-in calculator, screen dump, the capability of cutting and pasting text between programs, and a host of other features.

For best results, we recommend BGii be used only on systems with hard disk or RAMdisk.

JetFind

A string search utility is indispensible for people who have built up a large collection of documents. Think of how difficult it could be to find the document to "Mr. Smith" in your collection of 500 files. Unless you have a string search utility, the only option is to examine them manually, one by one.

JetFind is a powerful string search utility which works under any CP/M-compatible operating system. It can search for strings in text files of all sorts - straight ASCII, WordStar, library (.LBR) file members, "squeezed" files, and "crunched" files. JetFind is very smart and very fast, faster than any other string searcher on the market or in the public domain (we know, we tested them).

Software Update Service

We were suprised when sales of our Software Update Service (SUS) subscriptions far exceeded expectations. SUS is intended for our customers who don't have easy access to our Z-Node network of remote access systems. At least nine times per year, we mail a disk of software collected from Z-Node Central to you. This covers non-proprietary programs and files discussed in our Z-NEWS newsletter. You can subscribe for one year, six months, or purchase individual SUS disks.

There's More

We couldn't fit all Echelon has to offer on a single page (you can see how small this typeface is already!). We haven't begun to talk about the many additional software packages and publications we offer. Send in the coupon below and just check the "Requesting Catalog" box for more information.

Item	Name	Price	
1	ZCPR3 Core Installation Package	$49.00	(3 disks)
2	ZCPR3 Utilities Package	$89.00	(10 disks)
5	Z-Com (Auto-Install Complete Z-System)	$119.00	(5 disks) *
6	Z-Com "Bare Minimum"	$69.95	(1 disks)
10	BGii Backgrounder 2	$75.00	(2 disks)
12	PUBLIC ZRDOS Plus (by itself)	$59.50	(1 disk)
13	Kaypro Z-System Bootable Disk	$69.95	(3 disks)
14	Morrow MD3 Z-System Bootable Disk	$69.95	(2 disks)
16	QUICK-TASK Realtime Executive	$249.00	(3 disks)
17	DateStamper file time/date stamping	$49.95	(1 disk)
18	Software Update Service	$85.00	(1 yr sub)
20	ZAS/ZLINK Macro Assembler and Linker	$69.00	(1 disk)
21	ZDM Debugger for 8080/Z80/ HD64180 CPU's	$50.00	(1 disk)
22	Translators for Assembler Source code	$51.00	(1 disk)
23	REVAS3/4 Disassembler	$90.00	(1 disk)
24	Special Items 20 through 23	$169.00	(4 disks)
25	DSD-80 Full Screen Debugger	$129.95	(1 disk)
27	The Libraries.SYSLIB, Z3LIB, and VLIB	$99.00	(8 disks)
28	Graphics and Windows Libraries	$49.00	(1 disk)
29	Special Items 27, 28, and 82	$149.00	(9 disks)
30	Z80 Turbo Modula-2 Language System	$89.95	(1 disk)
40	Input/Output Recorder IOP (IOR)	$39.95	(1 disk)
41	Background Printer IOP (BPrinter)	$39.95	(1 disk)
44	NuKey Key Redefiner IOP	$39.95	(1 disk)
45	Special Items 40 through 44	$89.95	(3 disks)
60	DISCAT Disk cataloging system	$39.99	(1 disk)
61	TERM3 Communications System	$99.00	(6 disks)
64	Z-Msg Message Handling System	$99.00	(1 disk)
66	JetFind String Search Utility	$49.95	(1 disk)
81	ZCPR3: The Manual bound, 350 pages	$19.95	
82	ZCPR3: The Libraries 310 pages	$29.95	
83	Z-NEWS Newsletter, 1 yr subscription	$24.00	
84	ZCPR3 and IOPs 50 pages	$9.95	
85	ZRDOS Programmer's Manual 35 pages	$8.95	
88	Z-System User's Guide 80 page tutorial	$14.95	

* Includes ZCPR3: The Manual

Echelon, Inc.

885 N. San Antonio Road, Los Altos, CA 94022 USA

415/948-3820 (order line and tech support)

Telex 4931646

NAME _____

ADDRESS _____

_____

TELEPHONE _____ DISK FORMAT _____

☐ REQUESTING CATALOG

ORDER FORM

Payment to be made by:

☐ Cash
☐ Check
☐ Money Order
☐ UPS COD
☐ Mastercard/Visa:

_____

Exp. Date _____

California residents add 7% sales tax.
Add $4.00 shipping/handling in North America, actual cost elsewhere.

ITEM	PRICE
_____	_____
_____	_____
_____	_____
Subtotal	_____
Sales Tax	_____
Shipping/Handling	_____
Total	_____

68000

Why use a New OS & the 68000?

by Joe Bartel, Hawthorne Technology

Why Work With a New Operating System?

The small computer market is caught between two ruts today. On the small side is the PC and on the large side is Unix. The other players missed the boat by having a great (or so they thought) interface with nothing behind it to do any useful work. To be PC compatible is a dead end. The system is a kludge.

As developers try to squeeze the last bit of performance from the PC there will be problems. It is true that there are several million PCs in the world today. This doesn't mean there is a good market. Because the market is so large, it is hard (and expensive), for a small company to make themselves heard. There are public domain or low priced programs for every common application that anyone wants. These are hard to compete with. The pressure is to continue lowering prices while cutting profits. A business person needs to look at what point he can no longer afford to remain in this kind of market.

To break out of this rut a new system architecture is needed. Use the PC and clones where they fit but start to forge ahead in new directions. This doesn't mean trying to run a PC program on another machine. It is possible to emulate an 8086 on a 68000 but a full PC emulation is not worth while. In every case so far the emulation costs more than a PC clone. The interchange of disks on the other hand is very economical and easy to do. This protects the investment in data and makes it possible to add new machines without giving up the old ones.

The first step to a new architecture is to have a new operating system. It must be indpendent of a particular piece of hardware. This doesn't mean an operating system that can run on any processor. It means not being tied to a limited set of hardware like MS-DOS got tied to the PC hardware. The second step is to separate the application programs from the operating system itself. To use networks or multiple processors there must be a clear distinction between the logical and physical structure of the machine. To do otherwise would be to set a limit on what can be done with the operating system.

Bit map graphics and mice are good in some cases but to hobble an entire system with tricks that are not often needed or used is bad. The original use of mice was to allow people who knew little about computers to retreive information from them. They were not ones who had to put information into the computer or the more experienced users who want low cost and high performance. The operating systems like Mac and Atari are complex to the point where they hinder the development of new programs rather than helping. The windows that Microsoft has to sell are no better. Look at any stock broker, they have mulitple screens for dealing with different pieces of information at the same time, not tiny windows on a single screen.

A very promising area to look at for the future is multiple processor machines. With them, when more users are added to a system, more processing power is added also. This makes it possible to have multiple access without the slow down problems associated with trying to share a single CPU among many users. For cost sensitive or low performance users the muliple user approach can be used for lowest cost. For applications where high performance is important multiple processors can be used. If the operating system is independent of the hardware then the same program can be used in both cases.

Another area where multiple processors can be used to advantage is to split the operating system into component parts. For example the file management system can be duplicated for each disk in the system. Then when opening a file on disk A there would be no operating system overhead imposed on the system running disk B. If a disk is not involved then it would take no part in the activity. This allows large numbers of users to all access files at high speed if the load is balanced among different disks. A remote disk and file system can be like a new resource that can be easily added and integrated into a system. A company system can start small and grow to almost any size without requiring that the existing parts be replaced.

An individual workstation can have graphics and icons or not as need or tastes dictate. This will allow some users to access the system with icons but not impose that structure on other users of the system. It also means that some users could have windows and others could have more than one screen. Some users could have a local floppy disk or printer too. This approach to things opens a wide area of possible designs for working.

It is time to start planning for the future while the present generation of computers is still adequate for today. If we don't start now we won't have the next generation when we need it. At Hawthorne Technology we are working on new ways of doing things. All of our programs are compatible with K-OS ONE at the system call level. Our hardware varies a lot. We even use PC Clones for some things. But any program that uses K-OS ONE system calls to access the hardware, and doesn't depend on special terminals, will run on any K-OS ONE system. We intend to keep this compatibility in the future for all systems whether distributed, multitask or single task. You can join us in this by using K-OS ONE or by writing applications to run with it. The number of people using K-OS ONE is increasing every day. There is a growing market for Languages and application software. Anyone interested in doing a package should contact us. We will help out in any way we can.

Why Use a 68000?

Most of the time it is not easy leaving an old processor and going to a new one. On the old processor you are an expert and know all the small things that can and will go wrong. When you switch to a new processor you have to start all over again. So why switch?

The 8 bit machines are limited and there is little room to grow with them. The 68000 on the other hand has enough growth potential to last for many years to come. There are other 32 bit

processors, but none of the others offer the same advantages as the 68000.

After you start working with it, you will find that building hardware with the 68000 is as easy or in many cases easier than building the same thing with an 8 bit processor. For small controller projects there is even an 8 bit bus version (the 68008) that comes in a 48 pin package. The 68000 and 68008 both have an E clock signal output that allows you to directly connect any peripheral device from the 68xx or 65xx families.

In most cases, the cost of doing the software for a project is many times the cost of the hardware. All of the software cost has to be paid for before the first unit is shipped. Hardware is paid for as units are sold. The 68000 is easier to program than the smaller 8 bit machines. The mistake many people make is the idea that just because you have 16 registers you have to use all of them. You don't. Just use the parts that you want and ignore the rest. After you have some experience you can start using the other commands and addressing modes.

Inline code and programs in general can be as small for the 68000 as for any 8 bit machine. The reason many current programs are so large is that they were written for the 8086 processor family, which is sloppy and many of them are written in higher level languages with compilers that don't generate very good code. What was several lines of code for a Z-80 or 6502 can often be done with a single instruction in the 68000.

The main limitation for the 8 bit machines is the small memory space and the small size of the registers available. If you want to work with more than 64k of memory you have to have registers that are big enough to hold an address. This means that without a full 32 bit register you will spend lots of time and effort working on address pointers that sould be trivial.

The 68000 is much faster than an 8 bit machine for arithmetic and full size pointer manipulation. For simple 8 bit operations like those encountered in text editors it is true that the Z-80 is very hard to beat. But if you need more memory to edit a large file or a lot of features are added to the editor, things become difficult. With a large address space you don't have to page parts of the program into memory from the disk. For spread sheets and arithmetic, the larger register size of the 68000 is faster by far.

When you look at the small cost difference between the 68000 and the older machines the choice becomes easier. Keep the 8 bit machines for existing products or for very high volume or where there is not much programming involved but for new products go with the 68000. It is easier to program, faster, and has more of a future.

Starting With HTPL

Welcome to HTPL programming. If you are familiar with Pascal, Modula or Forth then HTPL will have many parts that you already know. From Forth we borrowed the use of RPN notation for expressions. From Pascal and Modula we borrowed a structure. HTPL is good for writing small, fast programs. It can also be extended to fit any special needs in other programming areas.

To start learning any new language it helps to see a complete example in that language. The example can then be related to the same program in a language you are more familiar with. This example is a simple but complete HTPL program that displays "Hello World!" on the console.

The first line is a comment. When anything is placed in parenthesis in an HTPL program it is treated as a comment and ignored. The word "root" indicates that this is not an overlay, and tells the compiler to include the runtime library with the generated object code. The word "program" tells where the program will start executing when it is run. The main part of the program continues until the first "end". The second "end" indicates the end of the entire file being compiled. The words in the double quote marks are a string constant. When a string constant is encountered the contents of the string are saved in a data area and the address of the string is placed on the evaluation stack. The word "sprint" is a call to a run time library routine to print the null terminated string. The 13 is the numeric value of a carriage return character. It is pushed on the stack. The word "putc" is another library routine that prints the low 8 bits of the top of the stack as a single character. The "10 putc" sends out a linefeed character.

Sample Program

```
root
program
  "Hello World!" sprint
  13 putc 10 putc
  end
end
```

This is a complete HTPL program. When it is run it will display "Hello World!" on the terminal. Most of the tokens are refered to as words in HTPL just like in Forth.

Editor's Note: When this file is compiled, the executable BIN file including the run time library is only 1,446 bytes. This is much smaller than a similar Pascal or C program.

HTPL Compile and Run

To compile and run an HTPL program you first write the program using any editor. The compiler assumes that all characters have the high bit a zero. The output of the compiler is an executable binary file.

To compile a program type HTPL at the command line prompt. After the compiler is loaded it will prompt for the name of the first input file. Next it will prompt for the name of the output file. Any extension can be given for the output file but the command processor will only try to load and execute files that have the extension ".BIN". Next you will be prompted for options. If you enter an "N", there will be no listing of the source code as the program is compiled. If you put an 'S", there will be no symbol table listing after the program is compiled. The options can be given in any order. The complier reads the source program and any files involved twice. The run time library hex file "HTPLRTL.HEX" must be on the default drive for the compiler to find it. An overlay doesn't include the runtime library so it is not needed. You can include as many source files as you want at compile time so each source file can be kept small to be easier to edit.

Stack Notation

The commands in the manual have a comment describing the stack before and after the call to the routine. This is necessary because in a stack oriented programming environment the programmer has to keep track of the stack. Errors in the size of the stack is perhaps the most common kind of error made.

The letters or words before the "--" are the contents of the stack before the call. The top of the stack is on the far right hand side. The words or letters after the "--" are the contents after returning from the routine or after the word is executed. If a word appears before and not after it has been used up. The number of items before and after the call indicate how the stack will grow or shrink when the routine is run. If a routine calls itself then this

can be used to estimate how many levels of stack will be required to run the program.

```
EXAMPLE:     a b -- c

     Shows the stack change:   b                 top
                               a        c
                              ---      ---       bottom
                             stack    stack
                             before   after
```

What is RPN, and Why Would You Want to Use It?

There are three different ways to mix operators and the things they operate on: prefix, infix, and postfix. These simply mean that the operator comes before the operands, between the operands, or after the operands. Most of the common languages like BASIC, C or PASCAL are infix languages. LISP is the only common prefix language. Postfix languages, refered to as RPN (Reverse Polish Notation), are represented by Forth, PostScript, and HTPL. The Teco editor used RPN. Adding machines all use RPN and most printing desk calculators use RPN. Each notation has its adherents. So why use RPN?

The compiler for an RPN language is smaller and simpler than the compiler for an infix algebraic language. A large portion of most compilers is a syntax analysis routine that converts the source language to an internal RPN format. If the source is RPN this step is eliminated. When a subscripted variable is referenced a lot of code needs to be generated to calculate the address to use. In RPN these calculations are explicit rather than hidden. For expressions, all an RPN compiler needs to do is push any operand on the evaluation stack and call or generate code for any operator.

In an RPN language, user created operators look the same as the built in operators. When a subroutine package is used to extend an infix language the subroutine calls are very different from the built in operators. If the extensions look the same as the built in operators they are easier to use and the whole program has a more natural look about it. It is easy to create a special set of words for graphics, statistics, mathematics or data base programs. By the time a conventional language has been extended very far it starts to look more like LISP than whatever it started out as. An RPN language in contrast looks the same no matter how far it is extended.

An RPN language is much simpler to learn than an algebraic language. There are no rules of associativity or precedence. The operations are done in the order specified. In the C language there are 14 levels of precedence. Some associate left to right and some the other way. With RPN languages things are much simpler. If it is data it goes on the stack. If it is an action word, the action happens.

In an RPN language the programmer has more control over what kind of code is produced. The sequence of operations is given by the source code. You don't have to worry about the compiler rearranging the order to get better code. Even when using an optimizing compiler you are assured that the operations will be executed in the sequence given.

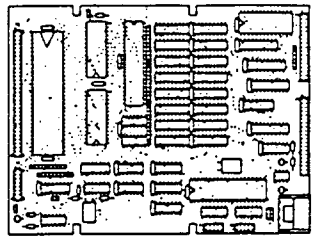
RPN languages are more flexible with the way arguments are passed to subroutines. You can pass parameters by value and parameters by reference in a single call.

The items on the stack become abstract items. They can be used as values or addresses. They can be used as byte pointers, word pointers, long pointers, pointers to structures, or pointers to

68000 SINGLE BOARD COMPUTER

$395.00

32 bit Features / 8 bit Price



-Hardware features:
* 8MHZ 68000 CPU
* 1770 Floppy Controller
* 2 Serial Ports (68681)
* General Purpose Timer
* Centronics Printer Port
* 128K RAM (expandable to 512K on board.)
* Expansion Bus
* 5.75 x 8.0 Inches Mounts to Side of Drive
* +5v 2A, +12 for RS-232
* Power Connector same as disk drive

Add a terminal, disk drive and power, and you will have a powerful 68000 system.

-Software Included:
* K-OS ONE, the 68000 Operating System (source code included)
* Command Processor (w/source)
* Data and File Compatible with MS-DOS
* A 68000 Assembler
* An HTPL Compiler
* A Line Editor

ASSEMBLED AND TESTED ONLY **$395.00**

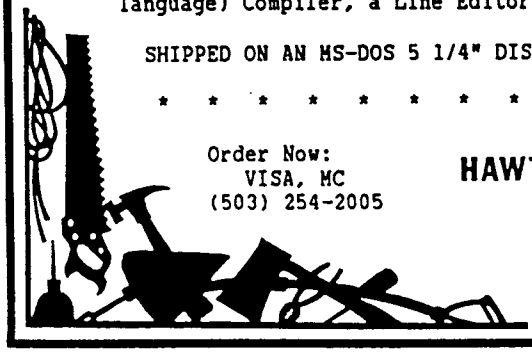
* * * * * * * * * * * * * * * * *

K-OS ONE, 68000 OPERATING SYSTEM

For your existing 68000 hardware, you can get the K-OS ONE Operating System package for only $50.00. K-OS ONE is a powerful, pliable, single user operating system with source code provided for operating system and command processor. It allows you to read and write MS-DOS format diskettes with your 68000 system. The package also contains an Assembler, an HTPL (high level language) Compiler, a Line Editor and manual.

SHIPPED ON AN MS-DOS 5 1/4" DISK. **$50.00**

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



Order Now:
VISA, MC
(503) 254-2005

HAWTHORNE TECHNOLOGY
8836 S. E. Stark
Portland, Or 97216

strings. Different numbers of parameters can be used by the called routine depending on what is found on the stack. A procedure can return a varying number of results depending on what happened. Conventional languages don't offer this kind of flexibility. The 68000 is a very good processor to use with an RPN language. All eight of the address registers can be used as stack pointers. In the other contending micros you have only a single stack pointer and that is used for return addresses. The 68000 also has a very effective set of opcodes that make for small efficient programs.

HTPL has very low overhead on procedure calls. In HTPL there is only a BSR or JSR to get to the procedure and an RTS to get back from the procedure. Any arguments used by the procedure are found on the evaluation stack. This means that there is no need for an explicit transfer of arguments.

Many new languages like Post Script are using RPN because as a subject gets more abstract the use of a stack to hold operands becomes more convenient. The algebraic languages were derived from math equations. When computing is less numeric in nature, it is useful to have a stack for a short term memory to hold what is being worked on.

There are not many books or articles on theory for RPN languages. In many cases this is because writers write about things that are easy to write about. If you look at any book on compilers you find good coverage of syntax and very little coverage of code generating. If you write a compiler you spend lots of time on the code generating and relatively little on the syntax.

Why Not Forth?

Because Forth is the best known of the current RPN languages many of it's quirks are assumed to be in all RPN languages. While some of these disadvantages may be true with Forth, they are not neccessarily true about all RPN languages.

RPN and threaded code are not the same thing. RPN is a way for a programmer to describe the problem to the computer. Threaded code is a technique for generating object code. Threaded code has been popular for Forth on microprocessors because it allows you to create a very fast interpreted instruction set. For 32 bit machines like the 68000 there is no real need to use threaded code.

Incremental compiling is also a technique that is often associated with RPN languages. This was a technique used to create an interactive environment without the slowness of a conventional interpreter. Many RPN languages are now compiled.

As you can see, RPN languages do not need to be feared. The weak points of popular RPN languages have given this method a bad name. One it does not rightly deserve. It may seem like an unnatural method at first. This is due to early mathamatics training. Anyone who has learned to use a 10 key adding machine has learned to use RPN with postfix operators. Most adding machine operators wouldn't recognize the terms, but after their first couple of weeks training, they don't even think about the order they enter the information into the machine. Ask someone you know who uses a 10 key by touch, what order they put the information into the machine. If they don't have a machine they can try it on to find out, they will have to think it through keystroke by keystroke. The actions have become automatic. It isn't so unnatural after all.

HTPL Programming Techniques When starting to use any new language there are lots of little tricks and techniques that you learn to make it easier and faster to write programs. Some of these are very dependent of the kinds of programs that are being written and some are of a much more general nature. For many languages there are collections of algorithms. These can be used to write a good sort program or to manipulate a data structure.

HTPL is a stack oriented language. If a stack is not a familiar thing, code can be produced by making a literal tranlation of simple algebraic code. Frequently used code sequences can be given a name and made into a procedure. The use of many small procedures results in a slight speed penalty but tends to make the object code generated a little smaller.

Because a stack is so easy to use, there is a tendency to try to do too many things on the stack and save too many values there. You should avoid ever having more than four values on the stack at any time. Saving a value in a temporary variable is not that hard. Having the wrong number of items on the evaluation stack is probably the most common error that occurs in RPN languages. The reset command is used to reset the return and evaluation stacks. Sometimes I use reset as a safty mechanism. When I'm not sure if the stacks are OK I use it to force them into a known good condition.

Strings and Characters

Many of the program modules that are used in the K-OS ONE system deal with strings or character manipulation. In the early days of computing, the processing of numbers was most important. Now it is more important to be able to easily manipulate characters. The stack is used to pass single characters or the address of a string. HTPL uses the C convention where a string is a group of bytes that ends with a null or zero byte. A string is referenced by pointing to the first charcter of it. Also by convention, a valid pointer can never be equal to 0. That is refered to as a null pointer and it means that it doesn't point to anything. Because all stack values are 32 bits long, a string can be kept any place in memory.

The first two routines below deal with adding a character to the end of a string that is being built. In both cases a pointer is used to save the character then the pointer is incremented so it will be ready for the next use. In some cases it is possible to have characters and pointers on the stack. In most cases however it will be easier if either the character or the pointer is in memory. A short assembly routine can be added to the run time library to do many of these things if they are used a lot.

In the first case, the destination pointer is on the stack. The item is placed on the stack. Then the over is used to make a copy of the address to store the character. The !1 uses the character and the copy of the pointer. The +1 then increments the pointer for next time.

In the second case, the destination pointer is in a variable and the character is on the stack. First we get the pointer on the stack. We then duplicate the pointer so that we will have a copy of the pointer as it is. We increment the top copy and store it back in the variable that holds the pointer. The other copy of the pointer that wasn't incremented is used by !1 to store the character.

```
Add a character to a string:

    1. if the destination pointer is on the stack:

            @item over !1 +1
```

```
2. if the item is on the stack:

        @pntr dup +1 !pnter !1

                ↓

Get the next character using a pointer and return it:
  @pntr dup +1 !pntr @1

Change lowercase letters to uppercase letters:
  if dup 'a' 'z' range then 32 - end

Looking for first space, (pointer is on stack):
  while dup @1 ' ' <> do +1 end
```

■

Reader's Feedback

(Continued from page 4)

take a 64 pin header and replace the Z80 with a HD64180Z, if the HD64180Z will run at 4 MHz. I think my 64K chips will baulk at 6 MHz.

If this works I plan to remove the 64K chips later and replace them with 256K chips and bring the system up to 6 MHz along with new ROMS.

At this point I am a systems programmer. Also am learning to type and will try to learn assembly language programming. There is a lot to crowd into my later life. I am 72 now.

What do you think of the idea?

Any help you can give me will be greatly appreciated.

Hiram Desantis
1896 Keewin Ave. N.E.
Palm Bay, FL 32905

How about some of you hardware gurus giving Hiram a hand on this project.

Disk Formats

The big issue around here is disk formats. Number one is getting other people's files moved to the Macintosh network, from IBM, CP/M, HP 3½", Tandy 100 3½", etc. Number two is keeping all the files on the CP/M systems accessible when 8" SSSD seems to be obsolete and the CCS 2422 (at least mine) won't read/write the Ampro or Kaypro formats everyone seems to be using for disk exchange.

If Ampro format is the new CP/M "standard," how about publishing all the details & hints on how to read it? Or a series of programs to force the common controller chips to read it (1793, 765A, etc) and only require the proper I/O ports to be patched in?

Maybe my problem is trying to use my Teac 55G (dBM-AT style 8" compatible) drives in their 5" 300 rpm mode, instead of regular 40 track drives... Has anyone made this work?

L.A.

Jay Sage Fan

The past year has been terrific! Jay Sage is a real boost. His insight and the brilliance of his work is monumental. Also, his understanding of those of us with small TPAs (Osb 1, hard drive) is an uplift.

I have used his new "SUB" to make programs leave ZCPR33, run in full TPA, and then return to ZCPR33, fantastic!

Keep Clark Calkins writing about debugging and Thomas Hilton's educational articles.

Thanks for a terrific year.

A.W.

Z80 User

I have two systems — both Z80 based

My "main" system is a TRS-80 Model II with 256K memory, two 8" 1 Mbyte floppies, and a high resolution graphics board. I run OASIS, a multi-user OS on this computer. My other system is a Televido 806 with two 5¼" floppies, running CP/M 2.2x.

I would like to see articles on interfacing hard disks to SCSI controllers — compatibilitiy of various SCSI based controllers with various hard disks, command sets for the controllers, example drivers, comparisions of different controller/disk combinations for speed and ease of use, etc.

I would also like to see more articles on interfacing speech chips (SP0256) and sound chips (AY-3-8210/8212) to various buses, using IBM keyboards and monitors with non-IBM equipment... (Which reminds me, pin outs and signal descriptions for the IBM keyboard, monitors, and other peripherals would be extremely useful to us non- IBM types to support interfacing attempts.)

L.S.

Miscellaneous Reader Comments

How about a wire wrap video board for the IBM PC Bus?. Possibly using one of the new graphics controller IC chips. Suggest you drop CP/M & Z80.

Would like to see more hardware and software articles for MS DOS, especially AT systems.

I am renewing because you have more articles on MS DOS. I am interested in understanding MS DOS so I can write programs such as device drivers or other enhancements to MS DOS.

Like C.D.M. (Reader Feedback, Issue #27), I was afraid that you might be following in the footsteps of Communications and Electronics, which I once thoroughly enjoyed because of its blend of hardware and software material (generally not too complex for my capabilities).

I was about to drop my subscription but decided to wait for issue #27 before I made my decision. Happily (as owner of an SB-180) Jay Sage's column appeared for a third consecutive issue with promises of continuing regularly. This alone was enough to make me reconsider. Also Jon Schneider's article on the HD64180 put the icing on the cake.

Please don't forget us hardware hobbyists (expert though we may not be).

I am still interested in CP/M stuff (North Star). I recently acquired a Sage II and would be interested in an article on how to install K-OS ONE on it.

My systems are Ampro Z80 Little Board (1A), STD Homebrew, expanded Little Board on STD Bus.

Just in — Hawthorne Little Giant. It's gonna need hack ports of: VDO, Disk7/Sweep, NULU, Small C, Superzap, Fbad, MDM740, Unera, Vfiler, DDT/SID, Config, Multidsk.

I'm designing/programming boards for

(Continued on page 41)

Detecting the 8087 Math Chip

Temperature Sensitive Software

by E. Clay Buchanan III

There I was happily hacking in 8086 assembly language when a customer phoned to describe a problem he was having with one of my company's older graphics products. The symptoms he described were unbelievable. The same program which ran fine on an XT or PC would fail on about one third of all the ATs at the customer site. The customer had gotten the program to work on one or two of the failing ATs by changing their motherboards although one AT required three different boards before one worked. In addition one AT would fail if recently powered up but once it had been running for 30 minutes or so it worked! So the customer asked me "What's the problem?"

My answer was simple "Sorry, I can't answer your problem right now, I have to jump out a window first." Fortunately I was saved at the ledge by a wise sage who told me that symptoms like these in graphics or math software are almost always the result of the program improperly sensing the presence or absence of the 8087 or 80287 math coprocessor. Since my DOS technical reference manual doesn't state how to detect the presence of an 8087 and the AT technical reference doesn't seem to mention it either I looked at a very good book called "The IBM PC from the Inside Out," by Murray Sargent III and Richard L. Shoemaker, copyright 1986, ISBN 0-201-06918-0, Published by Addison-Wesley. This book has a great deal of hardware information in it which includes the 8087 math coprocessor. On the cover is a subtitle "Includes the PC AT" and with its wealth of logic diagrams and TTL circuit descriptions it seems the authors must have an extensive background in hardware design. On page 167 is a simple program to detect the presence of an 8087:

```
ists87  dw      0ffh

chk87:  fninit          ;Try to initialize the 8087 or 80287
        mov   cx,64h    ;Wait long enough to let it finish
chk872: loop  chk872    ;(ifs it's there)
        fnstsw ists87   ;Store status word
        ret
```

The logic of this code seems perfect. Order the 8087 to store its status word in memory. If the store instruction changes memory then an 8087 must be present. There is only one problem. This little piece of code failed on one of the three "True Blue" IBM ATs I tried it on. It worked on every PC and XT I tried. For reasons known only to hardware designers one of the ATs I ran this on changed memory from 0ffh to 0ddh even though no 8087 was present. As a result the above code would improperly detect an 8087 and thousands of floating point operations would go zooming off to a nonexistent chip. From there its just a matter of time before an FWAIT instruction or some other combination of floating point instructions hang the AT or result in a wild jump into interrupt vectors. Clearly I was looking at a "feature" of the AT motherboard design.

I must emphasize Mr. Sargent and Mr. Shoemaker are quite knowledgeable on the AT motherboard. Page 230 of their book has a section titled "IBM PC AT System Board" with a schematic. Whole sections of the book are devoted to the use of the 8087. All in all it's a first rate book. But somehow these experts were a little off in detecting the math coprocessor. Everything was pointing to this as the problem so I began looking through our product's code to find where we detected the presence or absence of the 8087.

The product was written almost entirely in 8086 assembly language but some floating point code was written in Lattice C. I don't know the exact version of Lattice used but it must have been before version 2.14. Lattice was then advertising that its run time libraries automatically detected the 8087 and would use the chip if present or emulate it if absent. Stepping through the code with debug I quickly came across the sensing code. It was as follows:

```
mov word ptr [0134], 0fffh
fstsw [0134]
cmp word ptr [0134], 0fffh
```

Almost identical to the code in Sargent's and Shoemaker's book and consequently subject to failing on one third of the ATs I tried it on. When I relinked the product with version 2.14 of the Lattice compiler the program worked on all the ATs I tried it on. Stepping through the code showed the sensing routine had been slightly changed to:

```
mov word ptr [0140], 0fffh
fstsw [0140]
test word ptr [0140], 0b8bfh
```

Changing memory is no longer enough to indicate an 8087. A more complex data pattern is now required. Obviously the Lattice engineers had detected or heard of the bug and changed their code for version 2.14. When I ran the above code on one AT the 0ffffh value changed to 3eddh but the test for 0b8bfh worked and the routine correctly identified the absence of an 80287 chip.

Just to see if anyone else had noticed this problem I went through my back copies of Dr. Dobb's Journal and found a reference to the 8087 and 80287 in the September 1985 issue. The 16-BIT Software Toolbox section of that issue gave the code to detect a math coprocessor. It was almost the same as the Lattice C routine except it used a fnstcw instruction and then checked for a value of 03h as the high byte of the stored word i.e. variable + 1 in memory. The article warned about a difference between the 8087 and 80287. Apparently the 8087 stores a value of 03ffh if present and the 80287 stores a different value in the low byte. To be safe the article recommends checking for the 3. The article says this is the "accepted strategy" for detecting a coprocessor.

Where is IBM in all this? Isn't there an IBM approved way to detect the presence or absence of a math coprocessor? Yes, there is. But don't expect to find it where everyone will look. I expected to find a line in the AT technical reference manual that read something like "coprocessor, detection of." But nooOOOoo. That would be too easy. Ditto for the DOS technical reference

manual. I finally obtained a technical note from IBM on how to do it:

```
        Int 11h
        and ax,2
        Jz  mp_not_in       ;Jump if no math coprocessor
        nop                 ;Has coprocessor

mp_not_in:
```

IBM's solution is clear, use BIOS interrupt 11h to detect the 8087. The note goes on to say "This procedure applies to the PC, XT, and Portable, as well as the AT. On the PC, XT, and Portable the user must have set the switch on the planar board properly per published instructions. On the AT the Power On Systems Test (POST) code takes care of this initialization." Finally the note says "Other techniques to check for the presence of a math co-processor may yield unreliable results and, therefore, are not supported by IBM." No kidding.

I'm sure Lattice knew about int 11h when they coded their run time floating point library. Why didn't they use it? Obviously I can't be certain but I can guess since I write compiler run time libraries for a living. A constant goal of any library is to be portable. Write the routine once, debug it once, and then forget about it. Making BIOS calls in your library introduces a dependency. Your library now assumes an IBM compatible BIOS to work correctly. What if you're porting to a clone that doesn't have an int 11h? A BIOS interrupt takes more time than a status word store and besides, a store and compare seems so simple and self contained. Who would have thought IBM would design a computer that responds to instructions to nonexistent chips? XTs and PCs don't. Most ATs don't and although I wasn't able to verify it the customer was certain some ATs were temperature sensitive. So the choice is up to you. Personally I'm a big advocate of nonjudgemental design. Use or don't use the BIOS interrupt depending on your needs. Just be aware that IBM accepts no substitutes for int 11h.

If you come across software with these symptoms don't just set a breakpoint at the int 11h routine and assume that because some of your software is doing BIOS calls to check for the 8087 all of the software uses int 11h. In our product two different C compilers were used and one used int 11h while the other used status word stores.

In the end I replaced our product's status word store instructions with int 11h. I decided to adhere to the IBM standard. This time at least. I take comfort in hearing that even Intel has trouble with the interface between the 80386 cpu and the 80387 floating point chip. It seems Intel is coming out with a new step of the 80386 to fix "minor" glitches in the math coprocessor interface. Keep your debugger handy and your computer cool. You never know when a floating point bug is creeping around. ■

Wanted

• **Clocks** — We need articles discussing the pros and cons of various time keeping methods. I have choosen to use the KCT Ztime-I because I feel it is best for **my** aplications, and I have an article in preparation.

Submit **your** clock related programs and ideas — many different approaches are better than just one.

• **68000** — TCJ is planning extensive coverage of the 68000 series and we need articles on assembly language programming, interfacing to peripheral chips (parallel and series I/O, disk controller, graphics, etc.), and general articles on the 68000.

• **CUG Disk Reviews** — The C User's Group library contains many useful disks. I am reviewing several disks, but need help because there are more than I can handle alone.

Contact TCJ if you would like to review CUG disks.

TCJ is User Supported

If You Don't Contribute Anything....

....Then Don't Expect Anything

Floppy Disk Track Structure

A Look at Disk Control Information & Data Capacity

by Dr. Edwin Thall, Wayne General & Technical College

The floppy diskette, a popular mass-storage medium, contains more than just the user's data. The MS/PC-DOS 360K format delegates approximately one-fourth of usable diskette space for purposes other than data. Diskette control information is required to locate, separate, and check validity of data. In this article, I analyze the diskette track structure generated by the hardware of IBM PCs and compatibles. I will demonstrate how control information can be displayed and describe how it influences data capacity.

Track Strusture

The track organization of the 360K format is illustrated in Figure 1. Each side of the 5¼ inch double density diskette possesses 40 concentric tracks extending from track 0 on the periphery to track 39 near the hub. Although more space is available for storage on outer tracks, the hardware requires a constant number of bytes for all tracks. This number, approximately 6,200, is determined by the size of the inner tracks.

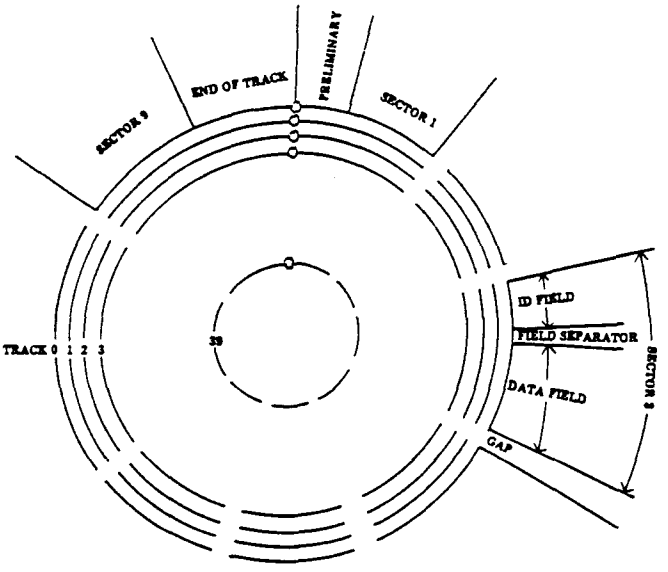


Figure 1. Track structure (360K format)

The format procedure creates the track structure needed to store and locate data. A 146 byte track preliminary begins immediately after the index hole and consists of:

```
80 bytes 4EH
12 bytes 00 (sync field)
 4 bytes C2C2C2FCH (address mark)
50 bytes 4EH
```

Nine equally spaced sectors follow and unused area is filled to the end of the track with 4EH. Sectors, the fundamental unit of diskette information, represent the smallest part of a track that can be read or written.

What's In A Sector?

The exact structure of a newly formatted sector is provided in Figure 2. Besides data, sectors contain 62 bytes of control information. Every sector has two major components: the identification (ID) field and the data field. As the name suggests, the data field storehouses user data and is the most important part of a sector. The ID field holds the information needed to locate sectors. Let's examine sector components in detail.

```
Sector ID Field:     12 bytes  00          (sync field)
                      4 bytes  A1A1A1FEH   (ID address mark)
                      4 bytes  CHRN        (sector ID)
                      2 bytes  CRC1        (error check)

Field Separator:     22 bytes  4EH

Sector Data Field:   12 bytes  00          (sync field)
                      4 bytes  A1A1A1FBH   (data address mark)
                    512 bytes  F6H         (data)
                      2 bytes  CRC2        (error check)

Sector Gap:          80 bytes  4EH
```

Figure 2. Structure of newly formatted sector

The ID field begins with a sync field followed by the ID address mark. Sync fields, 12 bytes of 00, forewarn the coming of address marks. You are probably aware that information is recorded in serial bit stream (1 and 0) on the coated surface of a diskette. When writing data, a change or "transition" is represented by "1" and lack of transition by "0." Double density diskettes, based on the MFM (modified frequency modulation) encoding scheme, store data in bit cells large enough to hold two transitions. The extra transition, known as a "clock," marks the start of a cell. A large string of zero bits (lack of transition) can prevent the detector from locating the beginning of a cell. Clock transitions, stored between consecutive zero data bits, identify the start of a cell.

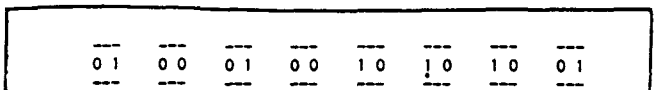


Figure 3. MFM encoding scheme for A1H (1010 0001)

To demonstrate how clock transitions are distributed, consider the bits in A1H (1010 0001) as they are stored in eight consecutive cells (see Figure 3). The first bit in every cell specifies the clock and the second bit data. Note the clock transitions (arrows)

Dr. Edwin Thall, Professor of Chemistry at The Wayne General and Technical College of The University of Akron, teaches chemistry and computer programming.

between consecutive zero data bits. Address marks are not given clock transitions and vary physically from other data bits. The four-byte field A1A1A1FEH would normally possess nine clock transitions, but contains none when written as the ID address mark. This unique pattern enables the computer to distinguish address marks from everything else.

The CHRN follows the ID address mark and is the most important component of the ID field. Formatted sectors are identified for future read/write operations by this four-byte field. The four CHRN parameters specify:

```
C  Track number      (00-27H)
H  Head number       (00=side 0, 01=side 1)
R  Sector number     (01,02,03,...)
N  Sector length     (128*2**N, N=0,1,2,3,4,5)
```

The fifth sector on track 07, head 00, and holding 512 bytes of data is identified by 07000502.

Cyclic redundancy checks (CRC) test the correct reading of a sector's identification (CHRN) and data block. The technique depends on an algorithm to verify the accuracy of data written to a sector. IBM and compatibles rely on the CRC-CCITT error detection code. This technique is based on the polynomial:

```
G(X) = X**16 + X**12 + X**5 + 1

The equation may be restated with 17 binary coefficients
(underlined) as:

1(X**16)+0(X**15)+0(X**14)+0(X**13)+1(X**12)+0(X**11)+...
-       -       -       -       -       -
```

The coefficients of the polynomial are represented by a 17 digit binary number (1 0001 0000 0010 0001). The CRC method takes an entire block of data and divides it by this 17 bit number. The complement of the remainder, a two-byte value, is stored as the CRC.

Every sector is given two cyclic redundancy checks. The first check (CRC1) is determined from the four-byte CHRN. When accessing sectors, CRC1 verifies that the correct sector was located. The sector ID field ends with the two-byte error check.

The ID and data fields are separated by a 22 byte gap. A sync field followed by address mark (A1A1A1FBH) initiate the data field and signal the approach of data. The size of the data is based on the algorithm:

```
data length = 128(N**2)
```

The sector length parameter (N) may take on values of 0-5 (128, 256, 512, 1024, 2048, or 4096 bytes). However, values of N greater than 5 lead to chaos. For example, N=6 (8192 bytes) requires a format that goes beyond one revolution of the disk. When a second pass to format the track is made, the ID field is overwritten and the sector is destroyed.

The data field ends with the second cyclic redundancy check (CRC2). For this error check, the entire data block is divided by the 17 bit polynomial, and the complement of the remainder stored as CRC2. Whenever data is read from a sector, the CRC is calculated and compared to the CRC2. An error message results if the two values disagree.

Sectors are separated by gaps. The standard size is 80 bytes but gaps as small as two bytes may be used. A trade-off exists between disk capacity and gap size. If the size of a gap is reduced, the number of sectors can usually be increased. But remember, smaller gaps increase the possibility of overwriting the next sector's ID field. This is most likely to be a problem when sectors are formatted on one machine and then written to by another.

Viewing Control Information

Now that you are familiar with the structure of tracks and sectors, I will show you how to display diskette control information. The method, which I call sector overlap, allows you to look at the preliminary, sector components, and gaps. You will need the DOS DEBUG utility and a scratch diskette.

Address	Defines	Default Value
0:525H	sector size (N)	N=2 (512 bytes)
0:526H	sectors/track	9
0:529H	sector gap	50H (80 bytes)
0:52AH	format fill	F6H

Table 1. The four track parameters

A formatted track is defined by four track (see Table 1) plus four sector (CHRN) parameters. The track parameters are stored in the diskette parameter table (address 0:522-0:52CH) during the computer start-up. When a track is formatted, the "track" N determines the number of bytes written to each sector. However, once the format is complete, the "sector" N takes over and regulates the number of bytes read or written. If the "sector" N exceeds the "track" N, diskette control information can be read. Our strategy is to format a track with nine standard sectors (N=2), but assign a CHRN to the fifth sector corresponding to 4096 bytes (N=5). Figure 4 lists the assembly language program to carry out such a format to track 01/side 00. Load the DEBUG utility and place your scratch diskette in drive A. Enter the program (omit comments) by means of the assemble command (A100H) and then execute.

```
A>DEBUG
-A100 (enter program)
-G

   To read Sector 5 into the buffer area beginning at offset
1000H, enter the following modifications to the format
program:

-E 105
DS:0105  05.02

-E 107
DS:0107  09.01

-E 108
DS:0108  01.05

-E 111
DS:0111  17.00  01.10
```

```
DS:0100  MOV  AH,00          ;RESET DISK CONTROLLER
DS:0102  INT  13             ;BIOS DISK I/O
DS:0104  MOV  AH,05          ;FORMAT TRACK
DS:0106  MOV  AL,09          ;9 SECTORS
DS:0108  MOV  CH,01          ;STARTING AT TRACK  01
DS:010A  MOV  CL,01          ;     "       SECTOR 01
DS:010C  MOV  DH,00          ;     "       HEAD   00
DS:010E  MOV  DL,00          ;     "       DRIVE   A
DS:0110  MOV  BX,0117        ;POINT TO CHRN
DS:0113  INT  13             ;BIOS DISK I/O
DS:0115  INT  20             ;RETURN TO DOS
DS:0117  DB   01 00 01 02    ;CHRN ENTRIES
DS:011B  DB   01 00 02 02
DS:011F  DB   01 00 03 02
DS:0123  DB   01 00 04 02
DS:0127  DB   01 00 05 05    ;LARGE SECTOR N
DS:012B  DB   01 00 06 02
DS:012F  DB   01 00 07 02
DS:0133  DB   01 00 08 02
DS:0137  DB   01 00 09 02
```

Figure 4. Assembly language program to format track with overlapping sectors

The program to read Sector 5 appears in Figure 5. Before you attempt to read Sector 5, you must change the data length parameter (address 0000:0525H) from N=2 to N=5.

```
-E 0:525
0000:0525  02.05
-G
```

```
DS:010C  MOV  AH,00          ;RESET DISK CONTROLLER
DS:010E  INT  13             ;BIOS DISK I/O
DS:0110  MOV  AH,02          ;READ TRACK
DS:0112  MOV  AL,05          ;1 SECTOR
DS:0114  MOV  CH,01          ;STARTING AT TRACK  01
DS:0116  MOV  CL,01          ;     "        SECTOR 05
DS:0118  MOV  DH,00          ;     "        HEAD   00
DS:011A  MOV  DL,00          ;     "        DRIVE   A
DS:011C  MOV  BX,1000        ;POINT TO BUFFER AREA
DS:011F  INT  13             ;BIOS DISK I/O
DS:0121  INT  20             ;RETURN TO DOS
```

Figure 5. Assembly language program to read Sector 5

The read operation stores 4096 bytes into the buffer area at offsets 1000-1FFFH. You can display these locations with the dump command:

```
-D 1000    (displays offsets 1000-107FH)
```

Starting with the data in Sector 5 (512 bytes of F6H), you can view 4096 consecutive diskette storage locations. Table 2 offers a summary of these locations (offsets 1000-1FFFH).

Offset	Description
1000-1FFFH	512 bytes F6H (sector 5)
1200-1201	CRC2
1202-1251	sector gap
1252-125D	ID sync field (sector 6 begins)
125C-1261	ID address mark
1262-1265	CHRN
1266-1267	CRC1
1268-127D	field separator
127E-1289	data sync field
128A-128D	data address mark
128E-148D	512 bytes F6H
148E-148F	CRC2 (sector 6 ends)
1490-1C39	sectors 7-9
1C3A-1D65	4EH to end of track
1D66-1DF8	track preliminary
1DF9-1FFF	sector 1

Table 2. Summary of 4096 consecutive diskette locations

Sector 5 required a "long" read that goes beyond the end of the track. The preliminary is read during a second pass of the track, and there is a one in eight chance of reading these bits in the proper synchronization. I had to make several runs before synchronization was correct. Figure 6 shows the results of an unsuccessful attempt preliminary. In this figure, the preliminary is read one bit too early and the initial 80 bytes (offsets 1D66-1DB6H) appear as 27H. Reading a series of hexadecimal 4E (0100 1110) one bit out of synchronization gives:

```
                  ---------------------------------
4E4E4E4EH = 0100 1110  0100 1110  0100 1110  0100 1110
                  ---------------------------------
                     27H         27H         27H

The preliminary address mark is also read one bit early and
appears as 6161617EH instead of C2C2C2FCH.

            -----------------------------------------
C2C2C2FCH = 0  1100 0010  1100 0010  1100 0010  1111 1100
            -----------------------------------------
               61H         61H         61H         7EH
```

```
-D 1D66,1DF6
DS:1D60                    A7 27-27 27 27 27 27 27 27 27
DS:1D70  27 27 27 27 27 27 27 27-27 27 27 27 27 27 27 27
DS:1D80  27 27 27 27 27 27 27 27-27 27 27 27 27 27 27 27
DS:1D90  27 27 27 27 27 27 27 27-27 27 27 27 27 27 27 27
DS:1DA0  27 27 27 27 27 27 27 27-27 27 27 27 27 27 27 27
DS:1DB0  27 27 27 27 27 27 27 27-00-00 00 00 00 00 00 00
DS:1DC0  00 00 00 50 D0 D0 FF 00-80 00 81 5E 6D A7 27 27
DS:1DD0  27 27 27 27 27 27 27 27-27-27 27 27 27 27 27 27
DS:1DE0  27 27 27 27 27 27 27 27-27-27 27 27 27 27 27 27
DS:1DF0  27 27 27 27 27 27 27
```

Figure 6. Track preliminary (one bit out of sync)

Run the format program again to help remedy synchronization problems. If you are unsuccessful after a few attempts, format a different track or diskette.

A track is considered unformatted when the preliminary is overwritten. If you should write 4096 bytes to Sector 5, the preliminary and Sectors 1, 6, 7, 8, and 9 are destroyed. However, you will still be able to read/write Sectors 2, 3, and 4.

Data Capacity

I conclude this article by examining the effect diskette control information has on data capacity. How many bytes of data can the diskette accommodate? Which of the six allowed sector sizes leads to the maximum storage of data? Before reporting the results of my investigation, I'll show you how to predict data capacity.

I calculate the total track capacity, including diskette control information, at 6253 bytes. Here is the breakdown for the 360K format:

```
 146 bytes   track Preliminary
4608         data (9*512)
 558         sector control information (9*62)
 640         gap between sectors (8*80)
 301         to end of track
```

The percent utilization of diskette space comes in at 73.7% (4608/6253) for the 360K format. Let's determine the maximum number of sectors (512 byte size) possible for a single track. For ten sectors, the number of bytes required are:

```
 146 bytes   track Preliminary
5120         data (10*512)
 620         sector control information (10*62)
 720         gap between sectors (9*80)
----
6606 bytes
```

To accommodate the ten sectors, 353 bytes must be eliminated from the track. The preliminary (146 bytes) and the sector control information (620 bytes) cannot be altered. If the track limit is to stay within 6253 bytes, the gap between sectors must not exceed 40 bytes. For ten sectors per track, the diskette capacity is computed as:

$$\text{capacity} = \frac{(\text{disk sides})(\text{tracks/side})(\text{sectors/track})(\text{size})}{1024}$$

$$= \frac{(2)\ (40)\ (10)\ (512)}{1024} = 400\text{K}$$

Any attempt to format a track with more than ten sectors of this size, regardless of gap size, is doomed to fail. The eleventh sector will overwrite the preliminary and the first sector of the track.

To facilitate my investigation of data capacity, I relied on the Disk Explorer. This handy utility, distributed by Quaid Software Limited, makes it easy to format a track. From the "edit format" screen, you need only define the four track plus four sector

```
Edit Format                    C  H  R  N  SC   flag
                               1  0  1  2
drive:track.head a:0.0         1  0  2  2
                               1  0  3  2
  N  SC  GPL   D kind          1  0  4  2
  2  10    2 246               1  0  5  2
                               1  0  6  2
                               1  0  7  2
   key   purpose               1  0  8  2
    select field               1  0  9  2
  PgUp   edit command          1  0 10  2
  PgDn   edit id's
   Esc   return
   + -   change field
    F5   format track
 enter   read sector
```

Figure 7. The Disk Explorer (edit format screen)

Sector size	Sectors/Track	Diskette Capacity
128 bytes	32	320K
256	19	380K
512	10	400K
1024	5	400K
2048	2	320K
4096	1	320K

Table 3. Data capacities for the six sector sizes

parameters. To maximize the number of sectors per track, I set the sector gap length at two bytes. Figure 7 illustrates the screen that I selected to format ten sectors to track 01/side 00. The flags (none showing) signal when sectors are formatted unsuccessfully.

A summary of my results may be seen in Table 3. The Disk Explorer permits a maximum of 24 sectors per track and the first format (32 sectors) was generated with a program similar to the one in Figure 4. The other five formats were produced with the Disk Explorer. For the six formats listed, I verified that all sectors could be read. ■

Reader's Feedback

(Continued from page 35)

STD & MIDI, and am targeting 'electronic musician', etc, for MIDI/Audio stuff.

What I like (at this time in my work and while in school): (1) Review the languages (i.e. C, Pascal, True Basic, Fortran, assembly) for the PC & RS1. (2) A/D D/A — hardware, software, plotting on dot and plotter machines for the PC. (3) How about AD/DA for something like C-64? This can free up my IBM when I'm doing 24 hour monitoring and 8-bit is OK. (4) Video/audio manipulation via digitization.

At home I use: (1) Xerox 820-1 with two 5¼" DSQD and 2 8" SSSD, currently updating to 1M RAM and QP/M OS. (2) Apple][+ with 6MHz Z80 card. (3) Will also soon start playing with a surplus Morrow MD-2 board recently acquired.

At work I do machine control applications in Forth on STD bus Z80s, and also use a MicroMint SB-180.

I am interested in Forth, real-time machine control applications; beyond-beginner treatment of algorithms, hardware, and topics related to above listed interests; newer, non-PC, inexpensive processing and control engines — like the SB-180 and Hawthorne Technolog's Tiny Giant/K-OS

I would like to see articles on hard disk drives and their controllers. How to trouble shoot them, set them up, how to program the controllers, what is standard and what is not for CP/M, MS DOS, Kaypro, IBM, SCSI, SASI.

I use a IBM XT at work, and have an Osborne 1 at home. This coming winter I plan on using some type of microprocessor to control and operate a model railroad. I/O devices are of interest to me.

I use a Corona (CORDATA) MS DOS "clone". I'm interested in coprocessor boards (Z80, Hitachi HD64180, or Motorola 68XXX) for MS DOS computers, and help in finding any OEM's that sell completely assembled systems using the Hitachi CPU — I am definitley not a "Hacker" — but am very interested in the "8-bit revolution."

I would like articles on: (1) Design/Construction articles: a) 68000 based machines and interfaces to peripherals such as floppy drives, hard disks, etc. b) use of computers in on-line, real time data acquisition, process control, etc. c) image processing. (2) Software articles: a) cross-assembler for 6502, 68000. b) image processing, file compression schemes, etc. c) AI programs that work! (i.e., neural nets), etc.

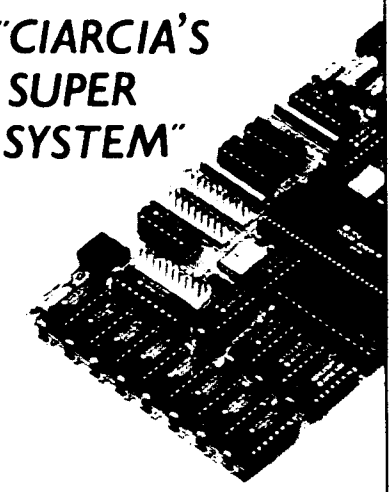
I like the articles on Operating Systems and programming in C, Forth, and Pascal. The feedback loop control article was good.

I use a SB180 with COMM 180, ZCPR3, and the TERM-MITE ST1000 terminal controller. I'd like more articles like The Art of Source Code Generation and Disk Parameters. ■

Editor's Note: Send your comments on the above plus information on what you're doing and what you would like to see. Contact TCJ if these give you an idea for an article that you'd like to write.

Byte Magazine called it.

"CIARCIA'S SUPER SYSTEM"



The SB180 Single Board Computer

Featured on the cover of Byte, Sept. 1985, the SB180 lets CP/M users upgrade to a fast, 4" x 7½" single board system

- **6MHz 64180 CPU**
  (Z80 instruction superset), 256K RAM, 8K Monitor ROM with device test, disk format, read/write.
- **Mini/Micro Floppy Controller**
  (1-4 drives, Single/Double Density, 1-2 sided, 40/77/80 track 3½", 5¼" and 8" drives).
- **Measures 4" x 7½" with mounting holes**
- **One Centronics Printer Port**
- **Two RS232C Serial Ports**
  (75-19,200 baud with console port auto-baud rate select).
- **ZCPR3 (CP/M 2.2/3 compatible)**
- **Multiple disk formats supported**
- **Menu-based system customization**

New Low Prices

SB180-1
SB180 computer board w/256K bytes RAM and ROM monitor **$299.00**

SB180-1-20
same as above w/ZCPR3, ZRDOS and BIOS source **$399.00**

COMM180-S
SCSI interface **$150.00**

Now Available

TURBO MODULA-2.....**$69.00**
TURBO MODULA-2 with Graphix Toolbox.....**$89.00**

TO ORDER CALL TOLL FREE
1-800-635-3355

TELEX
643331

For Technical Information or in CT, call:
1-203-871-6170

Micromint, Inc.
4 Park Street
Vernon, CT 06066

Back Issues Available:

Issue Number 1:
- RS-232 Interface Part One
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part Two
- Build Hardware Print Spooler: Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for the Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating, and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board: Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 14:
- Hardware Tricks
- Controlling the Hayes Micromodem II from Assembly Language, Part 1
- S-100 8 to 16 Bit RAM Conversion
- Time-Frequency Domain Analysis
- BASE: Part Two
- Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part 2

Issue Number 15:
- Interfacing the 6522 to the Apple II
- Interfacing Tips & Troubles: Building a Poor-Man's Logic Analyzer
- Controlling the Hayes Micromodem II From Assembly Language, Part 2
- 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 & 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 & Troubles: Memory Mapped I/O on the ZX81

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1
- The Computer Corner

Issue Number 19:
- Using The Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part Seven
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking and Windows with CP/M: A Review of MTBASIC
- The Computer Corner

Issue Number 20:
- Designing an 8035 SBC
- Using Apple 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
- 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
- Using Turbo Pascal ISAM Files
- The AMPRO Little Board Column
- The Computer Corner

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

Issue Number 24:
- Selecting and Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assembly Code for CP/M
- The C Column: Software Text Filters
- AMPRO 186 Column: Installing MS-DOS Software
- The Z Column
- NEW-DOS: The CCP Internal Commands
- ZTIME-1: A Realtime Clock for the AMPRO Z-80 Little Board
- The Computer Corner

Issue Number 25:
- Repairing & Modifying Printed Circuits
- Z-Com vs Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro Little Board
- Building a SCSI Adapter
- New-DOS: CCP Internal Commands
- Ampro '186: Networking with SuperDUO
- ZSIG Column
- The Computer Corner

Issue Number 26:
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside AMPRO Computers
- NEW-DOS: The CCP Commands Continued
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- The Computer Corner

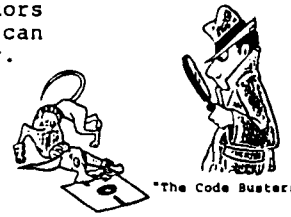
Issue Number 27:
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis and Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program for Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying The CP/M Disk Parameter Block for Foreign Disk Formats
- The Computer Corner

Issue Number 28:

- Starting Your Own BBS: What it takes to run a BBS.
- Build an A/D Converter for the Ampro L.B.: A low cost one chip A/D converter.
- The Hitachi HD64180: Part 2, Setting the wait states & RAM refresh, using the PRT, and DMA.
- Using SCSI for Real Time Control: Separating the memory & I/O buses.
- An Open Letter to STD-Bus Manufacturers: Getting an industrial control job done.
- Programming Style: User interfacing and interaction.
- Patching Turbo Pascal: Using disassembled Z80 source code to modify TP.
- Choosing a Language for Machine Control: The advantages of a compiled RPN Forth like language.

Ever Wondered What Makes *TURBOPASCAL* Tick?

Source Code Generators by C. C. Software can give you the answer.



"The darndest thing I ever did see..."
"... if you're at all interested in what's going on in your system, it's worth it."
Jerry Pournelle, BYTE, Sept '83

The SCG-TP program produces fully commented and labeled source code for your TURBO-Pascal system. To modify, just edit and assemble. Version 3.00A (Z80) is $45. SCG's available for CP/M 2.2 ($45) and CP/M+ ($75). Please include $1.50 postage (in Calif add 6.5%).

C. C. Software, 1907 Alvarado Ave. Dept M
Walnut Creek, CA 94596 (415)939-8153

CP/M is a registered trademark of Digital Research, Inc.
TURBO Pascal is a trademark of Borland International

TCJ ORDER FORM

Subscriptions		U.S.	Canada	Surface Foreign	Total
6 issues per year ☐ New ☐ Renewal	1 year	$16.00	$22.00	$24.00	
	2 years	$28.00	$42.00		
Back Issues		$3.50 ea.	$3.50 ea.	$4.75 ea.	
Six or more #'s		$3.00 ea.	$3.00 ea	$4.25 ea.	
				Total Enclosed	

All funds must be in U.S. dollars on a U.S. bank.

☐ 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

(Continued from page 2)

Technical Support

The computer industry, just like most other U.S. industries, is driven by sales. Consumable items such as food, soap, beer, and cigarettes which are used up must be purchased over and over again, so there is a lot of advertising money available for existing products which remain in production for a long time. Items such as computers, peripherals, and software tend to be used for a long time and only replaced with new revised products, so advertising money is available only for the newest products.

This means that the people who want to learn more about the older computer systems and utility programs receive little support from advertisiers who only want to sell them something new and different. That's why the big glossy magazines concentrate on the new products which their advertisers need to sell instead of supporting the users who want to learn about what they already have. It also means that it is up to us to provide our own technical support by sharing our knowledge with each other.

The lack of support is especially noticed by CP/M users, because very few vendors are advertising CP/M related products, and most magazines have abandoned CP/M because it is not generating enough advertising income. TCJ is here to serve **your** needs, but we can't do it if we don't hear from you — if you don't contribute anything, don't expect anything.

Do You Need to Program?

A writer in another publication stated that there is no longer a need to program because everything you need has already been written and is available as sharewhare or in the public domain.

I agree that some people don't need to program, but I disagree with the blanket statement that no one has to program. It all depends on what you are doing with your computer, and anyone who makes such a statement has a very limited view based on what they see. Someone who only needs a wordprocessor, a spreadsheet, and an accounting program can probably get by without doing any programming as long as they are content to use the programs exactly as they are written using the exact hardware and peripherals for which they were designed — but heaven help them if they want to change printers or modify a report format.

I am constantly writing short programs for the office. Most of them are for filtering text files, managing data files, or to drive the printers and the typesetter. These are primarily non-standard applications for which I doubt there are low cost programs readily available — and even if they are available it is probably faster to write them than it is to try to locate them.

"... it is up to us to provide our own technical support by sharing our knowledge ...."

One example is a mailing list which we purchased in DIF format and needed to convert to MailMerge® format. It didn't take more than about a half hour to write a BDS C program to do the job. I have pieces of code for handling input and output files plus other common tasks, which I read in with WordStar® and then modify. No, the code was not elegant, and it did not include good user interfacing, but the file was converted, and I may never use it again. I don't even know how long it took because it ran on another computer while I was doing something else. In this case it was much faster and easier to write the code than it would have been to search for something which might not even do exactly what I wanted. A couple hours after we received the DIF file on disk, we were using the data and I was doing something else.

We did not know what format the data would be in, so the first thing I did was to dump a few pages to the printer in HEX and ASCII — then it was easy to figure out what the C program needed to do. This brings up another programming need, which is the problem of determining file structures for custom modifications. The first thing I do with any new file type is a dump in order to see how it's put together. We use the Condor® database system and while it works well in most cases, there are some instances where we need something different. Condor can export the data to an ASCII file, but it is a bother always writing a file, so I just did a dump and then wrote a simple C program to extract (or even modify) the data in the main file.

Some of the routines I write for interfacing to our old Compugraphic typesetter are even more specialized and unlikely to be available elsewhere. The point of all this is that while many microcomputer users don't need to program, there are those of us who do need to program. I'll quote Hilton once more 'What's the use of having a computer if you can't make it do things YOUR way.'

You wouldn't be reading TCJ if you didn't want to be able to program, but is it because you HAVE to program or because you WANT to program? Let's run a little survey. Write and let us know if you could perform all your work or job functions without programming and you just program for fun, or if programming is a necessary part of the job.

Multi-Tasking or Multi-Systems?

While everyone else is pursuing multi-tasking and multi-user systems, I'm going the other way towards one user with multi-sytems. The cost of MS DOS clones and CP/M systems is so low (Xerox 820-II with 10 MB hard drive, CP/M 2.2, and WordStar, for $349 in BCE ad on page 459 of the August Computer Shopper) that it is cheaper and easier to use another system for time consuming tasks such as database sorts or printer drivers. Why buy a printer buffer when you can get a complete system with hard drive for $350? You can set up the extra system so that you can send the data over RS-232 at 19.2K baud (even between CP/M, 68000, and MS DOS systems, or you can move it on a disk).

Now if I just had a master operating system which would manage multi-systems instead of multi-tasking....

The Coming Bloodbath There have not been many announcements about layoffs or business closings during the past six months, but all is not well in the industry.

There are too many businesses trying to sell the same or equivalent products, and now that the demand is slacking off some of them are starting to hurt. We have talked to many suppliers who are cutting back on their advertising. Has anyone else noticed that the July Byte is 200 pages less than the 1983 issue? At the same time the August issue of The Computer Shopper has grown from 322 pages to 512 pages in the last year! This indicates a major change in buying habits as computers become a commidity. It is difficult to un-

derstand how all those advertisers in Computer Shopper are selling enough to pay for the ads, and if they are selling that much, how is that affecting the computer stores? Also, how long will the market support sales of that magnitude before the demand is satisfied?

I'm predicting that sales will decrease rapidly by October or November, and that the suppliers will try to hold out for the Christmas season, but that we'll see a lot of casulaties by February or March. It will be interesting to watch what happens in the computer magazine field as those businesses fail (remember that there is about a four month lead time for advertising). I've been plotting the page count for Byte since 1980, and Computer Shopper since August 1986, and would be interested in seeing a chart of the page counts for some of the other magazines for the past three or four years (if you keep your magazines, send the page counts for each month and year, and I'll publish a chart of the results). It is not impossible that we may see more computer magazines folding next year when their advertising revenue drops below an acceptable figure.

Again, any feedback or comments on this will be appreciated.

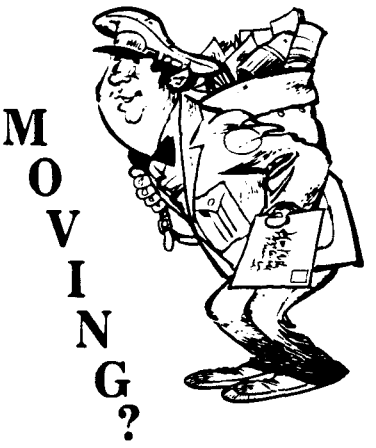
TCJ'S BBS — The Final Chapter

One thing that McCain didn't mention in his article "Starting Your Own BBS" in the last issue is making sure that you have access to a reliable phone line.

We are on a long rural line with such poor quality that we often have problems even with voice transmissions, and there are many times when we can not communicate at 300 baud much less 1200 baud. The final two weeks were so bad that we have finally had to discontinue the TCJ BBS because of transmission problems. While I blame most of the problem on our local lines, the quality is especially bad when the caller uses long distance services other than AT&T — even with voice calls I often have to have the caller redial using AT&T in order to be able to understand them. Perhaps this is because we are so far from a node for the alternative services.

I'll try to make arrangements to place the program listings on another board in an area with better communications, and until then I'll supply listings on disk (as far as we are from most people, a disk is probably about as cheap as a phone call).

■■



Make certain that TCJ follows you to your new address. Send both old and new address along with your expiration number that appears on your mailing label to:

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, MT 59912

If you move and don't notify us, TCJ is not responsible for copies you miss. Please allow six weeks notice. Thanks.

A book with
ADVANCED TOPICS in TURBO PASCAL

Turbo Pascal - Advanced Applications

Table of Contents

- Optimization Techniques
- Using the DOS Background Print Spooler
- System Level Tools
- Creating Libraries
- Exploiting Command Line Arguments
- Using a Binary Search Tree
- Techniques for Data Compression
- Claiming CP/M Memory
- Break the 64K Data Limit
- Linked Lists for Data Structuring
- Interrupts from Turbo Pascal
- Calling the DOS Command Processor
- Bit Mapped Graphics
- Teaching an Old Screen New Tricks
- Implementing 2D Core Graphics
- Build a Subset Pascal Compiler

Order ***Turbo Pascal - Advanced Applications*** for $19.95 postpaid in USA; with MS DOS disk, $32.95. Add $3.50 for surface shipment to Canada or other countries; air rates on request. Order from **Rockland Publishing, Inc.**, 190 Sullivan, Suite 103, Columbia Falls, MT. 59912. Visa or Mastercard accepted. Phone orders , call (406) 257-9119. **Free information is available.** Dealer inquiries welcomed.

"...packed with good advanced technical information."

NOT ANOTHER BEGINNER TUTORIAL

Turbo Pascal - Advanced Applications

Books of Interest

Advertiser's Index

AMPRO Computers 12
Anapro 4
Bersearch 17
C User's Group 11
C.C. Software 35
Computer Journal 42, 43
Echelon, Inc. 23, 30
Electronic Technical Services 20
Hawthorne Technology 33
Kenmore Computer Tech. 15
Micromint 41
MicroPro 21
N/Systems 4
North American 180 Group 20
Rockland Publishing 45
Sage Microsystems East 25
Turbo Power 47

The purpose of this section is to bring important reference resource material to your attention, and to help you determine which items should be added to your library. We will attempt to concentrate on the less well known and unusual titles, especially those not already in wide distribution.

We welcome your suggestions of titles which should be included, and also your requests for topics for which you can not find suitable reference material.

The 68000: Principles & Programming
by Leo J. Scanlon
Published by Howard W. Sams & Co.
1981, 5½ × 8½", 237 pages

This is a very useful book for someone switching over to the 680XX family, as it covers the background, chip hardware, and interfacing in addition to the instruction set and routines. Originally published in 1981, the copy I have is the 1986 seventh printing — a book which has been reprinted this many times must be selling well.

The contents are as follows:

- Chapter 1: An Introduction to the 68000 Microprocessor — Overview of the 68000; Internal Registers; Background on the Design of the 68000.
- Chapter 2: Cross Macro Assembler — Assembler Statements; Assembly Language Instructions; Stand Alone Comments; Assembler Directives; Expressions in the Operand Field; Conditional Assembly; Macros; Line Listing Format.
- Chapter 3: The 68000 Instruction Set — Instruction Format in Memory; Addressing Modes; Effective Addressing Mode Categories; Instruction Types; Data Movement Instructions; Integer Arithmetic Instructions; Locigal Instructions; Shift and Rotate Instructions; Bit Manipulation Instructions; Binary Coded Decimal (BCD) Instructions; Program Control Instructions; The Link and Unlink Instructions; System Control Instructions.
- Chapter 4: Mathematical Routines — Multiplication; Division; Division With Overflow; Square Root.
- Chapter 5: Lists and Look Up Tables — Unordered Lists; A Simple Sorting Technique; Ordered Lists; Look Up Tables; Jump Tables.
- Chapter 6: 68000 Microprocessor Chip Hardware — Clock, Power, and Ground Lines; The Data Bus and Address Bus; Function Code Signals; Asynchronous Control Signals; Synchronous Control Signals; Bus Arbitration Signals; System Control Signals; Interrupt Control Lines.
- Chapter 7: Processing States, Privilege States, and Execptions — Processing States; Privlege States; Exceptions; Internally Generated Exceptions; Externally Generated Exceptions.
- Chapter 8: Fundamentals of Interfacing — 68000 Support Chips; 6800 Support Chips; Interfacing a 6821 PIA to the 68000.
- Chapter 9: System Development — Motorola System Support Products; VME Bus; Other 68000 Related Products; 68000 Software Support.

The book provides a good overview of the 68000, with enough details to get you started. I like the fact that the chapter covering the instruction set is grouped by function with similar instructions together, instead of being discussed in alphabetical order. There are other reference books available with the alphabetical arrangement for use after you are familiar with the instructions. In addition to the chapter contents listed above, there are several appendices including Instruction Execution Times and a Summary of the 68000 Instruction Set.

(Continued from page 48)

prints, having all wires labeled and marked, as assembled on site prints (showing changes or additions made at the last minute), make sure it has a modular design (affords for easy replacement of parts), doesn't contain one of a kind bus system (uses VME, STD, or Multibus cards). The last kicker was the memory upgrade of $3000 per 64K of memory, all because the original memory cards were not designed correctly. Component wise that is $300 worth of memory going for $3000, not a bad mark up.

The point to consider here is watching your purchases for the near and long term. It may work great right now but what if it fails. The industries have been cutting back on using skilled personnel to service equipment, not supplying adequate documents to make your own servicing possible, and mostly expecting you to return the unit for repairs. This means you would need complete spares of every part used in the system, if you want to be up and running at all times. For GE systems that is $3000 (all boards cost about this) times 15, yes that is fifteen different board types. When I say this is a poor design, I am serious. The power supply alone is 4 different boards (that I know of), anyone of which can mess you up and can not be replaced with anything but the proper spare board. Most well designed systems, have a separate power supply, with a power monitoring circuit on the main CPU unit. Should the supply become flaky, the CPU sees it, shuts things down and you replace the unit with a spare or something like it. Down time on systems like that can be longer for me to get there than it is to fix them. Systems that don't have clear modular design can take longer to trouble shoot than repair.

I just recently bought a new car and before I did, I checked Consumers Report and compared what I was interested in to other comparable units. Unfortunately, there is not a consumers report for industrial controllers. We could also use one for regular computers as well. I have heard reports that IBM ATs are having hard disk failures as high as 60%. I don't know if this is true or not, but one just died the other day at work (less than 4 months use) and several friends have gone through two or three drives before getting one to last. I also had a course in a training lab with 10 XTs using 10Meg hard drives, half of which sounded like gum ball machines (you could hear every bearing). Now these are real problems, but there is not a publication that tells you about these problems, except us. Let me know if you have validating information about systems or designs which are particularly good or bad. Yup, that is "good" also, as I want to let you know what to buy as well.

This is enough for one month, next time I should have some more projects finished and be getting my new 68K system by then. I will be at the SOG again this year with Art. So till then, keep hacking. ■

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, Macintosh, 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.

TURBO PROGRAMMERS–

TDebug Plus™

...CUTS DEBUGGING FRUSTRATION.

TDebugPLUS is a ***new*** *interactive symbolic debugger that integrates with Turbo Pascal to let you:*

- ***Examine and change variables*** *at runtime using symbolic names – including records, pointers, arrays, and local variables;*
- ***Trace and set breakpoints*** *using procedure names or source statements;*
- ***View source code*** *while debugging;*
- ***Use Turbo Pascal editor and DOS DEBUG commands***

TDebugPLUS also includes a special MAP file generation mode fully compatible with external debuggers such as Periscope, Atron Symdeb, and others – even on programs written with Turbo EXTENDER.

An expanded, supported version of the acclaimed public domain program TDEBUG, the TDebugPLUS package includes one DSDD disk, complete source code, a reference card, and an 80-page printed manual. 256K of memory required. Simplify debugging! $60 COMPLETE.

TURBO EXTENDER™

Turbo EXTENDER provides you the following powerful tools to break the 64K barrier:

- ***Large Code Model*** *allows programs to use all 640K without overlays or chaining, while allowing you to convert existing programs with minimal effort; makes EXE files;*
- ***Make Facility*** *offers separate compilation eliminating the need for you to recompile unchanged modules;*
- ***Large Data Arrays*** *automatically manages data arrays up to 30 megabytes as well as any arrays in expanded memory (EMS);*
- ***Additional Turbo EXTENDER tools*** *include Overlay Analyst, Disk Cache, Pascal Encryptor, Shell File Generator, and File Browser.*

The Turbo EXTENDER package includes two DSDD disks, complete source code, and a 150-page printed manual. Order now! $85 COMPLETE.

TURBOPOWER UTILITIES™

"If you own Turbo Pascal, you should own TurboPower Programmers Utilities, that's all there is to it." Bruce Webster, ***BYTE Magazine***

TurboPower Utilities offers nine powerful programs: Program Structure Analyzer, Execution Timer, Execution Profiler, Pretty Printer, Command Repeater, Pattern Replacer, Difference Finder, File Finder, and Super Directory.

The TurboPower Utilities package includes three DSDD disks, reference card, and manual. $95 with source code; $55 executable only.

ORDER DIRECT TODAY!

- ***MC/VISA Call Toll Free*** *7 days a week.*
  800-538-8157 x830 (US)
  800-672-3470 x830 (CA)
- ***Limited Time Offer!*** *Buy two or more TurboPower products and save 15%!*
- ***Satisfaction Guaranteed*** *or your money back within 30 days.*

For Brochures, Dealer or other Information, PO, COD – call or write:

TURBO Power™

3109 Scotts Valley Dr., #122
Scotts Valley, CA 95066
(408) 438-8608
M-F 9AM-5PM PST

The above TurboPower products require Turbo Pascal 3.0 (standard, 8087, or BCD) and PC-DOS 2.X or 3.X, and run on the IBM PC/XT/AT and compatibles.

THE COMPUTER CORNER

A Column by Bill Kibler

The summer heat is here, and there is nothing to do about it except to work harder. I am nearing completion of my master's program and am starting to consider other job prospects. What I am facing is mostly an uphill battle however. Over the past year I have learned that the public, and management in particular, has a complete misconception of what is needed to teach computer skills. Some people equate it with learning to drive a car. I equate the skills needed more closely to those of learning to fly.

We take the skills we have learned using and servicing computers mostly for granted. Those of us who are technically inclined have little trouble handling the structure under which most computers operate. After usually a rather short period of time, we find we can master any new operations by relating it to other previous situations. The public at large unfortunately has none of these attributes.

My last two classes have had a number of school teachers included in them, and I have been able to see first hand what an educated person, who has never seen a computer before, encounters in trying to learn the operations. The first problem is they can't type, and lots have never used a typewriter before. I don't understand how these people ever got the credential, but I guess typing services are making more money than we think. These students get lost between DOS and program commands very easily. The concepts of structure and dividing programs into logical operations is a hard concept for them to understand.

The real battle for these people is with their schools. The schools expect these people, after one or two classes, to be able to teach other teachers and students. During the teaching they should also integrate and write some creative educational programs. Because we work with programs and computers we understand the fallacy of this concept, unfortunately the school managers know so little about computers, they can't understand why these programs are not working. As I see it, untrained managers in education appear to be a large part of the problem.

Managers in other fields can also be problems, especially if they were trained on mainframes. The manager of the computers services where I work has never used a microcomputer before. He told me once, he bought a C64 for his kids to play with, and that is his knowledge of micros. When it comes to training or purchasing of systems, this person in charge has little idea of the needs of users or skills needed to use such systems. Now that management has seen the productivity of micros, they are buying them and promoting programmers to teach their use. I feel strongly that these managers will also blame the computer company when these poorly trained people destroy a couple months of data with one key stroke.

I guess what I am getting at is the lack of appreciation for properly trained people in industry. I see this as one reason why the Japanese have been able to produce better products. They have valued training and skills very highly. We seem to have taken our education of late rather lightly. I know of lots of jobs where I work in which a large number of people have acquired their position by being related to the boss. Now don't get me wrong, I have nothing against properly qualified people that are related getting jobs over someone else who is not related. But what I have seen is people without skills and qualifications being given jobs because of relationships, or the cut of their clothes. It is my opinion that if we continue this practice it will not be long before countries other than Japan pass us up technically.

In the Mail...

On lighter subjects, I received the last copy of POOR MAN'S NETWORK, by Anderson Techno-Products. This is their version 2.00 and I was using version 1.0 before this. I haven't had a chance to really test the things out yet, other than seeing that it worked. One thing he didn't do was make it clear that the addresses used in the overlay package need to be changed from the version 1.0 equates. I tried using mine from the other version and they didn't work the first time. I changed the equates to the new addresses (10 hex higher, OFFSET now 0F00 not 0E00hex) and it came up without any other problems. I still have problems with his BIGBOARDB overlay as I am using a XEROX 820II and have to use some other code. I have to use a MVI A,10h and OUT PORTSTAT as part of each I/O (DR$ANYEXT and DR$EXTRDY) before the IN PORTSTAT. The ratebaud (0E not 0F) and stop bits (04 not 0Chex) were also changed.

The neat thing about this version is the overlays for the RPM-PC program. This is an CP/M emulation program for the PC compatibles. Using this overlay should allow you to run the network on the PC and transfer CP/M and PC files back and forth. I use a modem program now, but could see the advantage of using this on the PC. I tried it with the Z80MU program from PCBlue #185, but still haven't found all the changes needed yet. I also have the Atari ST and its CP/M emulation program to try and see if we can make it run on them. I am looking forward to later when I can spend more time wringing out the program. He has also changed the file handling a bit, but I will cover that later too.

The Saga Continues

I have had more rounds with the GE system at work lately, and hopefully have it fixed for now. We have spent over $30k just keeping this thing running. That is the cost of a new controller for the 5 AXIS lathe it is on. It really pays to check on replacement cost and technical support cost before one buys a system of any kind. The idea that these units will never break is just not true. After all this money we spent, we still don't have adequate spare parts, and there is going to be major problems when I leave later this year. The training on this system has really become specific, as it is just a one of a kind monster. I have heard however that all GE systems are much the same.

Some points to keep in mind when checking out other systems are: as built

(Continued on page 47)