The COMPUTER JOURNAL®

Programming - User Support Applications

Issue Number 36 | January / February 1989 | $3.00

Information Engineering
Introduction

Modula-2
A Bibliography

Temperature Measurement & Control
Computer Control for Agricultural Applications

ZCPR3 Corner
Z-Filer

Real Computing
National NS32032

SPRINT
A Product Review

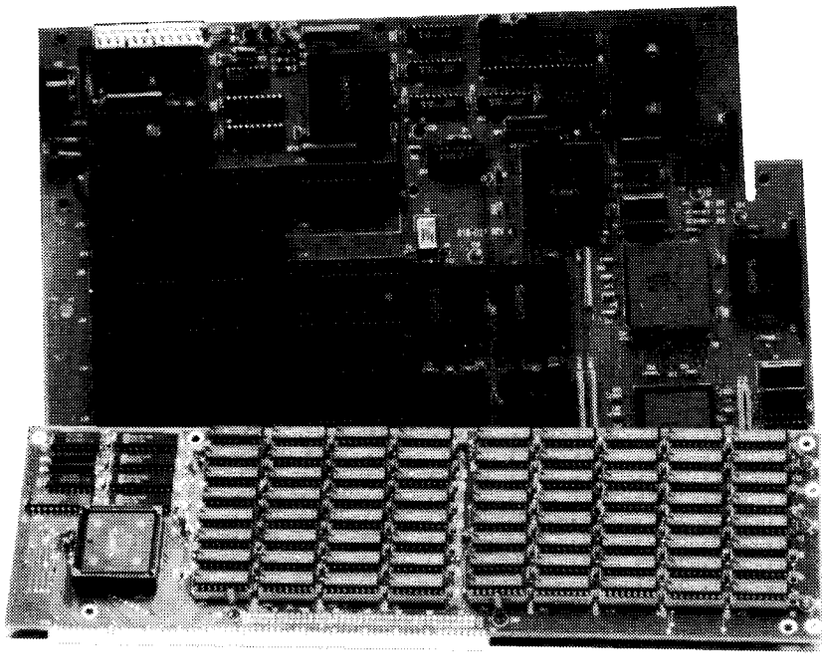
ZCPR3
Using Named Shell Variables

REL-Style Assembly Language for CP/M and Z-System
Part 2: Getting Started

Advanced CP/M
Environmental Programming

ISSN # 0748-9331

WORLD'S FASTEST 25 MHz 386 MOTHERBOARD



- Landmark's SPEED Test Rated at 43.5 MHz
- 1 or 2 MB Full Static RAM for Zero-Wait-States
- Optional 4/8/16 MB 32-bit DRAM Card
- Disk Cache and Print Spooling Software
- PC/XT (Baby AT) Form Factor
- Optional 80287, 80387 and Weitek
- LIM 4.0 and RAM Disk Utilities
- Made in U.S.A.
- 2-Year Parts & Labor Warranty
- No risk --100% money back guarantee

WAVE MATE INC.

2341 205th Street, #110,
Torrance, CA 90501

Tel: (213)533-8190 Fax: (213)533-5940
Technical Support: (800)876-5363

THE COMPUTER JOURNAL

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Circulation
Donna Carlson

Contributing Editors
Joe Bartel
C. Thomas Hilton
Bill Kibler
Bridger Mitchell
Bruce Morgan
Richard Rodman
Jay Sage
Barry Workman

The Computer Journal is published six times a year by Publishing Consultants, 190 Sullivan Crossroad, Columbia Falls, MT 59912 (406) 257-9119

***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.

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

The Lillipute Z-Node sysop has made his BBS systems available to the TCJ subscribers. Log in on both systems (312-649-1730 & 312-664-1730), and leave a message for SYSOP requesting TCJ access.

The COMPUTER JOURNAL

Features

Issue Number 36
January / February 1989

Columns

Editor's Page

Industry Watch

The advanced sales bookings for the semiconductor industry is an important indication of future computer sales, and the semi book-to-bill ratio has dropped from 1.17 in May to 0.92 in October. A ratio of less than 1.0 which means that their current shipments are more than the current new orders is very bad news for the industry.

Advanced Micro Devices has announced that it will lay off 1,000 of its U.S. staff out of a total of about 8,500. Entire projects, design groups, and manufacturing projects will be chopped. This follows an earlier layoff of 1,400 from AMD's Malaysian and Filipino assembly work force.

National Semiconductor is forecasting a significant operating loss for the current quarter due to sharp sales declines because of market softness. They have not announced layoffs, but that is the usual reaction.

Intel has announced that softening demand will enable Intel to satisfy the demands for the '386 for the first time.

A British semiconductor market research company, Semstat, states that the third quarter demand was below forecast, and that there are indications of a European industry slowdown.

These, and other reports, mean that the industry is in another of its roller coaster cycles— and we don't know where it will bottom out. Less semi sales means that the equipment builders are anticipating lower sales, which means that someone is going to be stuck with excess inventory, which means that you may see distress sales in the next 6 to 8 months. When semiconductor chips are in short supply, manufactures place inflated orders with multiple vendors in an attempt to assure themselves of a supply. When the manufactures sales drop, and the chips become readily available (often at a lower price), the manufactures cancel all their orders and pick up their needs from the spot market. This causes a snowballing effect which is disastrous for the semi industry.

Watch for the opportunity to acquire expansion RAM and memory intensive products at much lower products. Also be aware that marginal manufactures and resellers will be wiped out. I've heard that the PC demand has been met and that sales are dropping, but I have not seen the effects yet because of the long advertising lead times. Watch the page counts for *Computer Shopper* and the bloated PC magazines very carefully. Don't send checks to marginal resellers who may be gone before they ship your merchandise. Pay the extra charge for COD, or take other steps to assure that you will receive what you pay for!

Recalcitrant Vendors

In view of the tightening personal computer market, you would think that vendors should be anxious to make sales—but there are several major software houses which refuse to allow people to buy their products!

"Foreign companies are taking over because they are hungry and provide what the customer wants."

A person who is developing embedded controllers using 8 bit processors would like to be able to include copies of the language and development software for system maintenance. He contacted the vendors, but they said that the 8 bit software is obsolete and that they will not sell it. He offered to reproduce the software and manuals at his own expense, with a statement that support is no longer available from the vendors, and pay the vendors a license fee. They refused!

Apparently they feel that no one should use 8 bit systems—even for embedded controllers—and that they can't be bothered collecting license fees (with no effort) from obsolete products.

This is another example of how American industry is shooting it self in the foot. Foreign companies are taking over because they are hungry and will provide what the customers want to buy. American companies sit back fat and lazy while their 'experts' decide what they want the customer to use.

The success of the foreign countries is usually attributed to their low labor rates, but now they are building plants in the U.S. to build products using American labor! Their success is primarily due to good business planning and management's concern with what the buyer wants—they can win even with higher labor, material, and transportation costs. American management has squandered too much money on fancy corporate offices with too many layers of overpaid non-productive management, while ignoring the customer and the physical production facilities. Trade embargoes and high import duties will only force us to pay more for inferior products. The rest of the world will advance while we continue to produce outdated inferior products and lose what little export market we have left.

Foreign countries are progressing very rapidly in programming and software engineering, and we will lose our lead in these fields too unless we wake up and pay attention to what the customer wants to buy!

DTP vs. Wordprocessing vs. Typography

Tom Hilton and I get into some very interesting discussions about our writing tools. We have different needs, and different preferences, so we seldom end up in complete agreement. Lately, we have been talking about the differences between editors, wordprocessors, formatters, and Desk Top Publishing programs.

As a writer I spend a lot of time at the keyboard and I have definite ideas about what I want in an editor. We have contracted to enter a 75,000 word manuscript (about 450,000 characters), and have two more manuscripts waiting. That's about a million and a half characters to be entered in the next few months! With this amount of typing, I want a simple, friendly editor which stays out of the way and lets me do my job. There are several editors which would be satisfactory, but I am using WordStar 4.0 because I am familiar with it.

(Continued on page 26)

Reader's Feedback

I am including this note of praise for TCJ along with my renewal because I want to give you a pat on the back for your efforts as well as my check.

First off. Thank you for your generous coverage of CP/M systems. I almost believed the "CP/M is dead" cry of the masses until Jay Sage pointed me toward TCJ at the Trenton Computerfest this year. That's when I realized there is a strong undercurrent moving the 8 bit world along. This undercurrent seems to be propelled by the "Z" products and spearheaded by many dedicated individuals that are regulars in TCJ.

My complements to Bridger Mitchell, Bruce Morgan and Jay Sage among others for contributing so much to TCJ and the 8 bit community. I am constantly amazed at the depth of the material all of these authors cover. It is pretty impressive for a system supposedly stagnant. I am fully aware of the time and effort that must be expended to get such quality work and they have my sincere appreciation for that effort.

I have been very pleased with your coverage of other systems as well. This is well accomplished through Bill Kibler's Computer Corner and the many articles that appear (such as the SAGE 68000). You always manage just the right mix for my taste.

Since I brewed my S100 from bare boards and solder I find it nice that there is still a magazine that supports homebrewing and 'tinkineering' at the chip and bit level. With the advent of the cheap clones we have become a dying breed.

Finally, I like the format of TCJ, lots of substance without the gloss, glitz and glitter. Keep up the good work!

D.M.

I bought the minimum parts of a 10 MHz XT clone and had it running till I had to steal the floppies for a Kaypro IV I traded for an Atari 520. My 820-IIs and MD-3s and Dynabyte S-100 still doing fine.

I keep saying I will write for you, but it's almost all talk so far.

I expect to play with the XT and the ST as a user, and work on the CP/M system for hardware, programming, and word processing. I could be tempted with 68000 stuff though.

Keep up on CP/M, ZCPR, MR-OS, QPM, Bar Coding, S-100, I/Os for various machines, KOS, single boards (including the old BB1s).

I really need to sell some of my duplicate systems though, space is ever a problem and new projects languish without space.

Happy Forth, keep the faith.

W.M.

Enclosed find a check for a two year subscription to your fine journal. I have enjoyed reading it, even if a few of the articles are beyond me.

I am presently using a Heathkit H/89, purchased a year ago on something of a flyer for the sum of $200. I have added a pair of ½ height 360K drives, a 2400 baud Modem, and run the whole thing under Z-System. I am presently using the latest version of NZCOM from Alpha Microsystems. I have used the standard CP/M for a total of a couple of weeks. I began with the Z-System that Peter Shkabara of ANAPRO put out. I have fun amazing my MS-DOS friends with accounts of the features and capabilities of the system. Of course, the series of articles by Jay Sage in your journal have assisted greatly in the process.

Over the last year, I have acquired the set of software that I feel will do whatever I want for years to come. I got SuperCalc with the machine. I have purchased WordStar Release 4.0 for CP/M, Condor 3 for a data-base, and SMART, for a checkbook program for CP/M that works well. Almost forgot to mention OUTTHINK, a very good outline program.

Reason for using WordStar is simple: All of the utilities in Z-System use same control sequences, as does SuperCalc and Condor 3. OUTTHINK can be configured to use that same set. Result is not having to remember a new set of controls for each program. And, although I haven't used it yet, most of these programs will exchange certain kinds of files with each other. If I really wanted to get fancy, I am sure that using the ALIAS capabilities of NZCOM, one could pretty well automate most of such things.

I would really like to see some articles by someone who has put together his/her own Single Board Computer. I am planning to do so myself, when the finances allow. I am interested in an SB180FX, DT42, Grudge, Little Board, and any other Z80 or compatible equipment.

Otherwise, keep up the good work. This non-hacker does appreciate what you and all of the other bright folks do and share with us.

C.T.

I am using a British computer— an Acorn BBC (which is 6502 based) + a Z80 coprocessor which runs the ZSystem (NZCOM + ZCPR3.4). I have implemented BGii for this system and hope to also implement DOSDisk as the machines at work (a university) are all PC compatibles. My interest in TCJ is in the articles on ZSystem and CP/M primarily, i.e. Jay Sage and Bridger Mitchell. I use my system for word processing, programming in Pascal and Modula-2 (and occasionally to play Infocom games).

I am a researcher in Computer Science—my interests are in Object Oriented Programming and interchanging word processor documents using ODA (Office Document Architecture).

T.A. (England)

Editor's Note: Our readers would be interested in articles on interchanging word processor documents using ODA.

My own system is a SB180 which I use for 'hacking' around. At work I use '286 and '386 systems running MS-DOS and UNIX to develop device drivers for high resolution graphic cards (1024 by 768 and 1280 by 1024). These cards are based around AMD's QPDM, with a Brooktree palette and 2 Megs of 'Zig-Zag U-RAMs' on board.

I have also written other software running on 8080, Z80, and 32016, including VT52 terminal firmware, I/O processor firmware for a TEK 4108 emulator and code to drive Western Digital Floppy Disk controllers and NCR's 5380 SCSI controller. I'm therefore interested in all aspects of software and hardware available today.

The Computer Journal is a real 'gem' and I consider it a must for anyone interested in the ins and outs of 8 bit systems—which is where most of the articles lie. But why not? Eight bit computing's got a lot of life left in it yet!

K.P. (ENGLAND) ■

Information Engineering
Series Introduction
by C. Thomas Hilton

Information Engineering is a new science, especially to the world of microcomputers. This discipline is quite apart from dealing with a collection of raw information to achieve some practical end. It is important that this concept not to be confused with the world of data processing, as we have come to know it.

The tools used in Information Engineering, (IE) need not be as new as the discipline. The essence of the thought is in how the tool is used. There are a number of familiar products which lend themselves to the IE concept. There are also new products available, and being developed specifically for this facet of the "Information Age."

Tool Selection

Selecting the right tool can be a monumental task. Selecting the wrong tool can be as frustrating as it is expensive. The purpose of this series is to assist you in the selection and application of IE tools.

To these ends, we will review books directed at understanding the fundamental IE concepts. We will also review conventional, as well as new, software technologies which lend themselves to our purpose, and have been made available to us by their respective publishers for this series. To demonstrate fundamental IE concepts, these products will be applied to real world applications.

Real-World Application Projects

Many of the projects we will present, as examples, will be of special interest to educators, counselors, psychologists, and other professionals. It always seems that those who could most benefit from a series like this are seldom computer programmers. The projects we will present will avoid computer programming concepts whenever possible. This decision was made for several, additional, reasons. Conventional programming languages are just not up to the task. The purpose of this series is to demonstrate practical, usable concepts. These concepts can easily be lost in the depths of programming discussions. Finally, the tools we will be discussing are generally off-the-shelf products of such sophistication that they actually write their own error-free program source code, where programming is required.

While the projects presented in the series will be of specific interest to some, they will not be of interest to all readers. If you have a particular problem you would like presented in this series, feel free to join right in. These are public forums where your input, suggestions, and applications are considered important.

User Product Support

The basic tools we will be using are SPRINT® , QUATTRO® , PARADOX 2.0® , 1ST CLASS FUSION® , and the CLARION PROFESSIONAL DEVELOPER® . Other tools and products may be introduced or reviewed during the course of events, but these are the products I will personally be supporting.

While these products were selected for use in the IE series, by virtue of their universal nature, I have committed myself to moderating additional support columns for Borland & Clarion products, if reader interest dictates. So, if you have a problem, a tip, hint, or application note to share, do let us hear from you. As our publisher is fond of saying, "If you don't contribute anything, don't expect anything."

Looking for Knowledge Tools

The person reading this series is expected to be new to the concepts presented, or a data processing person looking for a new way to deal with old problems. The best way, I think, to deal with the many demands of "Information Engineering," (IE), as a whole concept, is to break the task into smaller pieces.

One half of the series will deal with concepts of information management, as we have come to think of it. Here we will take a look at the tools that can be used, the concepts important in information systems design, from a modern, nontraditional perspective, and new ways to get information from old data. This half of the series will also be broken down into smaller facets which deal with specific parts of the overall concept.

The primary tools will be:

1. QUATTRO, For Small Projects & Presentation Graphics.
2. Clarion, For Medium To Large Projects That Are Fully Defined.
3. PARADOX, For Projects That Deal With Large Amounts Of Information, And Information Management.

I have selected Borland products almost exclusively for use with this series. While the products I have received are evaluation copies, they were solicited. That is, I asked for these tools for use in this series. The reasoning is simple. SPRINT, QUATTRO, and PARADOX are from a single software publisher, hence their file types are compatible. Where there is a compatibility issue, the manuals show how to move data from one product to another. The ability to freely move data between products, without conversion errors, is critical in the IE field. I cannot recommend the use of SPRINT, QUATTRO, and PARADOX strongly enough for any person looking for good tools to work with.

For those with an investment in a favorite relational database manager, please note that you are not required to purchase PARADOX to participate in the series projects. Your database structure must be table oriented, however. Data tables may be moved between PARADOX and R:BASE, as an example, using the dBase III + import/export facilities common to both systems.

The primary use of QUATTRO will be the manipulation of smaller databases, or specific information parsed from PARADOX. QUATTRO, though a spreadsheet, also contains database management and presentation graphics facilities. These "extra" facilities are generally thought of as "options" in other spreadsheet packages. While you may be able to follow the concepts we present using the spreadsheet for Information Engineering, you will not be able to use the query & graphics features found only in QUATTRO.

At the end of each project a report must be prepared. If you feel that any word processor will produce a report, you have not seen SPRINT in action. SPRINT makes the hard work of sharing printed information easier than any other word processing system I have dealt with. In fact, I have switched from XyWrite III+® to SPRINT, which tells a grand tale in the world of professional writing.

You can follow the series without the tools I have mentioned. You will not be able to participate directly in the project unless you have the Borland products, or products which are compatible. The concepts are universal, and can be adapted easily. If you desire copies of the actual product data, scripts, or files, using the same tools as we use in the series is highly recommended.

Expert Systems & Development

The second facet of the series will deal with Expert Systems. This new world of IE goes beyond the simple, mechanical forms of dealing with raw data. Technically speaking, once our basic information has been collected, we have found an acceptable way of managing it, and have converted it into a logical format, we will then turn this information into knowledge.

Knowledge engineering is very much a part of the IE concept, but something different. I can recall many times feeling that, after completing an information system, "Now that I have all of this information, what can I really do with it?" Where the information system design project ended is where I wanted to start, not walk away.

The fundamental, mechanical, aspects of data management can be thought of as cold and lifeless. It differs little from collecting, storing, moving around, and sending out copies of little boxes. Knowledge engineering brings this collection of inanimate data to life, and allows us to communicate intelligently with it. This is where the fun can begin.

I tried to deal with data in the world of artificial intelligence, but, at the time, the tools available for microcomputers left a great deal to be desired. LISP required more effort than it returned value. PROLOG seemed to offer the most promise. Of the dialects I could afford, most lacked the power to do serious work. I have been looking at Borland's Version 2.0 Prolog with something more than lust. If it were supplied for evaluation, I might explore its real-world possibilities. Right now, however, I do not have the time to learn a new programming language. In fact, I really don't have a lot of time to do any programming at all.

The lack of time, interest, or ability to learn an exotic programming language is a common one. Where time is money, what can be found off-the-shelf, with a flat learning curve, is worth whatever price asked for it—if it will do the job.

I cannot stress enough that selecting the wrong tool for the job can be expensive and frustrating. The purpose of this series is to assist you in selecting the proper tools without the expense and frustration of error. If we do nothing else but convince you of this fact, I will feel I have accomplished something important.

While I am willing to go as far as I can in helping you, I cannot possibly cover all facets of the subject matter. The theoretical concepts you will require will be presented in a project oriented way. But, this will only be a small part the of IE and Knowledge Engineering worlds.

There are many ways you can prepare yourself for this series. One of the foremost is to read some books on the subject. Two books worthy of mention are discussed below. They will fill in any areas that we may overlook in the task oriented approach to our subject matter.

Expert Systems: Artificial Intelligence in Business
Harmon & King
John Wiley & Sons General Books Division
ISBN 0 471-80824-5 $18.95 Retail

Expert Systems: Tools & Applications
Harmon, Maus, & Morrissey
John Wiley & Sons General Books Division
ISBN 0 171-83950-7 $22.95 Retail

- What are Expert Systems?
- What are they supposed to do?
- What kinds of programming languages will have to be acquired & learned?
- How many programmers will I need?
- Have others dealt with problems similar to mine?
- If so, what are the costs of their solutions, and who do I contact for assistance?

All of the questions above are valid, and very important to YOU, as an Information & Knowledge Engineer. If any one of these questions have come to mind in reading this introductory column, then you need these books. It is simple as that. There is no question or debate on the subject.

The first Harmon book of interest, *Artificial Intelligence in Business* was released in 1984. In the way the industry travels along at light speed, this book could be considered slightly historical. But, if your interest is in understanding, then we have another case altogether.

Instead of dealing with the program code of some lost and forgotten language, Harmon has dealt with the fundamental concepts of what is now being called Information Engineering. He does call upon the products of the time for use in his examples, but these are systems which are still available today.

Harmon's approach to his subject renders his information timeless. The concepts are as accurate and applicable today as they were in 1984. I could think of few fundamental questions that Harmon did not answer. The how and why of his subject matter are presented clearly. Examples & graphics illustrate and support the textual material tastefully.

As with any conceptual exploration of a complex subject, *Artificial Intelligence in Business* would not be considered light reading. A definite desire to understand the subject matter will be required.

The age of the work is its only defect, but only in one area which could not have been anticipated. Harmon relies heavily upon Rule based technologies, which were dominate at the time. If the reader takes this into consideration, then a complete understanding of Rule based systems will be had. Rule based technologies are very much in existence, though few are suitable for microcomputer applications.

All aspects having been considered, *Artificial Intelligence in Business* should be purchased along with Harmond's latest book, *Expert Systems, Tools & Applications*. The two book set gos together extremely well.

In *Tools & Applications* Harmond recognized the shortcomings of his first book, shortcomings caused by the passage of time. The second book continues on where the first book left off, as an update of the original information. The skill in which this task was accomplished was well accepted.

The concepts of Rule based systems are updated, and the concepts of Inductive Tools are introduced. While Rule based systems still dominate the minicomputer and mainframe worlds, Harmond discusses the role of each tool as it is found in commercial software offerings. Having explained the new material, he completes this volume with an introduction of hybrid products, which combine the best of both Rule based & Inductive Reasoning tools into a single product.

In both of Harmond's books commercial tools and final applications are presented in a classic, "what they do, who has them, and what they cost" format.

Of particular interest in *Tools & Applications* are lengthy discussions concerning "1st CLASS," and "1st CLASS FUSION," Inductive Tools we will be using in our series.

Subjective Commentary

Here is the way I personally view these two books. I was presented with a copy of "1st Class FUSION" to be used in this series. Having been presented with the tool, a discourse on other tools was really not high on my priority list. But, I had to read the books as part of the project. In thinking of the material in the FUSION manuals, I often got bored, or fell asleep reading the about seemingly unrelated concepts. This is no reflection upon the books themselves! I just wasn't all that interested in anything but the FUSION product. Then, when the second book started discussing the tool that was driving me to distraction, the "world moved." All of a sudden everything started making sense! Some of the material was hard to grasp, but only because I had not paid attention in my earlier readings.

What had begun as task became a glorious adventure! This is what I meant when I said you have to have an interest in the material. Once I had grasped what Harmond was telling me about FUSION, as well as interesting tidbits about the mind that devised the product, everything he had to say was interesting, from the perspective of how other systems related to FUSION.

I personally was not all that interested in Artificial Intelligence, having found it lacking for real-world applications. I wanted something that would help me in my daily tasks dealing with information. What Artificial Intelligence is, holds little interest to me, personally. What it will do for me is well covered in Harmond's presentations. But promises are promises. I still need something more concrete, or so I thought.

If you are going to follow our series, then you really should get BOTH of these books. Here is why. You will find the discussion in *Tools & Applications* that are of extreme interest because they deal with FUSION, the product we will be using in the series. The discussions about FUSION are, in themselves, reason enough to buy the book. Once you have read the areas that deal with FUSION, which are interlaced throughout the text, you will have fits of meaningless curiosity that will drive you nuts, unless you are one of those souls who can put such thoughts out of your mind. Most computer people cannot do that. So, you need the first book to provide the background information that will satisfy your curiosity.

You should be able to follow the parts of the series dealing with Expert Systems without having these two books. I could have written the series without them. The difference in understanding these books will provide you is the difference between cake, and cake with frosting. Having read the two books will make everything a whole lot sweeter at the end of the meal.

Well, I am about out of space for this issue. In the issues which follow we will present an interesting project dealing with "Deficit Identification & Reduction," as it would be applied to the analysis of SAT (Stanford Achievement Test) scores. As we begin that project, I hope to hear from some of you educators out there. It should be an interesting first installment. Once again, in closing, let us hear your thoughts.

The products mentioned are available from:

Sprint, Quattro, and Paradox
Borland International
Call (800) 543-7543 for the dealer nearest you.

1st Class Fusion
1st Class Expert Systems
286 Boston Post Road
Wayland, MA 01778
(508) 358-7722

Clarion Professional Developer
Clarion Software
150 E. Sample Road
Pompano Beach, FL 33064
(800) 354-5444 ■

Modula-2
A Bibliography
by Dave Moore, Alex Pournelle, and Barry Workman

This month's topic is "by popular demand": we're forever hearing the question, "Are there any good books on Modula-2?" The short answer is yes, there are; the 'where', though, is easier than the 'what'. We raided the Computer Literacy bookstore in San Jose, California, for many of these titles; that's one source. Most technical bookstores, and some chains, will have these titles. And if they don't, they can be ordered (be sure to bring the ISBN along).

The bibliography is probably less current than it was two months ago. Books on Modula-2 are rapidly appearing, and most are quite readable. Modula-2 is being discovered (at last) by more and more people. When we released our first compiler, there were a grand total of two books available. This list contains twenty-six titles. We know of at least four more titles that are to appear but which are not contained in the following list.

If you have any other titles to recommend, drop us a note. We'd appreciate as much information as in our other listings—it makes ordering easier. If you have comments on them, too, they would be welcome.

There are ongoing attempts to completely standardize Modula-2, especially by the BSI (British Standards Institute). When it indeed becomes standard, Workman & Associates will conform. At the moment, we (and all the other vendors, too) are waiting for the shoe to drop, rather than conforming to the interim recommendations.

One of the best ways to keep up with progress in the language is to join MODUS, the Modula-2 User's Society. MODUS publishes *The MODUS Quarterly* which includes the very latest developments in the language. MODUS also has semiregular meetings. Membership is US$20 or SFR45. Write to George Symons, P.O. Box 51778, Palo Alto, CA 94303 or Heinz Waldburger, Postfach 289, CH-8025 Zuerich, Switzerland.

The facts in this list are as good as we can make them but we deny any responsibility for accuracy. The capsule reviews are of course subjective. We urge you to examine books for yourself before buying.

Book Reference Convention

The ISBN (International Standard Book Number) of each book, where available, is the American English edition number. Then follows a capsule review, where available, and a rating of 'Beginner', 'Intermediate', 'Advanced Intermediate' or 'Expert'.

Book Sophistication Level

'Beginner' means someone just starting to learn programming or someone migrating to Modula-2 as their second language, without much structured programming experience.

'Intermediate' means familiar with system programming and at least two computer languages, or having passed a comparative programming course.

'Advanced Intermediate' means having written a fairly large application in one or more languages, feels comfortable reading EBNFs, criticizes language texts for unclear examples, etc.

'Expert' means one of those semimortals who dreams in EBNF, draws DFAs as graffiti, knows three or more languages. Experts will probably learn Modula-2 from (a) our manual or (b) Wirth's Programming in Modula-2, or (c) their own textbook!

BIBLIOTHECA — Modula-2

Programming in Modula-2, Niklaus Wirth. Springer-Verlag, New York, 1982, 1983, 1985. Third Corrected Edition , 1985, 202pp. ISBN 0-387-15078-1.

This is the de facto 'standard' for Modula-2, as much as one exists. There are rumors of a fourth edition about to be released, but we know of no one who has actually seen it. FTL Modula-2 conforms to the third, corrected edition; most older compilers conform to the second. This book is not recommended for beginners, light reading or learning the language. It is, however, the standard. Workman & Associates has this book for sale if you cannot find it.
Advanced.

Modula-2 for Pascal Programmers, Richard Gleaves. Springer-Verlag, New York, 1984, 145pp. ISBN 0-387-96051-1.

This is a very concise book for anyone who knows Pascal well. The examples are good as a quick reference when you're converting Pascal code to Modula-2. Richard Gleaves was part of the team that put created Volition Systems Modula-2, the first Modula-2 compiler for microcomputers.
Intermediate.

Modula-2 A Seafarer's Guide and Shipyard Manual, alias Modula-2 A Seafarer's Manual & Shipyard Guide, Edward J. Joyce. Addison-Wesley, Reading, MA, 1985, 270pp.

Comprehensive and certainly not dry. The first title is given on the cover, the second on the frontispiece. The first 5 chapters are designed for those learning Modula-2 right after BASIC; then it explores the language in depth. I think it's terminally cute, but after dull textbooks, it's certainly refreshing. The examples are good.
Beginning to Intermediate.

Modula-2--Problem Solving and Programming Style, W.C. Jones, Jr. Harper & Row, New York, 1987, 580pp. ISBN 0-06-043469-4.
Intermediate.

The next three books have exactly the same title—beware!

Modula-2 Programming, I. Kaplan and M. Miller. Hayden, 1987, 228pp. ISBN 0-8104-6480-2.
Unreviewed.

Modula-2 Programming, E. Kepley and R. Platt. Reston Publications, Virginia, 1987, 390pp. ISBN 0-8359-4602-9.
Unreviewed.

Modula-2 Programming, J.W. Ogilvie. McGraw-Hill, New York, 1987, 304pp. ISBN 0-07-047770-1.
Unreviewed.

An Introduction to Computer Science with Modula-2, J. M. Adams, Philippe J. Gabrini, Barry L. Kurtz. D. C. Heath and Company, Lexington, MA, 1988, 592pp. ISBN 0-669-12171-1.

A collegiate-level textbook with very nice examples and emphasis on classroom use. Early coverage of procedures and modules. Not just a rewrite of a Pascal book; designed as a first course. The D.C. Heath Company is now offering this text to colleges and universities with or without the FTL Modula-2 compiler.
Beginner.

Modula-2: A Complete Guide, K. N. King. D. C. Heath and Company, Lexington, MA, 1988, 640pp. ISBN 0-668-11091-4.

A Modula-2 reference with comparisons to Pascal, real-world examples and an emphasis on advanced features. Dr. King teaches at Georgia State University, is on the ISO Modula-2 committee and knows his Modula.
Advanced Intermediate.

A First Course in Computer Science With Modula-2, L. Pinson, R.F. Sincovec, R. S. Weiner. John Wiley & Sons (Wiley Interscience), New York, 1987, 491pp. ISBN 0-471-81692-2.

A textbook with comprehensive examples, designed for classroom use.
Beginner.

Algorithms & Data Structures, Niklaus Wirth. Prentice-Hall, New York, 1987, 288pp. ISBN 0-13-022005-101.

See next reference. As with its predecessor, Wirth uses very short identifiers and doesn't present examples well. Still, it's better written and uses Modula-2 for examples.
Advanced Intermediate.

Data Structures Using Modula-2, R.F. Sincovec, R.S. Weiner. John Wiley & Sons (Wiley Interscience), New York, 1987, 500pp. ISBN 0-471-81489-X.

Niklaus Wirth wrote Algorithms + Data Structures = Programs after he invented Pascal; he wrote Algorithms & Data Structures after refining Modula-2. This book is in the same vein—a coursebook and guide to better, structured programming.
Advanced Intermediate.

An Introduction to Programming With Modula-2, P.D. Terry. Addison-Wesley, Reading MA, 1987, 460pp. ISBN 0-201-17438-3.

Clearly written with good examples. Terry teaches Modula-2 at Rhodes University; he's also on the ISO Modula-2 standards committee. He knows his Modula. This is a text more than a self-learning book.
Beginner.

Modula-2--An Introduction, D. Thalmann. Springer-Verlag, New York, 1987, 292pp. ISBN 0-387-13297-X.

Modula-2--Programming With Data Structures, B.K. Walker. Wadsworth, 1987. ISBN 0-534-05917-1.

Advanced Programming Techniques in Modula-2, T.A. Ward. Scott Forresman & Co., New York, 1987, 293pp. ISBN 0-673-18615-6.
Advanced Intermediate to Expert.

Modula-2, J. Beidler, P. Jackowitz. PWS Publishers, Boston, MA, 1987, 347pp. ISBN 0-87150-912-1.
Intermediate.

Modula-2 Primer, S. Kelly Bootle. Howard W. Sams, Indianapolis, IN, 1986, 453pp. ISBN 0-672-22560-3.

Bootle also wrote The Devil's DP Dictionary, a hilarious satire on data processing. He has written a nice textbook, with clear examples and humor. Useful as a self-teaching book. He contrasts Modula-2 with other structured languages, too. Doesn't start into the language as early as Joyce (above). This book uses Logitech's Modula-2 and is a little dated in that the compiler was second edition rather than third.
Beginner through Intermediate.

Introduction to Modula-2, P. M. Chirlain. Matrix Publishers, Beaverton, OR, 1987, 248pp. ISBN 0-916460-41-X.

Introduction to the language.
Beginner.

A Guide to Modula-2, K. Christian. Springer-Verlag, New York, 1987, 436pp. ISBN 0-387-96242-5.

A guide and reference rather than a beginner's book.
Intermediate to Expert.

A Software Development Approach, G.A. Ford & R.S. Weiner. John Wiley & Sons, New York, 1987, 404pp.
Advanced Intermediate to Expert.

Software Engineering and Modula-2, G. Pomberger. Prentice-Hall, New York, 1987, 264pp.
A discussion of software engineering coupled with Modula-2, not a Modula-2 textbook.
Advanced Intermediate to Expert.

Using Modula-2, D.D. Riley. Boyd & Fraser, Boston, MA, 1987, 641pp. ISBN 0-87855-236-8.

Unreviewed, but certainly comprehensive (look at its length).

Modula-2 Discipline and Design, A. Sale. Addison-Wesley, Reading, MA, 1987, 452pp. ISBN 0-201-12921-3.

Intended for use as the supporting text for a course in Modula-2.
Intermediate.

Programming Expert Systems in Modula-2, B. Sawyer, D. Foster. J. Wiley and Sons, New York, 1987, 201pp. ISBN 0-471-85036-5.

A book on a specialized topic rather than a general textbook.

Advanced Modula-2, H. Schildt. Osborne/McGraw-Hill, Berkeley, CA and New York, 1987, 379pp. ISBN 0-07-881245-3.
Advanced Intermediate to Expert.

Modula-2 Wizard, R.S. Weiner. John Wiley and Sons, New York, 1987, 209pp. ISBN 0-471-84853-0.

Companion to Pascal Wizard, also by the incredibly prolific Weiner (5 books in this list). Designed as a reference to those who already know the language. More readable than Wirth.
Advanced Intermediate and especially Expert.

Software Engineering with Modula-2 and Ada, R. Weiner and R. Sincovec. John Wiley & Sons, New York, 1987, 451pp. ISBN 0-471-89014-6.
Advanced Intermediate and Expert. ■

you see here.

THE AMPRO LITTLE BOARD™/286

Big power for smaller systems.
Little Board/286 is the newest member of our family of MS-DOS compatible Single Board Systems. It gives you the power of an AT in the cubic inches of a half height 5 1/4" disk drive. It requires no backplane. It's a complete AT-compatible system that's functionally equivalent to the 5-board system above. But, in less than 6% of the volume. It runs all AT software. And its low-power requirement means high reliability and great performance in harsh environments.

Ideal for embedded & dedicated applications. The low power and tiny form factor of Little Board/286 are perfect for embedded microcomputer applications: data acquisition, controllers, portable instruments, telecommunications, diskless workstations, POS terminals . . . virtually anywhere that small size and complete AT hardware and software compatibility are an advantage.

Compare features.
Both systems offer:

- 8 or 12MHz versions
- 512K or 1Mbyte on-board DRAM
- 80287 math co-processor option
- Full set of AT-compatible controllers
- 2 RS232C ports
- Parallel printer port
- Floppy disk controller
- EGA/CGA/Hercules/MDA video options
- AT-compatible bus expansion
- A wide range of expansion options
- IBM-compatible Award ROM BIOS

But only Little Board/286 offers:

- 5.75" x 8" form factor
- EGA/CGA/Hercules/MDA on a daughterboard with no increase in volume
- SCSI bus support for a wide variety of devices: Hard disk to bubble drives
- On-board 1Kbit serial EPROM. 512 bits available for OEMs
- Two byte-wide sockets for EPROM/RAM/NOVRAM expansion (usable as on-board solid-state disk)
- Single voltage operation (+5 VDC only)
- Less than 10W power consumption
- 0-70°C operating range

*AT is a Registered Trademark of IBM Corp.

Better answers for OEMs.
Little Board/286 is not only a smaller answer, it's a better answer . . . offering the packaging flexibility, reliability, low power consumption and I/O capabilities OEMs need . . . at a very attractive price. And like all Ampro Little Board products, Little Board/286 is available through representatives nationwide, and worldwide. For more information and the name of your nearest Rep, call us today at the number below. Or, write for Ampro Little Board/286 product literature.

408-734-2800
Fax: 408-734-2939 TLX: 4940302

AMPRO
COMPUTERS, INCORPORATED
1130 Mountain View/Alviso Road
Sunnyvale, CA 94089

Reps: **Australia**-61 3 720-3298; **Belgium**-32 87 46.90.12; **Canada**-(604) 438-0028; **Denmark**-45 3 66 20 20; **Finland**-358 0 585-322; **France**-331 4502-1800; **Germany, West**-49 89 611-6151; **Israel**-972-3 49-16-95; **Italy**-39 6 811-9406; **Japan**-81 3 257-2630; **Spain**-34 3 204-2099; **Sweden**-46 88 55-00-65; **Switzerland**-41 1 740-41-05; **United Kingdom**-44 2 964-35511; **USA**, contact AMPRO.

Temperature Measurement and Control

Computer Control for Agricultural Applications

by Matthew K. Rogoyski

Abstract

A simple, inexpensive temperature measuring interface for an IBM personal computer has been developed for use by orchardists. The conversion of analog to digital signal has been accomplished with the use of an IBM Game Control Adapter card connected directly to the computer bus. Two systems, the first utilizing the thermistor and the second a voltage output transducer, are discussed. The program for data acquisition and storing temperature with allied information is written in IBM PC BASIC. Another program, written in the C language, that stores transducer resistance data is also presented. Hardware and software can be readily expanded for use with other sensors. Concurrent usage of the MS-DOS personal computer for data acquisition and other tasks is explored. The role of the above interface in computer-aided orchard management is discussed.

Introduction

Producers of agricultural and horticultural commodities in Colorado are gradually adopting the powerful technology offered by personal computers. Present usage of these computers by orchardists is mostly limited to spreadsheet applications, accounting, and word processing (13, 22, 30). Another potential agricultural application of personal computers, explored in this paper, is the logging of weather data. A temperature monitoring system for the use by orchardists has been designed around the IBM PC Analog Input card, also known as the Game Control Adapter (7, 16, 18, 19, 25, 27). Interfacing of two types of temperature transducers: thermistor and voltage output temperature transducer is examined. Estimated cost of the system based on a IBM Analog Input card is less than $100 (US) including the card, electronic parts, and wire.

Many Colorado fruit growers are already manually recording daily minimum and maximum temperatures, as numerous mathematical models of biological processes that predict insect emergence, disease incidence, and occurrence of tree growth stages as a function of temperature, are available (11, 22, 28). The intermountain climate of fruit growing areas in Colorado is characterized by large microclimatic differences even between closely located sites (31). These differences are biologically significant. The closer the temperature transducer to the orchard, the more biologically valid data can be. Assuming that the majority of orchardists will own a personal computer in the near future, the weather data logging system based on these computers would considerably facilitate design of computer-aided orchard management systems.

The first system (thermistor based) was developed to record temperature readings in the 10° to 31° C range. This range corresponds to the minimum and maximum threshold of a degree-day model for the codling moth (11, 28), generally accepted as the number one fruit pest in Colorado orchards. The second system, based on a voltage output temperature transducer, has been designed to serve as an alarm for the upper temperature range encountered in the field. Usage of this circuit as an alarm is of interest in Colorado as heat stress related disorders are encountered in the state (31). The voltage output transducer, used in the alarm system, has been utilized in several temperature measuring systems described recently (14, 29).

The programming of the Game Control Adapter card is relatively simple as direct access to input ports is accomplished with the BASIC function, in the case of Microsoft IBM PC BASIC, called STICK (6). This greatly simplifies programming and eliminates need for costly commercial data acquisition software packages. The programming in C, on the other hand, requires definition of user functions. The major factor in simplicity of software and hardware design is the direct placement of the Game Control Adapter card in the computer bus; it eliminates potential complexity of interfacing data loggers, especially through an interface such as RS 232 (12). The resolution for this particular card is 256 discrete points, when software written in BASIC is used. This is equivalent to a conventional 8 bit analog to digital converter. The transducer resistance range, measured with aid of an Analog Input card, was expanded by software written in the C programming language.

Design of Interface and Testing Procedures

An IBM PC XT computer equipped with 256 KBytes RAM, 10 MBytes hard disk and one 360 KBytes floppy disk, and PC DOS version 2.00 (5) was used in the experiments. The program was written in Microsoft IBM PC BASIC (version A2.00). A similar program was also written in C and compiled with Ecosoft Eco-C-88 compiler (version 3.21). The data presented in this paper was obtained

Matthew Rogoyski works as a researcher at Rogers Mesa Research Center located in Hotchkiss, Colorado. He is assistant professor of pomology at Colorado State University. He graduated from University of Guelph with B.Sc. and received his Ph.D. degree from Cornell University. His work includes, among others, the development of various decision support systems for the fruit industry. His interests are in the area of computer interfacing and applied artificial intelligence. Correspondence can be directed to 3058 Hwy.92, Hotchkiss, CO 81419.

Funding was provided by Colorado Agricultural Experiment Station: (Project #157)

with the aid of a program written in BASIC. The analog signal from the temperature sensor was digitized by IBM's Analog Input half size card (7) (Table 1). The functional block diagram of the two systems tested is presented in Figure 1.

The temperature transducer of the first system was composed of two thermistors (type RL-D1) (Table 1). Two thermistors were connected in series (Figure 2) and mounted in the center of a perforated, unpainted aluminum box (10 cm × 8 cm × 4 cm). The transducer of the second system was the voltage output precision temperature sensor LM35CZ (8) (Table 1). A small heat sink (Table 1) was placed directly over the case of the LM35CZ transducer. The important component of the circuit needed for interfacing this transducer was a hermetically sealed cell combining a light emitting diode and a photo-conductive element (2,27) (Table 1 and Figure 3).

The sensors were placed inside the weather shelter (35 cm × 60 cm × 65 cm) about 135 cm above soil level. The shelter was located directly in the orchard. Approximately 90 m of twin-lead type cable was used (20-ga copper conductor twin-lead 300 Ohms impedance, with polyethylene foam insulation and weather sealed jacket). The wire was buried in 25 m vented PVC pipe under the road, the remaining wire was suspended aerially in a tree canopy. Partial protection from lightning for both systems was provided by the overvoltage, overcurrent circuit (Figure 2) (4, 15).

Transducers were calibrated and tested against a Polycorder Model #516 equipped with 32 K RAM and General Purpose Interface Board with six copper-constant thermocouples. The Polycorder itself was calibrated and programmed as described in Omnidata application note (3). Using data pairs sampled at the same instant of time, a model equation was derived. Thus, the transducer resistance value, R , was considered the independent variable and the corresponding temperature value, T, obtained with aid of Polycorder, was defined to be the dependent variable. Using Forsyth's method of generating orthogonal polynomials (24),a final second degree equation,

$$T = 0.00062\ R^2 - 0.3043\ R + 44.525$$

was generated for the system based on the thermistor. A similar second order equation,

$$T = 0.00031\ R^2 - 0.1143\ R + 47.894$$

was generated for the system utilizing a voltage output temperature transducer. Experiments were conducted at a later date, using Polycorder. The temperature values obtained with aid of Polycorder were compared with temperatures

Table 1 Sources for electronic components and parts.

Part	Manufacturer	Part Number	Source
Game Control Adapter	IBM	150 1 00	IBM
Thermistor RL-D1	Keystone Carbon Company	KC019N-ND	Digi-Key Corporation P.O.Box 677 Thief River Falls, MN 56701
Voltage Output Temperature Sensor	National Semiconductor	LM35CZ	Hamilton/ Avnet Electronics 068765 Orchard Road Englewood, CO 80111
Voltage to resistance convertor	Clairex Electronics	CLM 8000	Clairex Electronics 560 South Third Ave. Mount Vernon, NY 10550
Heat Sink	Avid Eng. Inc.	HS 100	Digi-Key
Signal Transmission Wire	Archer	15-1175	Radio Shack
Transient/surge adsorbers	Panasonic	ZNR K22056 ZNR 20K201U S53 ZNR 10K820 47	Digi-Key
Gas Discharge Tube		DSAR1-441LA	Radio Shack
Spark Gap Assembly	Custom Made		

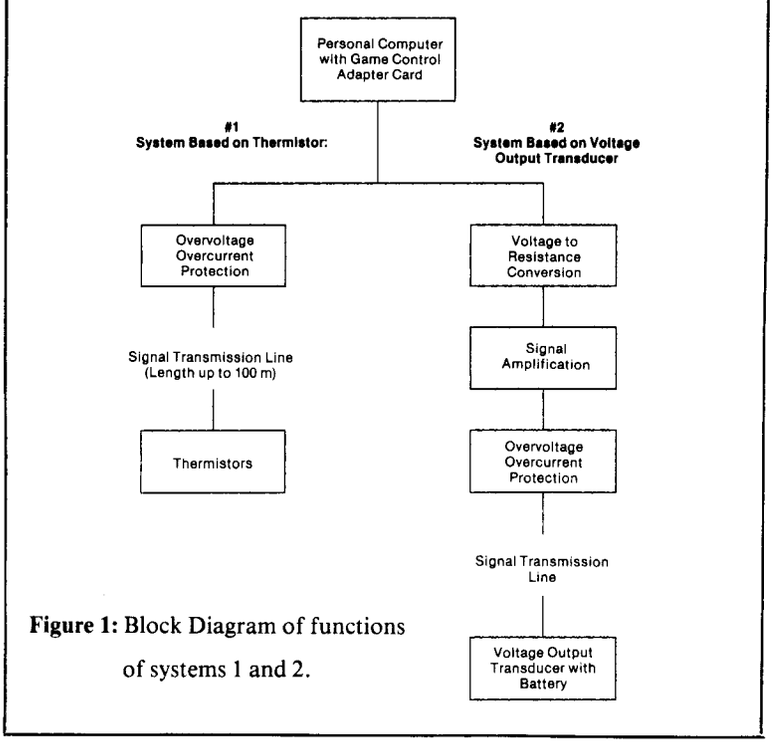


Figure 1: Block Diagram of functions of systems 1 and 2.

Figure 2: Circuit for thermistor sensor, overvoltage and overcurrent protection.

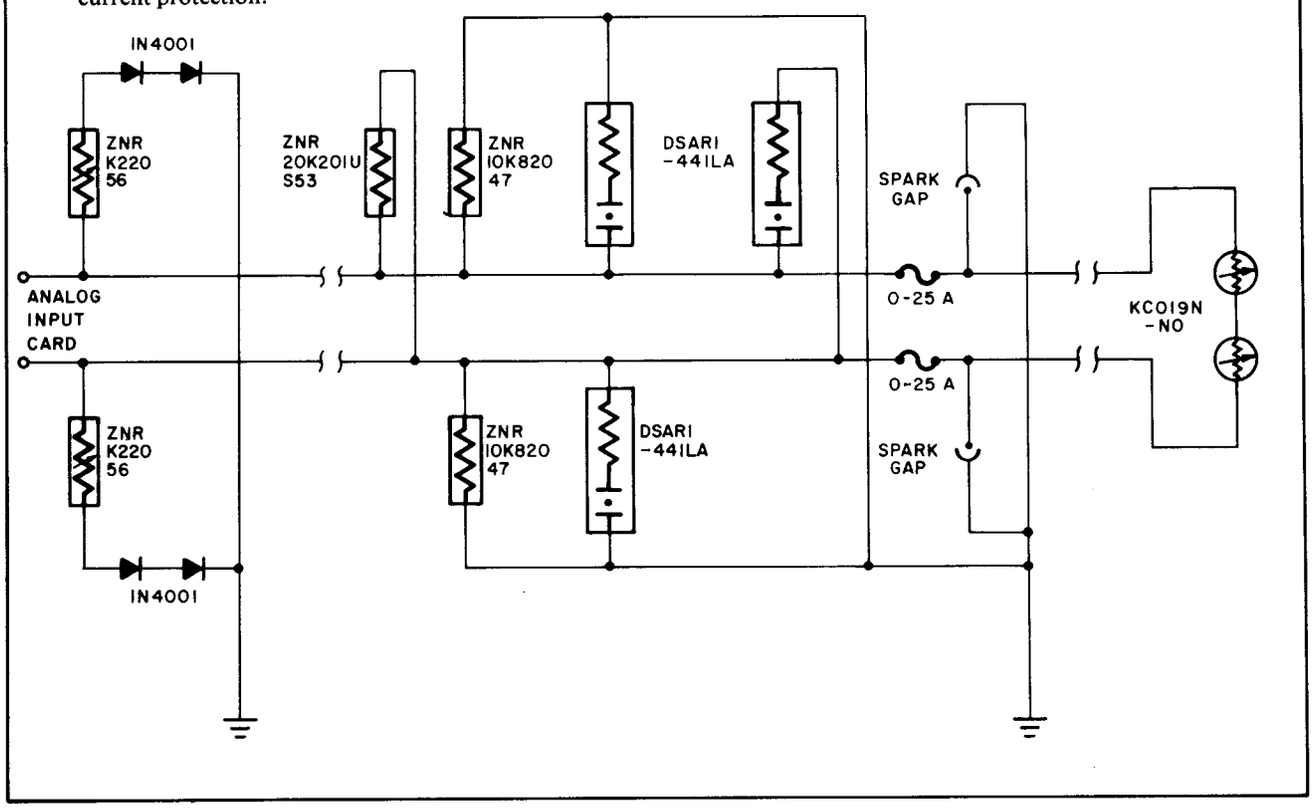
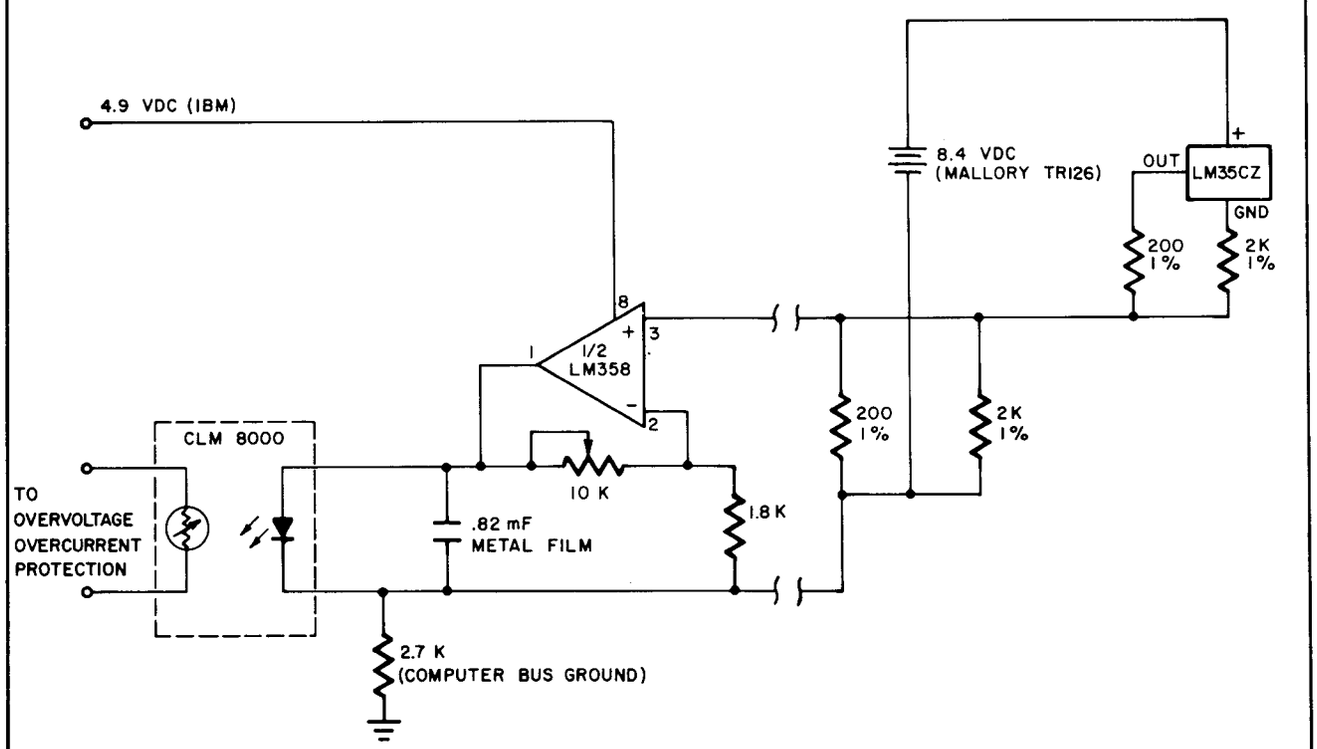


Figure 3: Circuit for voltage output temperature sensor.



calculated using the above model equation and the recorded transducer outputs. Statistical analysis on the above data was conducted as outlined by Daniel (17).

The thermistor based system was tested and calibrated under normal temperature fluctuation in the orchard. Performance of the system at the upper part of the temperature scale (20° C to 31° C) was evaluated by placing a forced air heater in the weather shelter which was covered with blankets to decrease temperature fluctuations. The thermistor sensor was located 1 cm to 3 cm from the Polycorder's thermocouples. In the case of the second transducer, the voltage output temperature sensor, its performance was also evaluated in the covered weather shelter equipped with a forced air heater. In this case, the thermocouples were taped with a electrical tape directly around the heat sink of the transducer.

Results and Discussion

The Game Control Adapter based system is an inexpensive alternative to dedicated data loggers and automated weather centers. The average correlation coefficient, for data from the thermistor based system was 0.96, indicating that the model was adequate (Table 2). The correlation coefficient for data generated by the system based on the voltage output transducer was 0.81 (Table 2). The precision was adequate for the circuits tested as indicated by the standard deviation (Table 2).

Table 2: Statistical results for curve fitting of experimental data for thermistor and voltage output transducer based systems.

System Transducer	Transducer Temperature Range	Experiment Number	Correlation Coefficient	F-value[1]	Standard Deviation[2] °C
Thermistor	10-23 °C	1	0.99	1097**	0.52
		2	0.97	639**	0.65
		3	0.87	132*	0.78
		Average	0.94		0.65
	20-30 °C	4	0.98	1048**	0.49
		5	0.98	959**	0.58
		6	0.98	877**	0.76
		Average	0.98		0.61
Voltage Output Transducer	37-47 °C	7	0.65	85*	2.1
		8	0.82	65*	1.9
		9	0.97	338**	0.8
		Average	0.81		1.6

[1]Significance level for F-value: ** 5% * 10%
(The F-value was calculated from the partitioned total sum of squares as outlined by Daniel (17).)

[2]This is estimated standard deviation, defined by (17) as the square root of the residual mean square.

The practical range of the Analog Input card under test was 3 to 254 KOhm, when the software written in BASIC was used; the C based software expanded the upper range of resistance that could be sampled by this card to values above 1 MOhm. The output of the BASIC program for every data point consisted of the date and the time when the reading was taken, the actual resistance value read by the card, and the calculated temperature value. The accumulated samples of data were forced to be stored on the disk every hour on the hour to prevent loss of data in event of power failure. It is important to note that a rapid change in input resistance, resulting in out of range value, caused a so called static condition to occur; i.e., a constant reading was returned each time the card was sampled. Because of this potential problem, the BASIC program tested for this condition and did not store these values.

The type of cable, used for a signal transmission line, had major impact on the absolute values returned by the card. A twisted pair of copper conductors coated with polyvinyl chloride (PVC) proved to be totally unacceptable resulting in excessive fluctuations of readings. Noise was reduced to manageable levels when polyethylene-coated conductors were used. It is postulated that PVC coating might be a cause of the excessive noise as the dielectric constant of PVC is 2.8 versus 2.2 for polyethylene (10).

The thermistor proved to be a well suited transducer for the Game Control Adapter card. This type of sensor had similar characteristics to the potentiometer for which this card was designed. No signal conditioning was necessary other than protection from overvoltage and overcurrent. The thermistor based system was not overly sensitive to sources of electromagnetic radiation encountered in the orchard environment, such as ignition systems of gasoline powered engines. The self heating of the thermistor was unlikely, as very low current was involved (less than 10 mA). Under the field conditions the thermistor needed to be painted with non-conductive coating, as the possibility existed that condensation water will short the electrodes attached to the semiconductor material of the sensor element.

The second system based on the voltage output temperature transducer was not directly compared with the thermistor based system. The complexity of the circuit was considerably higher than that of the thermistor based circuit, though manipulation of the signal could be readily accomplished because of an operational amplifier. The LM35CZ temperature transducer was linearly related to the temperature change (8); this linearity was lost when the signal was modified by the circuit for Game Control Adapter card input.

The MS-DOS operating system is not a multitasking system and the need exists for a computer to be used for tasks other than weather data logging. The simplest solution to this problem was to manually interrupt, for example, an execution of spreadsheet program, run the data acquisition program, and come back to the original task. This arrangement was not convenient, but it was workable. A number of commercial software packages, that simplify rapid manual switching between different programs, is available on the market for MS DOS (20). Some (e.g., Double DOS, Soft Logic Solutions, Manchester, N.H.) claim to allow concurrent use of programs and multitasking on some MS DOS computers. The Double DOS package, though not extensively tested, has allowed the apparent concurrent use of the above mentioned BASIC program and a word processor.

Acknowledgments

The author wishes to thank Dr. H. Larsen for fruitful discussions, Mr. L. Coffin for expert advice and assistance with curve fitting, statistical analyses, and programming, and Mr. J. Tembrock for technical assistance.

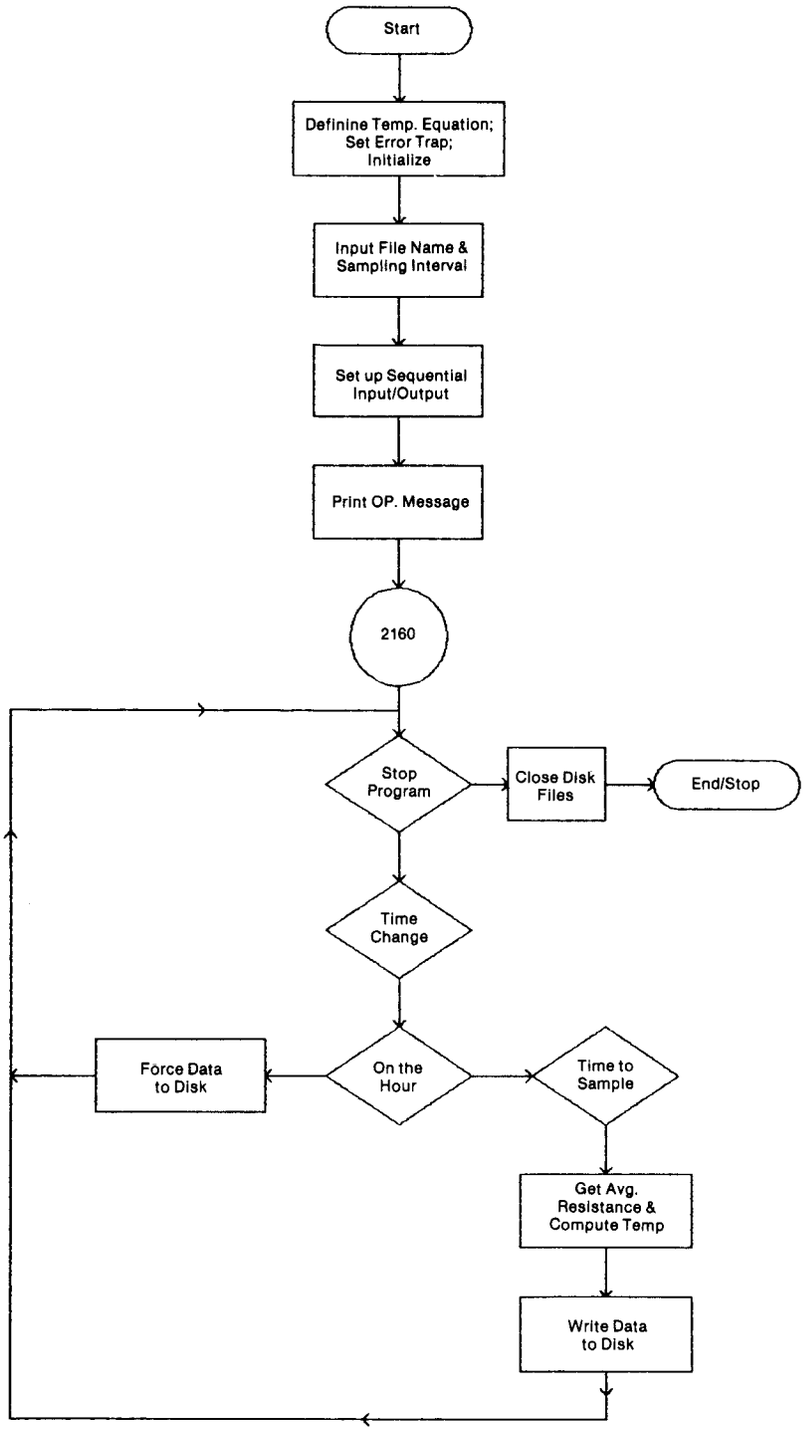


Figure 4: Flowchart for program to collect resistance vaules, calculate and store temperature values on disk.

★★★ *The diagrams for Figures 1 and 4 were prepared with Generic CADD® (Generic Software, Inc., 11911 North Creek Parkway South, Bothell, WA 98011 (800) 228-3601, output on a 9 pin Epson MX-80, and reduced to 75%. The text was set separately and pasted in.*

References

1. Allocca, J. A., A. Stuart, 1984. *Transducers: Theory and Application*, Prentice-Hall Co., Reston, Virginia, USA, pp. 497.

2. Anonymous, 1985. "CLM6000, CLM8000: LED - Photoconductor Isolators." *Optoelectronics Designers Handbook*, pp. 8.21-8.22, Clairex Electronics Mount Vernon, NY 10550, USA.

3. Anonymous, 1985. "Using the Polycorder Electronic Notebook to Measure Temperature Using Thermistors, TRD's and Thermocouples." *Application Note 2419/2460 Version 1.0*, Omnidata International, 750 West 200 North, Logan, Utah 84321, USA, pp. 182.

4. Anonymous, 1985. "ZapNot Technical Description," *ZapNot Technical Bulletin 85-1*, Omnidata International, Inc., Logan, Utah, USA.

m5. Anonymous, 1983. *Disc Operating System.* Version 2.00 by Microsoft Corp IBM part number 6024061.

6. Anonymous, 1983. *BASIC.* Version 2.0 by Microsoft Corp., IBM part number 6025010.

7. Anonymous, 1983. *Technical Reference Personal Computer XT.* Version 2.02 IBM Part Number 6936808, pages I203-I208 and D76.

8. Anonymous, 1982. "LM35/ LM35A, LM35C/ LM35CA, LM35D: Precision Centigrade Temperature Sensors." *Linear Data Book.* PP. 9.2-9.5. National Semi-conductor Corporation, 2900 Semiconductor Drive, Santa Clara, CA 95051.

9. Anonymous, 1982. "LM158/ LM258/ LM358, LM158A/ LM258A/ LM358A, LM2904: Low Power Dual Operational Amplifiers." *Linear Data Book.* pp. 3.216-3.219. National Semiconductor Corporation, 2900 Semiconductor Drive, Santa Clara, CA 95051.

10. Bruins, P. F. 1968. *Plastics for Electrical Insulation.* Interscience Publishers, New York, N.Y., USA, pp. 201.

11. Brunner, J. F., F. C. Hoyt and M. A. Wright. 1985. "Codling Moth Control." *Extension Bulletin #1072,* Cooperative Extension, Washington State University, Pullman, WA, USA.

12. Campbell, J. 1984. *The RS-232 Solution,* Sybex Inc., Berkeley, CA, USA.

13. Cardiff, J. 1985. *Farming and the Computer,* Group Four Publication, Inc. Seattle, WA, USA, pp. 231.

14. Carr, J. J. 1986. "Using an LM-335 Voltage-mode Temperature Sensor" *Modern Electronics,* (2):55-61.

15. Ciarcia, S. 1983. "Keep Power-line Pollution Out of Your Computer," *Byte* 8:36-44.

BASIC PROGRAM LISTING

```
10 '  ------------------------------------------------------------------------
60 '  -                                                                      -
110 ' -                                                                      -
160 ' -  X335.bas - a program to sample resistance values (using the        -
210 ' -  stick function), compute a corresponding temperature value         -
260 ' -  (using a prefitted curve) and to save these values, along          -
310 ' -  with time and date, on a sequential disk file.                     -
360 ' -                                                                      -
410 ' -  Inputs:         File name (F$) for data storage and the            -
460 ' -                  sampling interval (INTV).                          -
510 ' -  Output:         File records consisting of date, time,             -
560 ' -                  resistance value and temperature.                  -
610 ' -                                                                      -
660 ' -  Comments:       Program written in BASICA on an IBM XT PC.         -
710 ' -                  Prefitted curve function may be changed            -
760 ' -                  easily by modifying the  DEF function              -
810 ' -                  statement at the beginning of the program.         -
860 ' -                                                                      -
910 ' -                                                                      -
960 ' -                                                                      -
1210 '------------------------------------------------------------------------
1260 '
1310 '
1360 '
1410 ON ERROR GOTO 2810  'set error trapping for I/O
1460 DEF FNRE(RX)=.00062*RX^2 -.3043*RX  +44.525
1510 DEFINT I
1560 LAST$= ''a:tempx''   'set for later I/O
1610 CLS: KEY OFF
1660 INPUT ''type in data file name''; F$
1710 F$= ''a:'' +F$
1760 PRINT: PRINT
1810 INPUT ''Type in sampling time interval in minutes '';INTV
1820 CLS
1860 CURRENT$ = F$   'set file name for I/O
1910 OPEN ''I'',1,F$  ' see if file exists - if error will trap at 2810
1960 GOSUB 10360  'it exists - go set up for appending
2010 ILAST=99    'initialize variable for timing loop
2060 IHR =99   'init for date routine
2110 GOSUB 3210  'print operator message
2160 GOSUB 6560  ' see if user wants to stop
2210 GOSUB 3760  'get sampling time to show a multiple of input interval.
2260 GOSUB 4660  'obtain resistance value (avg.)
2310 WRITE#2,DATE$,T$,RE,TEMP        'save values to disk file
2360 GOTO 2160    ' loop to get next value for recording
2410 '
2460 '
2510 '--------------- End of Main Logic ---------------------------------
2560 '
2610 '
2710 '
2760 ' error trapping
2810 IF ERR=53 THEN OPEN ''c'',2,F$ : RESUME 2010
2860 ON ERROR GOTO 0  'error not known, turn off trapping
2910 '
2960 '------------------------------------------------------------------------
3010 '
3060 '
3110 '  subroutine to print operator message
3160 '
3210 LOCATE 10,5
3260 PRINT ''DO NOT TURN OFF MACHINE - TEMP PGM IS RUNNING'' : PRINT
3310 LOCATE 12,8
3360 PRINT ''TO END DATA COLLECTION, TYPE IN AN   E'' : PRINT
3410 RETURN
3460 '
3510 '------------------------------------------------------------------------
3560 '
3610 '
3660 '  subroutine to check for sampling time
3710 '
3760 T$=TIME$
3810 GOSUB 6560  ' see if user wants to stop
3860 LOCATE 14,55
3910 PRINT T$
3960 I=INSTR(T$,'':'')  'return position of first colon
4010 A$=MID$(T$,I+1,2)  'extract minute string
4060 IN=VAL(A$)   'convert minutes to numeric value
4110 IF IN=ILAST THEN GOTO 3760  'minutes have not changed since last time
4160 ILAST = IN
4200 ' when in=0, on the hour is indicated and data is forced to disk.
4210 IF IN=0 THEN GOSUB 4660: GOSUB 11110: GOSUB 10360: RETURN 2160
4260 IF (IN MOD INTV)=0 THEN RETURN  'it's ready for sampling
4310 GOTO 3760  ' get next time value
4360 '
4410 '
4460 '------------------------------------------------------------------------
4510 '
4560 '
4610 '
4660 '  subroutine to get resistance data and compute temperature
4710 '
4760 '
4810 TIMER ON
4860 ON TIMER(1) GOSUB 5110   'warm up electronics
4910 X=STICK(0)
4960 FOR J = 1 TO 30  ' delay a little
5010 NEXT J
5060 GOTO 4910
5110 TIMER OFF
5160 RETURN 5210
5210 GOSUB 7710 : CLS : GOSUB 3210
5260 T$ = TIME$
5310 R=0
5360 FOR J = 1 TO 100
5410    X=STICK(0)
5460    Y=STICK(1)
5510    R= R+Y
5560            FOR I=1 TO 10   ' delay
5610            NEXT I
5660    LOCATE 14,55  :  PRINT T$
5710 NEXT J
```

```
5760 RA = R/100
5810 RE= CINT(RA)
5860 TEMP= FNRE(RE)
5910 TEMP =CINT(TEMP)
5960 PRINT ''Time =  '';T$ : PRINT
6010 PRINT ''re = '';RE : PRINT
6060 PRINT ''Temperature =  '';TEMP;'' 'C''
6110 LOCATE 21,40  :  PRINT ''File name ''F$
6160 LOCATE 8,55  :  PRINT  DATE$
6210 GOSUB 3210
6260 RETURN
6310 '
6360 '
6410 '-------------------------------------------------------------------
6460 '
6510 '
6560 '  subroutine to see if user wants to end - signified by pressing
6610 '   either the E  or e key.
6660 '
6710 B$=INKEY$  'check to see if user wants to end
6760 LOCATE 8,55  :  PRINT  DATE$
6810 IF B$='''' THEN RETURN
6860 IF B$=''e'' OR B$=''E'' THEN GOSUB 7210
6910 RETURN
6960 '
7010 '
7060 '-------------------------------------------------------------------
7110 '
7160 '
7210 '         subroutine to close files and end
7260 CLOSE
7310 IF CURRENT$ = F$ THEN CLS: END
7360 KILL F$
7410 NAME CURRENT$ AS  F$
7460  CLS: END
7510 '
7560 '
7610 '-------------------------------------------------------------------
7660 '
7710 '         subroutine to test for out of range resistance values
7760 '          or in static condition
7810 '
7860 IOUT=0    'init out of range count
7910 ICONST=0
7960 V= STICK(0) : W= STICK(1)
8010 IF W < 11 OR W > 254 THEN GOSUB 9360 : GOTO 7960  'get new value
8060 IL = W    'save last value
8110 IC = 0    ' init change counter
8160 FOR I = 1 TO 100
8210   V=STICK(0) : W = STICK(1)
8260   IF W<>IL THEN IC = IC+ 1
8310                  FOR Q=1 TO 10   ' delay
8360                  NEXT Q
8410 NEXT I
8460 IF IC <> 0 THEN RETURN  '      go back to avg.
8510 GOSUB 9910  ' print const values
8560 GOSUB 8660       'print time and wait
8610 GOTO 7960        'get new value
8660 '
8710 '
8760 '-------------------------------------------------------------------
8810 '
8815 '
8820 '  subroutine to print time and wait
8825 '
8860 FOR J = 1 TO 3000  ' timer loop approx 3 sec
8910  LOCATE 14,55  :  PRINT TIME$
8960   GOSUB 6560
9010 NEXT J
9060 GOSUB 6560        ' see if user wants to end
9110 RETURN
9160 '
9210 '-------------------------------------------------------------------
9260 '
9310 '
9360 '         subroutine to print out of range data
9410 '
9460 IOUT = IOUT + 1
9510 IF IOUT >=2 THEN LOCATE 18,40: PRINT ''out of range  '';IOUT;'' times''
9560 LOCATE 17,40  :  PRINT ''RE value =  '';W
9610 GOSUB 8660  ' wait 3 seconds
9660 RETURN
9710 '
9760 '-------------------------------------------------------------------
9810 '
9860 '
9910 '         subroutine to print constant values
9960 '
10010 ICONST = ICONST+1
10060 IF ICONST>=2 THEN LOCATE 19,40  :  PRINT ''Constant value '';ICONST;'' times''
10110 LOCATE 20,40  :  PRINT ''RE value =  '';W
10160 RETURN
10210 '
10260 '-------------------------------------------------------------------
10310 '
10360 '         subroutine to force data to disk on the hour
10410 '
10460 CLOSE
10510 OPEN ''o'',2,LAST$
10560 OPEN ''i'',1,CURRENT$
10610 WHILE NOT EOF(1)
10660   INPUT #1,D$,T$,R,TX
10710   WRITE #2,D$,T$,R,TX
10760 WEND
10810 CLOSE 1   'leave output file open
10860 SWAP CURRENT$, LAST$
10910 RETURN
10960 '
11010 '-------------------------------------------------------------------
11060 '
11110 '         subroutine to save data to disk file
11160 '
11210  WRITE#2,DATE$,T$,RE,TEMP
11260  RETURN
11310 '
11360 '-------------------------------------------------------------------
11400 '                 end of x335.bas
```

C Program Listing

```
/*************************** thrm.c ************************************/

                                        /***  v.  5-29-87   ***/

/*****************************************************************************
                        Source code for:
                    DATA ACQUISITION PROGRAM
                FOR IBM PC XT'S ANALOG INPUT CARD
        Program Description:
        1. This program samples the resistance of transducer,
           such as thermistor, and returns its resistance
           value in KOhms.
        2. The sampling interval is set to be 3 minutes.
        3. The structure is used to store the results: resistance,
           corresponding time and date.
        4. The stored data is written to a binary file on disc after
           each sampling event.
        5. The source code was compiled with the Eco-C-88 C compiler
           (version 3.21).
        6. The user can stop the data acquisition by pressing
           any key from keyboard.
        7. Description of the interface for the thermistor output:
           ( thermistor wired to the analog channel #2 )
                a. first - the port at the address hex 201 (decimal 513)
                   is activated by sending value ''0'' using outportb()
                   fuction from the ecoc library.
                b. second - then the value returned by the inportb() function
                   (from port hex 201) is examined to determine how long it
                   remains hex ff.  A counter (cnt) is used to detrmine this
                   time duration.
                c. the value returned from the port hex 201 will change to
                   hex fd after a time duration proportional to
                   the resistance of the transducer.

*****************************************************************************/

#include  <stdio.h>
#include  <fcntl.h>

#define NEWLINE   printf(''\n\n\n'');
#define SPORT   513     /** Analog input card port */
#define KPORT   96      /** Keyboard scan port */
#define MAXSCANS   10000  /** Maximum scans allowed */
#define MAXSAMPLES   3   /** Maximum samples per one scan */
#define VALUE   0       /** Value to be sent to port 513 */
#define CONSTANT   6.30   /** Used to calculate resistance */

struct date {
        unsigned int   month;
        unsigned int   day;
        unsigned int   year;
};

struct time {
        unsigned int   hour;
        unsigned int   minute;
        unsigned int   second;
        unsigned int   hundred;
};

struct result {
        float   KOhm;               /** Thermistor resistance */
        unsigned int   yr;
        unsigned int   mh;
        unsigned int   dy;
        unsigned int   hr;
        unsigned int   min;
        unsigned int   sec;
};

main()
{
     int   fx, i = 0, rcnt;
     float   resistance;
     char   *gets(), f_name[20];
     struct date   d, *d_ptr;
     struct time   t, *t_ptr;
     struct result   r, *r_ptr;

        /** Set up a binary file for storing resistance, date,and time data */
        printf(''\n\t Enter file name \n'');
        gets(f_name);
        fx = open(f_name,O_RDWR|O_CREAT|O_EXCL,0); /** Test if a binary file*/
                                                   /** with the same name exists  */
        if(fx == -1) {
             printf(''\n Unable to create a file '');
             exit(-1);
        }
        r_ptr = &r;
        while(i < 1) {
                stop_pgm_test(f_name,r_ptr);  /** See if the user wants */
                                              /**  to stop.             */
                for(i = 0; i < MAXSCANS; ++i) {
                      getdate(&d);
                      gettime(&t);
                      r_ptr -> yr = d.year;     /** Move date and time   */
                      r_ptr -> mh = d.month;    /**   to result structure*/
                      r_ptr -> dy = d.day;
                      r_ptr -> hr = t.hour;
                      r_ptr -> min = t.minute;
                      r_ptr -> sec = t.second;
                      rcnt = get_avg_resistance();  /** Thermistor output*/
                      resistance = (float)rcnt * CONSTANT;
                                                    /** R units are KOhms */
                      r_ptr -> KOhm = resistance;
                      store_on_disc(f_name, r_ptr);  /** Write to disc */
                      scr_message(r_ptr);  /** Display results and msgs*/
                      while(( t.minute % 3 ) == 0){  /** Wait one minute */
                                gettime(&t);
                                stop_pgm_test(f_name,r_ptr);
```

```
                    }
                    trigger(f_name,r_ptr);  /** See when to start  */
                                            /**  sampling again    */
                }       /** End of a loop for maximum scans allowed */
        }      /** End of a while(1) loop  */
}

/************************** End of main  ***********************************/

/************** The termination of program execution function *************/

/** The first argument passed to this function is the name of binary file.
    The function tests for scan code of hex 80, indicating a key realese.
    The second argument passed to this function points to the result struct.*/

stop_pgm_test(namebf,m_ptr)

char   namebf[20];
struct result   *m_ptr;

{
    int   i;
        if(inportb(KPORT) == 0x80) {  /** To stop the program press any key*/
                store_on_disc(namebf,m_ptr);  /** Store the data on disc */
                for(i = 0; i < 12; ++i)   /** Clear screen */
                        NEWLINE
                exit(0);            /** Exit to DOS */
        }
        return;
}

/************************ end of stop function ****************************/

/******************  Screen message management function *******************/
scr_message(m_ptr)

struct result   *m_ptr;

{
    int   i;
        for(i = 0; i < 3; ++i)          /** Clear a part of the screen */
              NEWLINE
        printf(''\t Data Acquisition in Progress - Do Not Turn Off Computer'');
        NEWLINE
        printf(''\t\tTo Stop Program Execution,  Press Any Key Several Times'');
        NEWLINE
        printf(''\t Today's Date Is  %d - %d - %d  '',
                        m_ptr -> mh, m_ptr -> dy, m_ptr -> yr);
        NEWLINE
        printf(''\t\t\t Sampling Interval Is  3  minutes '');
        NEWLINE
        printf(''\t Last Sampling Time Was  %d : %d : %d '',
                      m_ptr -> hr, m_ptr -> min, m_ptr -> sec );
        NEWLINE
        printf(''\t The Average Thermistor Resistance Was   %6.1f  KOhm '',
                              m_ptr -> KOhm );
        NEWLINE
}

/************************  Sampling trigger function  **********************/

/** The first argument passed to this function is the name of the binary
    file to be used by stop_pgm_test() called within this function.
    The second rgument points to the result structure */

trigger(namebf,m_ptr)

char   namebf[20];
struct result   *m_ptr;

{
    int   i;
    struct time   t;
        t.minute = 1;
        while(( t.minute % 3 ) != 0 ) {   /** Delay for 3 minutes */
                gettime(&t);
                stop_pgm_test(namebf,m_ptr);   /** See if the user wants */
                                               /**      to stop */
        }
        return;
}

/******************** end of sampling trigger function  ********************/

/******  Get average resistance value of thermistor sensor function *******/
get_avg_resistance()

{
    int   i, j, cnt;
    int   sum = 0, average;
        for(i = 1; i <= MAXSAMPLES; ++i) {
             cnt = 0;
             outportb(SPORT,VALUE);  /** Activate port 513 */
             while(inportb(SPORT) == 0xff) {   /** Test how long the port  */
                    ++cnt;                     /**    is active            */
                    if(cnt > 1000)     /** Stop when counter over 1000 */
                          return(1000);
             }
             if(cnt == 0) {
                 printf(''\n Port cannot be activated or R too low '');
                 return(0);
             }
             sum += cnt;
             average = sum / i;
             for(j = 0; j < 33; ++j)  /** Short delay needed to stabilize */
                  ;                   /**   electronic circuits           */
        }
        return(average);
}

/********************** end of sampling function  ************************/
```

```
/********************* Store data on the disc function ********************/

/** The first argument of this function is the name of the binary file.
    The second argument is a pointer to the structure containing experimental
    results from the single evaluation of the thermistor's resistance.
    This pointer points to output buffer for the write function. */

store_on_disc(namebf, k_ptr)

char   namebf[20];
struct result   *k_ptr;

{
    int   fx, c_er, w_er, number_bytes;
        fx = open(namebf,O_RDWR|O_APPEND,0); /** Open binary file for append*/
        number_bytes = sizeof(unsigned)*6 + sizeof(float);
        w_er = write(fx, k_ptr, number_bytes); /** Put results to binary file*/
        if(w_er == -1) {
             printf(''\n Error writing to file '');
             exit(-1);
        }
        c_er = close(fx);    /** Close binary file */
        if(c_er == -1) {
             printf(''\ Error closing to file '');
             exit(-1);
        }
}

/******************** end of storage function ****************************/
```


16.Ciarcia, S. 1979. "Joystick Interfaces," *Byte* 4:10-18.

17.Daniel, C. and Wood F. S., 1971. *Fitting Equations to Data,* Wiley-Interscience, New York, USA, pp. 342.

18.DeJong, M. L., 1985. "Gamepaddle Control Linearity Test," *Byte* 10: 161-162.

19.Eggebrecht, L. C., 1983. *Interfacing to the IBM Personal Computer,* H. W. Sams Co., Inc., Indianapolis, Indiana, USA, pp. 246.

20.Fullerton, P. 1985. *IBM PC Expansion and Software Guide,* Que Corporation, Indianapolis, Indiana, USA, pp. 900.

21.Hanan, J. J., 1984. *Plant Environmental Measurements,* Bookmakers Guild, Longmont, Colorado, USA pp. 326.

22.Hatch, A. H., 1984. "Orchard Management With Help of Computers," *Proceedings of the 41st Annual Convention of Western Colorado Horticultural Society,* pp. 34-39.

23.Horowitz, P. and W. Hill, 1980. *The Art of Electronics,* Cambridge University Press, Cambridge, U.K.

24.Kelly L. G., 1967. *Handbook of Numerical Methods and Applications,* Addison-Wesley, Publishing, Reading, Massachusetts, USA, pp. 354.

25.Kettman, S. C., T. J. Yorkey, and K. Kaumarek, 1986. *Interfacing User Input Devices in Interfacing Sensors to IBM PC,* ed. by W. J. Tompkins and J. G. Webster, pp. 11.1-11.16 (in preparation).

26.Lambert, J. R., 1985. Editorial, *Computers and Electronics in Agriculture,* 1:1-5.

27.Mims F. M., 1985. *Forrest Mims's Computer Projects* McGraw-Hill, Inc., Berkeley, CA, USA, pp. 249.

28.Riedl, H., Croft, B. A. and Howitt, A. J., 1976. "Forecasting Codling Moth Phenology Based on Pheromone Trap Catches and Physiological Time Models," (*Entomol,* 108:4).

29.Penfold, R. A., 1986. "Temperature-analogue Interface," *Practical Electronics,* 22(4):28-31 (U.K.).

30.Rogoyski, M. K., 1987. "Computer - Aided Orchard Management," *Proceedings of the 44th Annual Convention of Western Colorado Horticultural Society,* pp.20-24.

31.Rogoyski, M. K., 1985. "Horticultural and Physical Aspects of Fruit Sunburn," *Proceedings of the 42nd Annual Convention of Western Colorado Horticultural Society,* pp. 39-42.

32.Siemer E. G., 1977. "Colorado Climate," *Colorado Experiment Station publication.*

■

MOVING?

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.

The ZCPR3 Corner

by Jay Sage

I wonder if the approach of a TCJ deadline will ever find me with no pressing commitments to interfere with my writing this column. That is always my dream, but the prospects of its happening get dimmer by the month. My schedule just keeps getting worse and worse. This time I was not even able to start the column until after the deadline had passed. As a result, it will surely be shorter than usual, and it will probably not have benefited from my usual multiple rewrites and the careful scrutiny of Howard Goldstein, on whom I have relied in the past not only to check my code but to check my writing as well. He is remarkably good at both jobs!

As has become the pattern for my columns, before I turn to the main technical subject, which will be a discussion of some of the capabilities of the ZFILER shell, I would like, to talk about a few nontechnical issues.

Z Systems Associates

It may have taken the retirement of Frank Gaude' and the demise of Echelon (at least as a force in the Z community) to make me realize just how much work Frank must have been doing, and I can now understand why he was so burned out in the end. The reason why I now appreciate his efforts in a way that was impossible before is that I have been in the process these last few months of setting up a new company—Z Systems Associates or ZSA—to serve as a central marketing organization for Z-System and related products.

Frank's retirement probably could not have come at a worse time for us. With NZ-COM and Z3PLUS we finally had Z-Systems that did not require a programmer's mentality and programmer's abilities to be able to set up and use. We were also no longer limited to CP/M-2.2 computers. Our potential market had now become the literally millions of people with CP/M computers of any kind, including especially the Commodore 128s and Amstrads running CP/M-Plus and the CP/M-2.2 ADAMs. We have never had any contact with those communities in the past, and it is going to take a lot of work to develop those contacts now.

The community of CP/M-2.2 hobbyists will, of course, be our marketing front line. To that end, we have established two plans, one involving the Z-Node remote access computer systems and the other, the hundreds of computer clubs around the world.

The Z-Node Plan

Echelon had a nice plan that recognized the important role Z-Node sysops play in disseminating information about the Z-System and in providing support to those who use it. It allowed Z-Nodes to act as dealers for Echelon's products and thus to gain some compensation for their efforts. Z Systems Associates has started a similar plan. I will not go into the details here, but if you are a Z-Node sysop and have not heard from me, please drop me a line at the address in the Sage Microsystems East ad.

Unfortunately, we do not have a list of the names and addresses of the Z-Node sysops. Somewhere along the way that information got lost and was never passed on from Echelon. So, a couple of weeks ago I sat down at my computer on a Saturday afternoon and started to call all the Z-Nodes in the United States and Canada. I did not count how many numbers were listed in Echelon's last ZNODES.LST file, but there must have been well over 50. At first I tried to figure out which ones were accessible by PC-Pursuit, but the task was monumental enough without having to put up with the perpetually busy PCP circuits. My phone bill came a few days ago, and it was amazing to see the number of pages of calls. Since each call was rather short, however, the bill was surprisingly low.

Although I started making the calls in the afternoon, I kept at it well into the night. At a point when I could hardly keep my eyes open, I connected to a system out West and, as a new user, went through the procedure of identifying the city and state from which I was calling. The system then greeted me very nicely and reported that the time was 1:05 a.m. Amazing, I thought! That was just what my watch showed here in Boston. This was the first system I had ever called that was so sophisticated that it actually adjusted the time display to the caller's time zone. Well, it wasn't until the next morning that I realized that the battery in my wrist watch was failing and that the watch had jumped back three hours. It had really been 4 a.m.! No wonder I felt so tired.

The whole experience of calling the Z-Nodes was rather disheartening. I discovered that many Z-Nodes had long ago gone off the air. The numbers of the more recently departed were answered with messages from the phone company reporting that they were no longer in service. Those long gone were answered by actual people, who usually had had that number for over a year. Considering that when my watch said 11 p.m., it may actually have been as late as 2 a.m., it is amazing how civil all of these people were to me. Yes, they admitted, they did get an awful lot of calls with no one on the other end. I explained why this was happening, that they were being called by a computer, and I

Jay Sage has been an avid ZCPR proponent since version 1, and when Echelon announced its plan to set up a network of remote access computer systems to support ZCPR3, Jay volunteered immediately. He has been running Z-Node #3 for nearly five years and can be reached there electronically at 617-965-7259 (on PC-Pursuit) or in person at 617-965-3552 or 1435 Centre St., Newton, MA 02159.

Jay is best known for his ARUNZ alias processor, the ZFILER file maintenance shell, and the latest versions 3.3 and 3.4 of ZCPR. He has also played an important role in the architectural design of a number of programs, including NZ-COM and Z3PLUS, the new automatic, universal, dynamic versions of Z-System.

In real life, Jay is a physicist at MIT, where he tries to invent devices and circuits that use analog *computation to solve problems in signal, image, and information processing.*

---

promised to try to get their numbers removed from the lists.

Several Z-Nodes had become private systems, and I was prompted for a password without any signon message at all or just a brief, "private system, enter password." A few systems, as one would expect, had gone over to MS-DOS. All in all, no more than half the nodes on the list appear still to be active.

In view of this situation, I would like to encourage the establishment of new nodes. Some of the nodes on Echelon's list, I learned, actually never went into operation because the sysop was unable to get Z-System running on his computer. With NZCOM and Z3PLUS this will be much less of a problem. If you think you might be interested in setting up a Z-Node or converting your present system to one, please give me a call or drop me a note in the mail.

The Z-Plan for Computer Clubs

Computer clubs are probably the single most effective and valuable source of support to computer users. To promote membership in clubs and at the same time to encourage more people to take advantage of the new Z-System software, we have established a plan whereby clubs can purchase the programs at a discount. It is their option how that discount is distributed. It can be passed on entirely to the club members, or the club can keep at least part of it to support its activities.

The Z-Plan project was initiated by Tony Parker, who serves as a marketing representative for ZSA. He uploaded a file to many bulletin boards describing the plan, and you may be able to find it on a system near you. ZSA had not yet been formed at that time, and the file instructs clubs to send the necessary registration forms to Alpha Systems. Alpha Systems has agreed that ZSA should take over responsibility for the administration of this plan.

A revised version of the Z-Plan file should be on systems by the time you read this column, but if you already sent information to Alpha instead, it would be a good idea to send another copy to ZSA (again, see the Sage Microsystems ad for the address). More importantly, if you belong to a club that has not registered, have them write to me requesting a description of the plan and registration forms.

My New Amstrad Computer

One of the computers on which Z-System now runs and one which exists in very large numbers, especially in Europe, is the Amstrad. This CP/M-Plus machine—once sold in the US by Sears, Roebuck—uses a very unusual 3" diskette (unique might be a better word for it). We were afraid that we would not be able to produce Z3PLUS diskettes for this machine, so I bought one second hand (just what I needed, another computer!). Since it appeared that we really had to have the machine, I probably paid a good bit more than it would otherwise be worth, but I must say that it has been a very pleasant surprise. (On the other hand, paying close to $400 for a hundred diskettes was a most *unpleasant* surprise.)

I expected the Amstrad to be little more than a toy. Instead, I have found it to be a very capable machine and an excellent platform on which to run Z-System. The main reason for this is its very nice RAM disk. My PCW8512 (it started life as a PCW8256, but it was upgraded) has a total of 512K of RAM, about 350K of which is available as a RAM drive. With ARUNZ, EASE, ZFILER, all their support files, and a few other critical files on the RAM drive, the Z-System really zips along.

The native (non-CP/M) mode has also proved to be quite useful. The Amstrad was largely promoted as a stand-alone wordprocessor, and its Locascript wordprocessing software is actually rather nice. Like Macintosh software, it is very easy to use, and my 12-year-old son has really taken to it in a way he never did to my super-sophisticated PMATE editor. The keyboard is set up specially for the software, and the computer includes an integral printer.

That printer is, in fact, rather interesting in its own right. To keep the cost down, Amstrad buys just a raw printer mechanism, and they supply the software drivers to emulate an Epson FX80 in the host machine. The printer is pitifully slow, but I actually use it myself now for quick letters because the Amstrad is so much faster to set up than my fancy systems. The printer even loads single sheets of paper automatically, and that's more than my (at one time) $3000 Diablo HyType-II printer can do. If you have an Amstrad computer or know of someone who does, I would be happy to help them get started on Z-System.

Special Acknowledgments

Before we go on to ZFILER. I would like to acknowledge publicly some very special recent contributions to the development of Z-System.

David Johnson of Sunnyvale, CA, took my ZCPR34 source code and must have gone over it not only with a fine-toothed comb but with an electron microscope. In programming, I always give top priority to writing code that has good features and runs reliably. Code compactness is secondary. Thus, I am never surprised when I learn that one of my routines can be improved slightly. Nevertheless, I would never have believed that the Z34 code could reduced by close to a hundred bytes, but David Johnson did just that! Even Joe Wright, who is deservedly acclaimed for his coding skill, had only taken out 10 or 20 bytes, and he was especially impressed by David's achievements. With all the new space available, I can now start to think about more new features!

The second person whose contribution I want to acknowledge is Bill Tishey of Severn, MD. Bill has proven something I have been trying to get across for a long time: that you do not have to be a programmer to contribute in a significant way to Z-System. Bill has systematically compiled the documentation for the complete collection of Z-System programs in a set of help files that runs (uncompressed) to more than a megabyte! He has grouped the help files into libraries with names of the form Z3HELP-n.LBR, where 'n' is the first letter of the command name. Z3HELP-A.LBR, for example, contains 21 files covering ARUNZ, AFIND, ALIAS, and many other commands. To my mind, this contribution is at least as valuable as those of program authors because it makes the programs accessible to the user.

The complete set of files is posted on my Z-Node #3 and will gradually make its way around the world (slowly probably, because of their size). In view of the exceptional value of Bill Tishey's help system, Sage Microsystems East will make it available on diskette for only $10 (SME's usual copying charge is $15 to $20 per diskette). Here are the rules. You have to send (1) preformatted diskettes clearly marked with the exact format and sufficient to hold 800K of files; (2) a disk mailer for returning the diskettes (unless the one you sent the disks in is reusable); (3) return postage; and (4) a return address sticker. We can accept 8" SSSD IBM standard format (including 'flippy' diskettes) and most 5" soft-sectored formats (anything on the menus of Uniform or Media Master on either the SB180 or an IBM AT). Bill, himself, has an Apple and (I just spoke with him) is willing to make the same offer for diskettes in that format. His address is 8335 Dubbs Drive, Severn, MD 21144.

ZFILER, The Point-and-Shoot Shell

Now let's turn to the technical subject for this issue, the ZFILER shell. Having written about shells so much in the past few columns, I am tempted to jump right into the thick of the subject. However, judging from the number of new subscriptions that SME alone takes each month, TCJ must have lots of new readers with each issue. Therefore, I will begin at the beginning. Since time and energy are in short supply, however, I will not attempt to provide the same comprehensive documentation that I did for ARUNZ. Instead, I will concentrate on the basics, on the one hand, and on some of the special features that many users may overlook, on the other.

Z-System Shells

A Z-System shell is a program that takes over the user-input function of the command processor. The way this works is that the Z-System environment includes a special area in memory called the shell stack where shell command lines can be kept. Whenever the ZCPR3 command processor is finished processing all the commands that have been passed to it in the command line buffer (another special area in memory), it checks the shell stack. Only if no command line is present there does the command processor itself prompt the user for the next command line. If there is an entry in the shell stack, then that command line is run instead, and the user no longer sees the command processor directly.

Some shells, like the EASE history shell, while making a big change in how the system is *actually* running, make relatively little change in how it *appears* to run. A command prompt is still presented, and one enters commands more or less as usual. The difference is that one has a more capable editor at one's disposal, and the commands are saved to a history file from which they can be recalled, edited, and run again. As we shall see, the ZFILER shell presents the user with a dramatically different user interface.

What is ZFILER For?

Historically, ZFILER is a descendant in the line of file maintenance utilities like SWEEP and NSWP (hence the "filer" part of the name). File maintenance is generally concerned with copying files, looking at their contents, renaming them, erasing them, and so on. ZFILER provides all these functions and more.

ZFILER's immediate parent was VFILER, where the "V" stood for video. The TCAP facility in Z-System makes it easy for programs to take advantage of the full-screen capabilities of whatever video display terminal happens to be in use at any time. In contrast to applications under CP/M, Z-System programs need not be configured to match the terminal. It was, therefore, natural to build a file maintenance program in which the files are displayed graphically on the screen. When I decided to explore some new directions with VFILER, to avoid confusion I gave the program the new name ZFILER, for Z-System Filer.

The file maintenance tasks described above would not require a shell. Making the program a shell, however, allows it to go beyond the functions included in the program's own code. Because a shell can pass command lines to the operating system, ZFILER can perform any operation that the computer is capable of. Like a menu system, however, it helps the user by generating the commands automatically at the touch of a key.

When ZFILER is running, the screen is filled with an alphabetized display of the files in a specified directory, and there is a pointer that the user can manipulate using cursor control keys. If we had a mouse to move the pointer, it would be a little like having a Macintosh. Actually, it would be a lot more. It would be like having a mouse with fifty buttons! Once the pointer has been positioned on a file, pressing a key (or two or three) causes any of a great number of functions to be invoked to act on that file. We will describe how this works in more detail shortly.

Invoking ZFILER

Since ZFILER performs full-screen operations, a proper Z-System terminal descriptor (TCAP) must have been loaded. If you have not done that, or if you have selected a terminal that does not support all the functions ZFILER needs, then ZFILER will give you an error message. The TCAP, unfortunately, does not include information about whether dim or reverse video is used by the terminal, and since these two modes for highlighting regions on the screen are so different, ZFILER is made available in separate versions for each.

There is also an option to have either four or five columns of file names in the display. Personally, I prefer the four-column version, which gives an uncluttered screen with plenty of restful

white space and a very distinct, easy-to-spot pointer. Others think it is more important to be able to see the maximum number of files on each screen and prefer the five-column display.

Then there is the issue of support for time and date stamping of files. ZFILER contains the code for preserving the time stamps when files are copied. So as not to inflict the overhead of this code on those who have not implemented DateStamper (though they should do that!), ZFILER is also provided in versions with and without the DateStamper code.

If we supported all combinations of the above choices, there would be eight different versions of ZFILER. Typically, the distribution library contains four or five of the combinations. For example, a five-column file display is not particularly compatible with reverse video highlighting, because the reverse video of tagged files runs into the reverse-video pointer.

When you get ZFILER, you have to choose which version you prefer, extract it from the distribution library, and give it a working name (some of the early Z-System shells had to have a specific name, but you can give ZFILER any name you like). I prefer the name ZF, since it is very quick and easy to type, and I will use that name in all the examples that follow.

The general syntax for invoking ZFILER is

```
ZF filespec
```

where "filespec" is a standard Z-System ambiguous file specification (that is, it may contain the wildcard characters "?" and "*"). The filespec selects the directory area and the files from that area to be included in the screen display.

Various parts of the filespec can be omitted. If no filespec is given at all, then "*.*" for the currently logged directory is assumed. Similarly, if only a directory is specified (e.g., B: or 3: or B3: or WORK:), then all the files ("*.*") in that directory are displayed. If a file name/type is included, then it will serve as a mask on the files to be displayed. Thus "ZF WORK:*.DOC" will show only files of type DOC in the directory WORK:.

The directory and file mask can both be changed from *inside* ZFILER as well using the "L" or LOG command. I bring this up now because there is a confusing difference in the way the "L" command works. VFILER originally allowed one to change only the directory and not the file mask from inside the program. To save the user the trouble of typing the colon after a directory, its inclusion was made optional. Since users became so accustomed to this shorthand, it was carried over into ZFILER. Because of this, if you want to change only the file mask, you must remember to precede it with a colon. Otherwise your mask will be taken as the name of a directory (which generally results in an error message).

One brief aside for programmer types. ZFILER can be loaded from any directory. One of the special features of Z-System since version 3.3 of the command processor is that it allows a program to find out both its own name and the directory from which it was actually loaded, perhaps as the result of a path search. ZFILER builds the shell stack entry to invoke ZFILER under its current name from the directory in which it is actually located. This sometimes makes it run faster, and it allows ZFILER to be invoked from a directory that is not on the search path.

The ZFILER Display

The main ZFILER display contains three parts. At the top of the screen there is a message line. In the version of ZFILER that is current at the time I am writing this column (version 1.0L), this line contains, from left to right, the following information: (1) the directory that has been selected, in both DU and DIR (named directory) format; (2) the indicator "[PUBLIC]" if that directory is a ZRDOS public directory (if you don't know what this is, just ignore it); (3) the current time of day if DateStamper or one of the new DOSs (ZSDOS or ZDDOS) is running; (4) the program's official name and version; (5) the text string "Current File:"; and (6) the name of the file currently being pointed to (this changes as the pointer is moved).

At the bottom of the screen is a command prompt of the form

```
Command? (/=Help, X=Quit):
```

The cursor (don't confuse this with the file pointer) is positioned after this command prompt to indicate that ZFILER is waiting for you to press a key.

The center 20 lines of the screen show the selected files. The character string "-->" (only "->" in the five-column display) floats between the rows of file names and designates the so-called "pointed-to" file. Many of the ZFILER commands automatically operate on this file.

What we have described so far is the main ZFILER screen, but it is not the only one. As the command prompt suggests, pressing the slash character (or "?" if you prefer) brings up a help screen that summarizes the built-in commands of ZFILER. This help screen replaces the file display but leaves the status line at the top and the command line at the bottom, except that "/=Help" changes to "/=Files". As you might, therefore, guess, pressing slash again will take you back to the file display screen.

I do not know if anyone makes use of this feature, but all ZFILER command operations can be invoked from the help screen. Although you cannot see the file pointer, you can manipulate it in the usual way, and you can tell what file you are pointing to from the name displayed at the upper right on the status line.

ZFILER Commands

I am not going to attempt to describe all of ZFILER's commands, but I will try to list most of them. Basically, the commands fall into several classes.

One classification reflects where the code for the command resides. There are two categories:

A. Built-In Commands
B. Macro Commands

Class A includes the functions for which the code is part of ZFILER. Macro commands are like aliases in that they generate command lines that are passed to the command processor for execution. These commands make ZFILER a shell. In this column I will discuss only the built-in commands, and I will take up the more complex subject of macro commands next time.

A second classification depends on what the command acts on. Three categories describe the object of the commands:

1. the pointed-to file
2. a group of tagged files
3. neither of the above

We will begin the discussion with commands of class A3, resident commands that do not perform any action on the files.

Pointer Commands

Class A3 includes the commands that move the file pointer. These are shown on the help screen, and I will not list them here. One can move the pointer to the next file on the screen or to the previous one (with wraparound); up, down, left, or right (with wraparound); to the first or last file on the current screen; or to the very first or very last file of those selected by the file mask. One can advance to the next screen of files or to the previous screen. Obviously, some of these functions will be redundant in some cases, such as when all the selected files can fit on one screen (think what happens when there is exactly one file selected).

ZFILER learns from the TCAP the control characters sent by any special cursor keys on the keyboard (provided they send a single control character and provided the TCAP has been set up correctly), and it makes them generate the up, down, left, and right functions. If the cursor keys generate control codes normally used for another function, then that function will be lost (the cursor keys take precedence). That can cause problems. One

solution is to eliminate the definition of the cursor keys in the TCAP and simply use the default WordStar diamond keys for those functions. Alternatively, one can patch ZFILER to use different keys for its own functions, but this is not straightforward to do, and I will not describe it here.

The "J" (Jump) command allows you to jump to a file that you name. This is very handy when there are many files in the display or when the file you want is not on the current screen. Press the "J" key, and you will be prompted for a file name. You do not have to enter the exact name. ZFILER automatically converts what you type into a wildcard filespec, and it finds the first file that matches. For example, if you enter only "Z" followed by a return, this is equivalent to "Z*.*", and ZFILER will move the pointer to the first file that starts with a "Z". Similarly, if you enter ".D", ZFILER will move to the first file with a file type that starts with "D".

The "J" function is very handy; however, there is more. Many people are not aware that you may press control-J to repeat the same search and find the next matching file. The search will wrap around from the end of the files back to the beginning. This function is not listed on the help screen because I could not find room for it.

Other Non-File Commands

Some other commands that do not act on files are: X, L, A, S, E, H, Z, and O. "X", as the command prompt reminds you, is used to exit from ZFILER. Besides terminating the current execution of the program, it also removes ZFILER's entry in the shell stack (if it did not, you would just reenter it right away).

We already spoke about the "L" (Log) command earlier. The "A" (Alphabetize or Arrange or Alpha sort) toggles the way in which the files are sorted, namely alphabetically by the file name or by the file type.

The "S" (Status) command prompts you for a disk drive letter and then tells you the amount of space remaining on that disk.

The "E" command (refresh scrEEn—I know that's stretching things, but "R" was already used) redraws the screen. You might think that this would never be needed, but there are two circumstances in which it comes in very handy. One is when ZFILER is being used on a remote system. It is true that very few RASs make ZFILER available, but I do on Z-Node #3. If you get some line noise, the screen can become garbled. Then the "E" key can be used to draw a fresh screen.

The other circumstance in which the "E" command saves the day is with Backgrounder-ii if you do not have a screen driver (I don't for my Concept 108 terminal—never got around to writing one, partly because all the programs I use frequently have a redraw key like this one). I simply define a BGii key macro specifying "E" as the "redraw" key, save the key definitions to ZFILER.BG, and attach that definition to ZF.COM. Then whenever I swap tasks back into ZFILER, BGii simulates my pressing the "E" key, and the screen is redrawn. This often gives a faster screen refresh than one gets with a full-fledged screen driver.

The "H" (Help) command generates a macro command to invoke the Z-System HELP facility. To tell the truth, I have not used this and don't even remember precisely what it does. I would have to look at the source code.

The "Z" (Z-system) command prompts you for a command, and whatever you enter is passed on to the Z-System multiple command line buffer for execution. When that command line is complete, ZFILER is reinvoked automatically.

When you use the "Z" command, you will normally be logged into the directory that is currently displayed. However, this will not always be possible. ZFILER allows you to select directories with user numbers from 0 to 31. Unless you are using a version of ZCPR33 or ZCPR34 with the HIGHUSER option enabled, you cannot log into user areas above 15. In that case ZFILER will put you in the directory your were in when you invoked ZFILER. In any case, the command prompt will indicate the directory from which your command line will be executed.

Since commands you run using the "Z" function may put some information on the screen that you would not want ZFILER to obliterate immediately, there is a flag set that signals ZFILER to prompt you and to wait for you to press a key before putting up its display. Here is a tip for advanced users. If you enter your command line with one or more leading spaces, this shell-wait flag will not be set, and ZFILER will return without your having to press a key. The leading spaces are stripped from the command line before it is passed to the command processor. This means that you cannot use a leading space to force invocation of the extended command processor (ECP); you have to use the slash prefix instead. A space and a slash will force invocation of the ECP and will disable the shell-wait flag.

The final command in class A3 is the "O" (Options) command. It is a complex topic, and I will leave it for next time. If you can't wait until then, experiment with it. It should not be able to do any harm to your system.

Single-File Built-In Functions

Now let's discuss the commands in class A1, the built-in commands that act on the pointed-to file. These are invoked by pressing one of the following keys, whose meaning is indicated in parentheses: C (Copy), M (Move), D (Delete), R (Rename), V (View), P (Print), F (File size), T (Tag), and U (Untag). Some of these are self-explanatory, and I will not discuss them.

The "C" command copies a file to another directory under the same name; it does not allow one to give a new name for the destination file (however, you can do that with a macro command). The "M" command does not really move a file; it copies the file and then, if the copy was successful, deletes the original file. It is really a combination of "C" and "D". Moving a file this way is inefficient if the destination directory is on the same drive as the source file. A macro command that invokes an ARUNZ alias can get around this limitation (and almost all other ZFILER limitations).

The tag and untag commands are used to select a group of files on which operations can be performed. Tagged files are indicated in two ways. A special character ("#") is placed after the file name in the display, and, if the terminal supports video highlighting, the file is highlighted.

Two related commands are W (Wild tag) and Y (Yank back?). "W" allows you to tag or untag groups of files designated by an ambiguous file spec. After tagged files are operated on by the built-in group commands described below, the tag marker "#" is changed to "'" (a soft tag). The "Y" command changes the soft tags back into hard tags so that further group operations can be performed on those files.

Built-In Group Commands

Group commands are initiated by pressing the "G" (Group) key. The command prompt at the bottom of the screen changes to

```
Command? (/=Help, X=Quit)  Group: (A,C,D,F,M,P,R,T,U,V)
```

For now we will consider only the built-in group functions (class A2) and will take up group macro commands (class B2) next time.

Except for the four functions described below, the letters invoke the same action as the individual command corresponding to that letter, but the function is performed on all the tagged files. We will not discuss those further. Note in particular that the keys "A" and "R", however, have a group function that is completely different from the individual function.

The "U" and "T" group functions do not act on the tagged files; they change the tagging. The former untags all files; the latter tags them all.

The "R" group function is another one that does not, strictly speaking, act on the tagged files. It reverses the tags, tagging the files that had been untagged and untagging the ones that had been

tagged. This can be very handy in several circumstances. For example, you might want to copy all the files except two. It is easier to tag those two and then to reverse the tags. As another example, you might want to copy some of the displayed files to one diskette and the others to a second diskette. I do this frequently. I begin by tagging the ones to go to the first diskette. Then I group copy ("GC") them to the destination diskette. Next, I yank back the tags using the "Y" command and then reverse the tags with "GR". Now I can group copy the rest to the second diskette.

The "A" (Archive) group command is very handy for automating backups. When it is entered, the tags are removed from any tagged file whose archive flag is set. As a result, only files that have been modified since the flag was last set will remain tagged. In addition, the "A" group command automatically initiates a group copy operation but with one special feature. After the file has been copied successfully, the archive flag on the source file is set to indicate that the file has been backed up.

Under later versions of VFILER, the group "A" command automatically tagged all unarchived files; under ZFILER it untags the archived ones. This difference is very important. With VFILER, you were forced to back up all the files selected by the VFILER file mask. Under ZFILER you can select the files that will be candidates for backing up. If you want the achieve the same function as under VFILER, just tag all the files first with "GT" and then archive them with "GA". On the other hand, if you want to exclude BAK files from the backup, you can "GT" all files, untag the "*.BAK" files using the "W" command, and then use the "GA" command.

After you enter the command "GA", you will be prompted for a destination directory. You do not have to supply one! If you simply enter a carriage return, the copy operation will be skipped, and you will be left with tags on the files that need to be backed up. You can then use a macro function to back them up in a specialized way, such as crunching (compressing) them to the backup disk (instead of copying them as they are) or putting them into a library on the backup diskette. Next time we will discuss the macro techniques required to do this. ■

Editor

(Continued from page 3)

I prefer a simple, but powerful, editor without a lot of bells and whistles which I don't need for the kind of writing I do. For the initial work on book manuscripts, which may run 400 pages or more, I don't need multiple windows or on-screen italics and bold facing. I absolutely do not want WYSIWYG (What You See Is What You Get) or any fancy formatting. I just want a spell-checked ASCII file which I can dump to a dot-matrix printer for editing. It is foolish to talk about WYSIWYG at this stage because we will be adding and removing words, combining and breaking up sentences, restructuring and moving paragraphs, determining what illustrations and graphics are required, and all the nasty things that editors do.

After the editing is completed and we know how much text there is and how much space will be required for graphics, we'll define the design parameters (actually, rough layouts are done while editing). The design decisions include the front matter—Testimonials, Bastard Title, Frontispiece, Title Page, Copyright Page (Title Page Verso), Dedication Page, Epigraph Page, Table of Contents, List of Illustrations, The Foreword, The Preface, Acknowledgments, and possibly an Introduction.

Next comes the text design, which includes Divisions, Chapter Title Pages, Subheads, Footnotes, and of course, the basic page design including line length, column length, type size, line leading, and type fonts.

This is followed by the back matter—such as The Appendix, Authors Notes, The Glossary, The Bibliography, The Index, The Afterword, and possibly a Colophon and/or a Coupon.

At this point we want to see hard copy for the front and back matter, a few (3 to 4) sample text pages, plus a report of the exact number of pages in the books and a printout of all the design parameters for all three sections. It is very important to know the exact number of pages because trade books are printed on large presses which may print 16 or 32 pages on a single sheet. These pages, called signatures, are bound together to produce the book, and the bindery operations become very expensive if the book does not use an integer number of signatures. For example, if the printer's standard signature is 16 pages, it is less expensive to produce a 48 (16 × 3) page book than it is to produce a 44 page book—you either expand the material or add four blank pages (now you know why you see blank pages in many of the books).

The reason I want a total page report is so that I can adjust the number of pages by changing the parameters (line length, line leading, column length, etc.) before I set the book—it can be very expensive to reset a large book using high resolution Linotronic® output. The reason I want a printed list of the parameters is that I may present three or four alternatives to the client with sample pages, and I want to be sure that I remember exactly how a particular sample was produced—even if the client comes back six months later.

Laser Programming

Within the next few years, laser printers will replace the daisy wheel and dot matrix printers in almost all business applications except possibly for mailing labels (and that's being taken over by ink jet printers which print directly on the mailing piece).

We are going to have to learn to talk directly to the laser printers, just as we did with daisy wheel and dot matrix printers. We'll have to either learn Postscript® and PCL® or have libraries with the needed routines so that we can access the laser directly from our programs.

I'm starting to design a laser driver designed for publishing books, because the available programs do not meet my requirements. Feedback and articles from others working with lasers will be appreciated—as will suggestions for additional features (let us know what you can't do with existing programs).

If we get enough response, we will add a regular section. ■

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, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder ii, DosDisk; Plu*Perfect Systems; Clipper, Nantucket; Nantucket, Inc. dBase, dBase II, dBase III, dBase III Plus; Ashton-Tate, Inc. MBASIC, MS-DOS; Microsoft. WordStar; MicroPro International Corp. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C; Borland International. HD64180; Hitachi America, Ltd. SB180 Micromint, Inc.

Where these, and other, terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

Real Computing

The National Semiconductor NS32032

by Richard Rodman

Return with me to the halcyon days of yesteryear, when Grace Hopper pulled literal bugs from the relays of the ancient computing machines. Before there was magnetic core, memory was storage CRT's with photocells, or it was a rotating magnetic drum. Alan Turing scattered instructions throughout the drum so that the next instruction would be coming under the head just as it was needed.

Back in those days, an address was an address. If you went to address 0410, those were the address bits that went direct to memory. The first generation of microprocessors and personal computers all followed this flat, linear addressing scheme, with a 64K addressing space with 16 bits of address.

When Zilog designed their 16-bit microprocessor, the Z-8000, they wanted to be able to address more than 64K bytes, so they implemented segmentation, a scheme which had been implemented on the PDP-11 earlier. The Z-8000 used a memory management unit (MMU) that allowed any 64K segment to exist anywhere in a 16M addressing space. Intel's 8086 implemented a simple segmentation scheme using a fixed translation. This segmented architecture gave rise to the second generation personal computer, now in vogue.

The third generation personal computers soon to come will feature an enhancement developed long ago in the mainframe and minicomputer world, virtual memory. Virtual memory means that you can run a program that needs 256K on a system having only 32K. The computer will save a memory image on disk and "move" the physical RAM around as needed by manipulating the translation tables. The only way to do this without causing fragmentation is to handle the memory in blocks of fixed size, called "pages". This is what is known as "demand-paged virtual memory" (DPVM).

Now, all of the major 32-bit microprocessors claim to support DPVM, the 68010/68020, 80386, and 32016/32032. But in the case of the Intel and Motorola chips, DPVM has been ad-

--- LISTING ONE ---

```
;MMUTEST.A32 -
;This is a program to test the operation of the 32082 MMU.
;The  level-1 page table (1024 bytes) is built at  00000400,  and
;the level-2 page table(s) (512 bytes each) at 00000800 up.

;Assemble with Z32.  Remember that in Z32, numbers beginning with
;0 are interpreted as hexadecimal.

;880925 rr initial version

LEV1ADDR  =  00000400
LEV2ADDR  =  00000800

MODBASE
        DD      0
        DD      LNKBEG-MODBASE
        DD      CODBEG-MODBASE
        DD      CODEND-MODBASE
LNKBEG
CODBEG

;Clear all the tables to zero

        MOVD    LEV1ADDR,R1             ;where to start clearing
        MOVW    512,R2                  ;how many words to clear
CLEAREM
        MOVQD   0,0(R1)                 ;clear a word
        ADDQD   4,R1                    ;bump the pointer
        ACBW    -1,R2,CLEAREM           ;loop

;Set up level-1 table.   Each entry in the level-1 table controls
;64K bytes.

        MOVD    LEV1ADDR,R1             ;where level 1 table should be
        MOVD    LEV2ADDR+0007,0(R1)     ;entry 0 (00000000 - 0000FFFF)
        MOVD    LEV2ADDR+0207,4(R1)     ;entry 1 (00010000 - 0001FFFF)

;A couple of others are necessary for the I/O and PROM

        MOVD    LEV2ADDR+0407,768(R1)   ;entry for 00C00000 - 00C0FFFF
        MOVD    LEV2ADDR+0607,1020(R1)  ;entry for 00FF0000 - 00FFFFFF

;Set up level-2 table for 64K RAM no translation

        MOVD    LEV2ADDR,R1             ;where level 2 table 0 should be
        MOVW    128,R2                  ;how many entries
        MOVD    00000007,R0             ;first entry
L2LOOP
        MOVD    R0,0(R1)                ;store entry
        ADDD    00000200,R0             ;bump to next page
        ADDQD   4,R1                    ;increment entry pointer
        ACBW    -1,R2,L2LOOP            ;loop

;Set up level-2 table for I/O no translation

        MOVD    LEV2ADDR+0400,R1        ;where level 2 table for C0 should be
        MOVW    128,R2                  ;how many entries
```

```
        MOVD    00C00007,R0             ;first entry
IO2LOOP
        MOVD    R0,0(R1)                ;store entry
        ADDD    00000200,R0             ;bump to next page
        ADDQD   4,R1                    ;increment entry pointer
        ACBW    -1,R2,IO2LOOP           ;loop

;Set up level-2 table for PROM no translation

        MOVD    LEV2ADDR+0600,R1        ;where level 2 table for FF should be
        MOVW    128,R2                  ;how many entries
        MOVD    00FF0007,R0             ;first entry
PR2LOOP
        MOVD    R0,0(R1)                ;store entry
        ADDD    00000200,R0             ;bump to next page
        ADDQD   4,R1                    ;increment entry pointer
        ACBW    -1,R2,PR2LOOP           ;loop

;All ready to turn on the MMU chip

        SETCFG  [M]                     ;enable use of MMU instructions
        LMR     PTB0,LEV1ADDR           ;set level 1 page table address
        LMR     MSR,00430000            ;set mmu status register
        RXP     0                       ;return to srm
CODEND
        END
```

ded on to the existing processor. National's architecture, on the other hand, was designed around implementing DPVM at the outset. Most current NS32 designs include the 24-bit MMU, the NS32082.

The 32082 Memory Management Unit (MMU)

National's 32082 MMU is tightly integrated with the CPU. It is connected by the slave processor bus, a private communications channel to the CPU used by it and the FPU, and the CPU has special instructions for programming the MMU. Thus, in any NS32 system, the MMU will always be programmed the same way. It is as though the MMU were on the same chip as the CPU, and it is in the case of the 32532.

The MMU's basic function is *address translation*. This means that the 24-bit address from the CPU is translated to a 24-bit physical address. By means of this translation, we can not only rearrange our memory space at will, but we can also *protect* parts of it from being damaged. For example, a user task can have its address translation set up so that it cannot access the operating system's memory areas. We can also catch a program that goes haywire and stop it cleanly. We can catch a program that tries to use too much memory—and give it some more, if we think it deserves more.

The MMU also has a number of other features such as hardware breakpoints. It keeps track of accesses to the memory pages, so that we can know if they have been used or changed.

Basically, the MMU points to a table in memory which, in turn, points to a number of other tables. The first table is the "level 1 page table." This table selects, based on the high-order byte of the 24-bit address, which of the level 2 page tables will be used. Thus, the level 1 page table has an entry for each 64K bytes of memory (we'll call this a "superpage"). In the 32082, this table can have 256 entries, so it takes up 1024 bytes.

The level 2 table entry pointed to by the level 1 table then supplies the physical address, down to the page boundary (the page size is 512 bytes). One of the most pleasant aspects of the 32082 MMU is that the address bits are in the right places. For example, bit 10 of the address corresponds to bit 10 of the table entries.

The low-order 9 bits of the page table entries are used for the write-protect, modified and accessed flags. Any given entry can also be flagged as an invalid reference by having its low-order bit be zeroed.

I've included, in Listing 1, a program which sets up a one-to-one translation of addresses 000000-00FFFF, C00000-C0FFFF and FF0000-FFFFFF, then enables the MMU. What happens when you run it? Nothing seems to happen. However, if you access memory outside of those areas, the access will be caught. (The first time I ran this program, I'd forgotten about the I/O and PROM memory areas, and nothing worked!)

Once that program is running, you can start moving your memory around. I set up the level 1 and level 2 table for 010000-01FFFF also. Now, to move an unused page of memory from 001000 up to address 010000, just store the value 00001007 at location 000A00 (remember, LSB first).

The program as listed will work under the SRM monitor. You can modify it easily for TDS or some other monitor. If your RAM does not begin at zero, change the equates at the beginning. The level 1 table must be on a 1K boundary, and the level 2 tables must be on 512-byte (page) boundaries.

How Do We Use the MMU in an Operating System?

In an operating system, we can use separate page table sets for the system mode and for the user mode. The operating system should have read-write access to all of the system memory; the user mode should have access only to memory areas for the running task. Any attempt by a user task to use memory outside of its allotted area will be caught by the MMU and handled appropriately by the OS.

Task switching can be done quickly by having a separate level-1 page table for each task, which takes 1024 bytes. When switching tasks, the supervisor needs only to load the PTB1 register with the task's level-1 table address. If we only allocate task memory on superpage (64K) boundaries, then we only need one set of level-2 page tables. The write protection, et al., can be handled by the level-1 tables.

The user task must have read-only access to the system base superpage (000000-00FFFF) so that it can make service calls. It should have read-write access to its own memory.

Page swapping would be done on a 512-byte page basis. The valid bit would be cleared on a page being swapped out. If the modified bit in the entry is not set, there is no need to write the page to disk. When an access occurs to a swapped-out page, the MMU will generate an abort trap. The OS will see if the page is valid but swapped out. If so, it would free some other physical page (the least-recently-used one, perhaps) and assign it to this logical page, then reload the memory from disk. Upon the return from the abort trap, the instruction that caused the trap would simply be re-executed.

More on the Free Operating System

Version 1 of the free operating system will be single-tasking, like CP/M or MS-DOS, and will not make use of the MMU. It is currently called Bare Metal (Metal for short).

In version 2, multitasking will be implemented, and an MMU will be required. Most NS32 systems already have MMU's, and if not, the chip is not expensive. We will have memory protection and virtual memory, and hope to implement MMU-based, built-in debugging. Yet, the system will maintain the ease of use and customization of version 1.

We will be needing lots of programs and utilities for the OS. If you'd like to join this brave group of hardy pioneers, please write to me care of this magazine or to my home address.

(Continued on page 39)

Sprint, The Professional Word Processor

A Product Review

by C. Thomas Hilton

While awaiting my evaluation copy of Borland's new word processor, Sprint® , I was intrigued by the varied evaluations in other publications. Some reviewers loved it, some hated it, and some could not make it work. In all cases I had the feeling that the reviews left something to be desired.

Installation

My first impression of Sprint is that of another fine Borland product. I had no problem loading or installing the system. Further, I am using Sprint to write this column, less than an hour after taking the system out of the box.

If you take the time to read the screens presented to you by the installation program, you can install Sprint with confidence. I found it difficult to improperly install Sprint! I certainly tried to anticipate every error a user could logically make. The system is just too smart to accept too much bad information in the installation process.

The Nature of the Beast

Once the installation sequence has been completed, Sprint is ready for use. Sprint is a very complex word processing system. As with all complex and powerful software, if you are not attentive to detail, you have the capability to get yourself into trouble. Sprint's complexity is well hidden, and you do not need to deal with the underlying structure of the system, unless you want to. Should you desire to deal with the program at the base level, expect to spend a nominal amount of time with the program's documentation. While Sprint can be used without ever touching a manual, it is complex enough in its design to require a minimal understanding fundamental design concepts.

The bottom line is, if you have used other Borland products, or are familiar with WordStar® , Sprint is usable right out of the box!

Who is Sprint For?

I could be convinced that Sprint may not be the best choice for the novice who just has to know everything about everything, without an understanding of his limitations, or who must tinker with every possible option. Again, the availability of power brings with it the ability to get yourself in to deep trouble, quickly.

Out of the box, Sprint does not implement all the features it is capable of, but neither does XyWrite III+® , another popular, professional, word processing system. Some may feel that Sprint needs to be configured to accommodate individual preferences. The difference between Sprint and other professional systems is that you can add features as you recognize the need for them. In the meanwhile, the program is perfectly usable.

Sprint is a system that can grow with a person's individual skills. As your skills and understanding develop, so may Sprint's capabilities. Even if you are a seasoned professional, Sprint will not restrict your growth potential. Such a dynamic nature is made possible by the fact that Sprint is only marketed as a "Word Processor." In actuality, Sprint is something totally different and amazing.

Will the Real Sprint Please Stand Up

Sprint is a word processing development environment. It has a built-in compiler. Sprint's programming language is similar in syntax to the "C" language. In typical Borland fashion, the complete source code for the system is included in archive format. While you have the opportunity to alter the source code, and compile yourself a creature of your own design, you are not required to do so.

This softly spoken feature is one of many unexpected treats. Sprint presents itself, out of the box, as a powerful system in its own right, but permits you the ability to change the entire system to something more to your liking. It is quite easy to modify the Sprint system into something totally yours, and totally unrecognizable as the original product. Sprint will not soon become obsolete, as you can add any features you feel are missing from the base model, or delete features you may not care for. This feature is in itself more than enough for me to grow instantly fond of the product.

Format Conversions

Until Sprint arrived I had a number of text editors and word processors that I was forced to use quite often. The reasoning was simple, some publishers like text in a Wordstar type format. Another magazine would prefer another format, and so on. Sprint has the ability to import from and export to a wide variety of alien product formats. With Sprint, a single system can provide all the formats that are normally required of me. Instead of becoming marginally skilled in the use of many text editors and word processors, I am free to devote myself to only one system.

A separate value of Sprint's import and export facilities have to do with an experience at my office worthy of sharing with you. The client was new to computers, and did not have any idea as to what editing system was right for his application. The State of Montana specified one popular word processor that had such a steep learning curve that productivity suffered. As each new editor was tried, existing text files had to be converted to the new editor's format. In some cases, important files were lost during the conversion frenzy. Using Sprint would have allowed them to convert their files into the native format without the loss of a single character.

Sprint's versatility does not end with the ability to import and export document files. In the series we will be presenting on Information & Knowledge Engineering, we need to access information in the process of preparing reports. Sprint will allow access to Reflex® , Paradox® , and dBase III® data files. Data format compatibility is a concept that is often overlooked when software products are selected. Data compatibility is critical to the Information Engineer. The authors of Sprint understood this concept, and made provisions for importing information, within the scope of what a word processor is supposed to do. The Sprint manuals clearly define the procedures required to import data from other Borland products, as well as popular data base management programs. As our series develops, the true importance of these concepts will be recognized.

The Learning Curve

The concept of the learning curve is very important in a production environment. Sprint has the ability to emulate the command sequences of popular editing and word processing systems.

The "look and feel" of the editing screen, when an alternate user interface is used is pure Sprint, not that of the emulated product. There are many reasons for not emulating the "look and feel" of another product. Not the least of which has been those software publishers who felt the need to litigate the concept. It is important for the user to recognize that Sprint is not one of these other products. There are some variations, when emulating another word processor system, from the expected way exotic command sequences may function. I can not disagree with the way Borland chose to approach this concept. If you want to emulate Word Perfect® exactly, then purchase a copy of Word Perfect!

The alternative user interface options allow temporary or new workers to use the system with the least application of the learning curve. The screen display remains Sprint, but the basic editing commands emulate a familiar editing package. With a familiar environment, the new worker is faced with a reduced learning curve, and increased productivity. In attaining this goal Borland has done an excellent job! Will the person who has become immediately functional with an alien software product please stand up and tell his tale? For we mere mortals time is required to "get up to speed" with any new product. I wish to wax redundant and say that I began this article less than an hour after receiving the product. Your experience with Sprint may, or may not be similar.

The Many Faces of Sprint

Sprint is not a WYSIWYG, (What You See Is What You Get), type of system, technically speaking. It does a reasonably good job of faking it though. Sprint consists of an editor and a text formatting and printing module. Each of the two parts may be used as a stand-alone program. The use of these two parts of the system is something many would not be aware of.

Many professional writers prefer to use an external text formatting and printing program. The primary reasons for this are founded in the fact that off-the-shelf editors, and word processors, lack the needed text processing power. An external program allows you to build a document "style" with great sophistication. Sprint has skillfully presented the best of all possible worlds in combining a user transparent text formatting program with a powerful editing system.

The debate over the value of an external, post-processing type of text formatting & printing system is an old and unresolved one. It centers on the quantity of the work to be done, basically speaking. Some would also argue that the issue of the quality of the work to be done is equally important. If all you want to do is write a few memos, or letters, then perhaps Sprint's power may be overwhelming—not the best choice for you. If, on the other hand, you need to prepare large, or complex reports, as would be consistent with our IE series, then Sprint deserves your serious attention. But, if you have a need for consistent looking memos and correspondence, Sprint will allow to to create a document style, or "format" that will exceed any need you may demand of it.

Sprint's screen display is clean. It isn't as fast as XyWrite II + in some key responses or screen updating. But even on an eight megahertz V-20 machine the performance is acceptable.

The spelling checker is healthy, containing some 100,000 words. Its response is slower than some spelling systems, but that could be because of its large word selection. I hate those spelling systems which omit the words I most often misspell. Sprint's system has shown me that it is as professional as the rest of the system. The Thesaurus is as healthy, though no faster than the spelling system. Sprint seems to know just how much delay I will tolerate, and does not abuse my tolerance. Of course, I really do not need to keep the auto-spell-check mode active all the time, which would probably speed the system up substantially. I don't really have a problem with Sprint's operational speed. It is certainly more responsive, faster, and cleaner than many of the other popular word processors I have been using. Keep in mind that I am more demanding than the average user. With operational bad points, as well as good points, considered together, Sprint is the only word processor active in my system, and I expect this circumstance to become permanent.

Sprint is Goof Proof

I read where Phillip Kahn has a habit of turning his laptop computer off in the middle of an editing session! Horror of horrors! Well, I just had to try it for myself! Imagine my surprise when I turned the system off in the middle of an edit, waited a few seconds for the system to settle, and turned it back on to find everything as it was, including the proper positioning of the cursor! This is a nice feature! Here in rural Montana the power is subject to go off at any time, for any reason. Knowing that Sprint will save my work every few seconds, without really making me aware of it, is in itself, enough to endear me to the product. The automatic save feature Sprint uses is not a new concept, but I have yet to have a product that implemented this concept without becoming intrusive. I hate it when the system stops every few seconds to write the text to the disk file. With Sprint, the only indication of this activity is when the disk access light goes on for a brief period.

Now Let's Talk Printing

I use an off-brand printer that I found in the pages of some discount rag. We have all heard the claims that this, or that, software package supports 50 million different printers. Well, Sprint is the first word processor that ever knew what a Seikosha printer was, and how to use it properly! Normally I have to install the Seiko as an Epson "work-alike" and hope for the best. I was quite pleased that Sprint knew all about the printer, and returned an excellent print quality THE FIRST TIME! I like the little Seiko printer, but was not thinking of purchasing another because it was too hard to interface with some software. With Sprint's intellect I will be buying another "discount" printer that will outlast the name brands, be just as fast, and have better font quality. Thank you Sprint!

When they say that Sprint supports over 200 printers, what they mean is that Sprint supports 200 printers that mere mortals buy and use. If you have anything from a laser printer to something left from over the kid's last Christmas, Sprint will find a way to use the beast to its maximum capability.

Fresh out of the box, I really didn't expect much from any printer and text formatting interface. A very large part of any editing system concerns itself with formatting and printing of text. There are word processors which attempt to allow you to print in the background and continue editing. Thus far I have never found a word processing system that can implement this concept properly. The problem is seldom in the word processing software, but in the ability of the printer to accept text at the speed at which the editing system wants to send it. If the printer is too slow in accepting data, then the word processor will "hang" for a fraction of a second waiting for the printer to catch up. If the printer can accept data faster than the word processing system can send it, then the word processing system spends more time than it should servicing the printer. In the text editors I have written it was always required that I adjust the balance of service between the key board and printer service.

Sprint does not attempt to print a document as a background process. The main reason is that the editor is inactive while the formatting and printing module has control of the environment. Most people use the time while a document is being printed as a break period. I admit to being one of those people. I do not miss an attempt to do two things at once.

The formatting of documents can be as complex as your

imagination will allow them to be. The text formatting program is smart enough to know about leaving reserved areas for photographs and graphics, though graphics are not totally supported by Sprint. If you make an error in a complex format command structure, the formatting module will advise you of the error. Of course, Sprint will work just fine without a single formatting command being used. It is this lack of reliance upon other modules that demands a great deal of respect for the product. How you prepare your document is entirely up to you. If you want to keep it simple, Sprint does not force you into any form of complexity. This is not the case with other programmable word processing systems.

Considering the way the industry has been heralding desktop publishing, I at first thought that Sprint's ability to only support POST SCRIPT® graphics was a serious defect. While I am disappointed that there isn't a whole world of graphic support in Sprint, I do understand the reasons why graphics were downplayed, and agree with Borland's concept. In fact, I have some question about the wisdom of including POST SCRIPT support. The reason is for this paradox is simple. What Borland wanted to do, with Sprint, was to present a word processing system that would last. Products in the computer industry have a very short life. Sprint has great appeal to professional writers, the target market. Professional writers generally do not, and cannot, put graphics in their text. This is a separate process in document production. Typesetting systems just do not understand graphics data. The number of writers who will actually become involved in the printing of their work is extremely low. That is just the nature of the publishing business.

While the desire to wax creative in a personal document is high, I have read too many tales of woe concerning the arrival of desktop publishing systems in an office environment. Multiple fonts and cute graphics are not really appreciated in the world where Sprint will be found most useful. This is not a fault of Sprint, but of humans such as myself, whose artistic perception is often found to be abhorrent.

The control Sprint gives you over your printer is total. Sprint knows just what your printer is capable of, and will do its best to produce documents of fine quality. I am not really thrilled with printer test programs. What Sprint caused my aging Seiko to produce was impressive, and I have been known to be highly critical of document appearance.

My old favorite, XyWRITE III+ would "snake" columns of text, (like newspaper columns), with the best of them, providing you asked it nice enough, did not alter the page format in midstream, and did not want to change fonts in column structure. If you changed fonts in one column, the placement of text in any column to the right of a larger, or smaller font, would be upset. This is one advantage to Sprint's use of a powerful text formatting program deal with the printing tasks. Sprint will allow you to change column formats anywhere in a single page, change fonts, do paragraph titles, chapter titles, or just about anything your imagination can devise.

The Sprint text formatting and printing program is so powerful that you would expect all the graphics features of a full powered desktop publishing system. But, graphics standards come and go. Sprint's ability to conform to the needs of the day is such that it will be around for quite some time. If you actually need a form of graphics interface for your application, you can create your own software driver, Sprint's approach to all things is an open design architecture. Remember that you can alter and recompile the complete Sprint system to suit your particular application. The basic Sprint concept, is quite profound: "Just because you don't see a feature you need does not mean Sprint is not capable of adapting to your desires." For my applications, presentation graphics would normally be too large to attractively mix with my text. For graphics I would prefer to use Quattro's excellent presentation graphics facilities.

Sprint's Documentation

Sprint comes with three thick manuals and a small manual dealing with alternate interface emulation. Sprint's manuals are written in classic Borland style. For some this will be enough said. For those who have never dealt with Borland products, allow me to say that Borland produces excellent manuals. Sprint's documentation is simply elegant, which ignores the fact that the manuals were developed using Sprint.

Whatever your level of understanding, there is a section of a manual that covers what you want to know, written in a manner to assure comprehension. Where one manual deals with complex inner workings of the development environment, sections are included with literal "How To" paragraphs.

To get the most from any system one must resign their self to spending some time with the documentation. While you may use Sprint with effectiveness and never look at a manual, to do so is similar to having purchased a fancy sports car with no intention of ever driving it!

Adapting Sprint to Your Needs

You do not need to be a computer programmer to adapt Sprint to your needs, whatever they might be. If you do take the time to look through the manuals, you will be impressed. I was surprised to discover that Sprint can be made to work with terminal based systems. A few years ago there were no lack of terminal based text editors and word processors. Today they are rare for the terminal based, multiuser, office automation system. When considering software for my information systems, it would be nice to have a word processing system that can easily be adapted. When the choice is between expensive networking systems and simple, inexpensive terminal systems, Sprint has again provided an impressive feature.

The many ways Sprint can be programmed is beyond the scope of this review. However, it is a subject that should be dealt with in greater detail. We all have the need to develop reports, of some design in our work. Each of our demands upon a word processing system are different. This would be a good place to share our experiences, and deal with problems which may be just beyond our immediate grasp. Let me hear from you if there is an interest in supporting Sprint though the pages of TCJ.

The Bottom Line

Sprint may not be suited for every user. The casual user may desire something else. On the other hand, Sprint can grow with an unskilled person as few other word processors can. For the professional, Sprint will be found to be an extremely powerful tool that will save many hours of document production. For the programmer, Sprint is a "dream" product that can be molded to suit any taste. For all users, Sprint will always produce professional looking documents, whether you like it or not.

No product is perfect, being all things to all people. In my work as a consultant I have always said that the word processor you need is very much dependent upon the work you intend to accomplish with it. Now I simply say, "All you really need is Sprint." Sprint comes highly recommended for all users.

If you are using Sprint, and have an idea, problem, hint, or tip to share, let us hear about it. TCJ exists as a forum for information exchange. While a regular column concerning a word processing system is not regular TCJ fare, Sprint is not a common product. Did you know that you can access DOS functions, or develop database structures from within Sprint? If Sprint will allow these functions, what else can a clever reader devise? Let us know, eh? ■

Using ZCPR3's Named Shell Variables

by Rick Charnes

Hello, Z-System aficionados. I have been so pleased that Jay Sage and Bridger Mitchell are writing for TCJ that I wanted to see what I could do to pitch in and help our grand cause along, and am honored to be writing for such a fine magazine. By way of introduction, I've had a Morrow MD-3 since April of 1984 which I purchased for writing. Around about the summer of 1986 a series of articles by Ted Silveira appeared in our local San Francisco computer magazine, *Bay Area Computer Currents*, about something called ZCPR3. Stars appeared, the ground shook, the walls trembled, and when I woke up I was a changed man. ZCPR3 has since been a part of my life that has provided an unending source of joy and utter delight.

I actually have been intending to write an article for TCJ for quite some time, but it was only recently that I completed a project about which I was so excited that I could no longer contain my enthusiasm. I find that my experience with Z-System is often like that: I go on and on using my own inventions, discoveries, aliases, ZEX files and other assorted odds and ends, mostly keeping them to myself until finally at some point I realize that what I am doing is really quite exciting and enjoyable. I figure if I like it so much there must be *someone* out there who might feel the same way. The Muse usually appears just around then and informs me in no uncertain terms that should I make any further attempt to keep what I've done to myself it will be judged a sin of vanity and punishable by a permanent visit to Computer Purgatory. A week or so ago I came to that point, and in the spirit of appeasing the spirits that animate our wonderful world I am enjoined to set the experience down on paper.

Jay Sage and I have for a year or more been having a contentious if good-natured argument about the use of the shell stack. Since my Eureka! experience involved heavy and unorthodox use of said stack and most good things in life come out of conflict of one form or another, I am moved to write about my use of the shell stack.

Most of the shell programs that we use might be called "menu-based" utilities: ZFILER, VMENU/ZMANG, MENU, etc. Steve Cohen's ZPATCH and W use the shell stack in a slightly different but generally similar manner. However, there is an entirely different set of utilities, surprisingly little-known and infrequently-used, that use the shell stack in a very different manner and permit the user access to a feature of the shell stack referred to as named variables. I have been using these utilities since the very beginning of my enjoyment of ZCPR3 and it is about these that I now wish to write.

It will take me two articles to get it all down. I wish to devote this first to a discussion of my original involvement with the shell stack utilities and then a description of the special Shell Named Variable utilities upon which I relied for my project. With the next issue we will get into the project itself.

SHSET and CMD

I recall my first involvement with ZCPR3 way back in the spring of 1986. The particular love of my life at the time was spending night after sleepless night writing MEXPLUS scripts to log on and upload messages to computer bulletin boards. I would have 3 or 4 messages previously prepared to send to various individuals. The script was set up so that with a single command and appropriate parameters each message would be uploaded one by one to the appropriate individuals. It was quite lovely and satisfying to watch. It involved more than the usual amount of testing and trial and error: normally when you write a program you have only your own system to worry about, but here I had the BBS software as a variable as well.

I'm sure my neighbors were rudely awakened on more than one occasion when at 2 a.m. they would hear victorious shouts of joy emanating from my living room when I had finally gotten my script successfully honed to give me completely automated operation: logon, message upload to 4 different people, and finally logoff. What an achievement! Naturally it involved a great deal of (1) running the script, (2) going back to it for debugging, and as you can imagine, looping back and forth many times between these two procedures. Luckily, BBSs give you a certain amount of time to work on your own system while still connected, so though I may have antagonized a sysop or two for which I here, finally, offer apology, I was able to get much work done on my script in this manner.

I use the wonderful public domain VDE for all my editing and writing. The actual command lines, then, for which I was soon to realize ZCPR3 was ready-made, was to loop back and forth between: (1) VDE SENDMESS.MEX and (2) MEX. It's very tedious to have to close your editing session with VDE and manually type in 'MEX<ret>' (or if I was actually running the script from the command line, 'MEX SENDMESS PARM1 PARM2 PARM2...' each time. But how specifically to set up this loop?

Enter the dynamic duo, the wonderful SHSET and CMD. My hat is off to these tiny hard-working utilities for starting me down the adventurous and cosmic road to ZCPR3 those many moons ago. For that I can never be thankful enough.

I'm not sure how familiar Z users are with these and other similar utilities, so I'd like to take some time to describe their purpose and use. First, SHSET. SHSET defines the commands which follow it as the command sequence to be placed on the top of the shell stack. Anything appearing after SHSET on the command line will be placed in the command buffer and executed, repeating and looping from beginning to end over and over again until the shell stack is cleared or popped. In other words, the command "SHSET VDE" would run

```
VDE;VDE;VDE;VDE;VDE;VDE...
```

infinitely.

How to get out of this never-ending loop? It was for this purpose that we have CMD. CMD was built specifically for use as SHSET'S counterpart. It is a lazy utility. It does nothing—or rather it does nothing itself. It prompts the user for input. Whatever the user enters at its prompt gets executed. It's perfect for our purposes, because if we add it to the end of our SHSET command line, it will stop the sequence and allow us to experiment with one more shell stack-related utility and the one that

can get us out of this loop, SHCTRL.COM. We need only to enter "SHCTRL P" (or, if you are using Bruce Morgen's latest CMD13, simply CTL-C) at the CMD prompt. The purpose of SHCTRL.COM is, as the name states, to control the shell stack. The "P" parameter "pops" the shell stack, in other words clears whatever is on the first level—which in our case is, of course, "SHSET VDE." Hence the sequence is terminated and we are returned to the regular ZCPR3 prompt.

You say you can't think of uses for this? Have you ever been in a position where you are doing several different things on your system, performing a number of diverse procedures, but you realize you want to get a directory listing after each operation? Try "SHSET DIR CMD". You will first get your directory listing, then CMD will prompt you for a command. Enter whatever you like and do your operation. When you are finished DIR is automatically run once again. And so on. "SHSET DIR;CMD" is like having a menu shell such as ZFILER without ZFILER; you are constantly returned to a file display after each operation.

Or how about when you are writing and testing a ZEX script? I often find myself in the situation of wanting to loop back and forth between debugging and running it.

```
SHSET VDE SCRIPT.ZEX;ZEX SCRIPT;CMD
```

provides a wonderful degree of automation to your work, saving many keystrokes. I think the next time you are writing a ZEX script and try this you will wonder how you ever did without it.

Another example: the major programming work I do with Z-System is writing ARUNZ aliases, which are often quite complex. I generally need to test them many times before they work properly. Looping back and forth between editing them in my ALIAS.CMD and executing them is another operation tailor-made for SHSET and CMD:

```
SHSET ALIASNAME;VDE ALIAS.CMD;CMD
```

So as we have seen we don't have to limit SHSET's parameters to a single command such as DIR. The beauty of SHSET for my purposes in writing the MEXPLUS script was its ability to recycle through my entire command sequence, until it finally hits CMD. So for the MEXPLUS script I was working on,

```
SHSET MEX SENDMESS PARM1 PARM2 PARM3;VDE SENDMESS.MEX;CMD
```

worked like a charm for me and saved me lots of repetitive typing. (Don't try a command line of this length, by the way, without expanding your shell stack entries to 64 bytes each as I have done.) It has allowed me to stay connected to a BBS and quickly switch back and forth between executing and debugging my MexPlus scripts. Since MexPlus has for years been so much a part of my life I wouldn't do without this convenient little combo for the world. It was, as I have said, my introduction to ZCPR3 and I use it constantly, for many different purposes, to this day.

Shell Named Variables

There is an extraordinary feature of the ZCPR3 shell stack that has not received its due since the esteemed Dreas Nielsen stopped writing ZCPR3 programs, named shell variables. Those among us with experience with MS-DOS will be familiar with the concept of environment variables, and ZCPR3's named variables are quite similar. Mr. Nielsen's superb ZCPR3 programs GETVAR and RESOLVE are the two most common named variable utilities and we are all indebted to him for the immense amount of work he has done in the field. I would like to thank him for the tremendous inspiration he has given me over the last two years to work further with these and other tools to find many new and creative uses for them, for which which debt I hope to partially discharge through this article.

Throughout one's general ZCPR3 operations, but especially in writing ARUNZ aliases, one often wants some way of storing a string of characters to a variable, and then accessing the string later through the variable. The GETVAR/RESOLVE duo does this perfectly. The command:

```
GETVAR FOOD WHAT IS YOUR FAVORITE FOOD:
```

will display on the screen:

```
WHAT IS YOUR FAVORITE FOOD:
```

If in response you enter "Eggplant Parmesan," that string will then be stored into the variable 'food'. Later the command:

```
RESOLVE ECHO YOUR FAVORITE FOOD IS %FOOD
```

will display:

```
YOUR FAVORITE FOOD IS EGGPLANT PARMESAN
```

Many of you will remember Frank Gaude's excellent use of GETVAR and RESOLVE in some of the early ZCPR3 aliases he gave us in *Z-NEWS*, R.I.P.

The string "%FOOD" (note the "%") is what we call a named shell variable. Both the variable and its definition are stored in a file with the extension of VAR which you will find on the last directory of your path. By default it is called SH.VAR, though as we learn and explore more we will be creating and using other *.VAR files. It is to the wonders of these shell variable files that we will turn as we mine their depths and explore their mysteries in order to tackle many interesting projects...

Any number of individual variables may be stored in a VAR file, and they are available for use at any time. We may even have as a variable definition a multiple command line sequence! For instance, if we type:

```
GETVAR COMMAND ENTER A COMMAND LINE:
```

the computer will come back and say

```
ENTER A COMMAND LINE:
```

If we then respond with a multi-command sequence:

```
ECHO HERE IS A DIRECTORY;DIR;ECHO THAT WAS A DIRECTORY LISTING
```

that entire sequence, including the semicolon, will be stored into the variable named 'COMMAND'. If we then enter at the ZCPR3 prompt:

```
RESOLVE %COMMAND
```

this will "expand" or "resolve" into "ECHO HERE IS A DIRECTORY;DIR;ECHO THAT WAS A DIRECTORY LISTING" and run precisely that entire command line.

We may use GETVAR (or other utilities to be explored later) to store any number of variables into our *.VAR file. These variables are all then available to us through the courtesy of RESOLVE at any time—tomorrow, next week, or whenever, since they are saved to disk. It is this basic concept that makes the shell variables so powerful and such a joy to use.

SH

If we are the adventurous sort we may even take this concept one step further. Here we will find ourselves swimming in waters where only the bold (and with RAM or fast hard disks) dare to tread. Those who have had the experience of spending some time browsing in awe through our erstwhile bible, Richard Conn's *ZCPR3: The Manual*, turning each page with trembling hands as if it were a precious leaf, might have come across a utility named simply SH.COM. SH is a fascinating utility. The latest version, v2.0 by Dreas Neilsen, is a command editor/history shell in the fashion of EASE or HSH but is an entirely different type of program as well. It loads just like EASE or NHSH and stays resident. Where it is different, however, is that it then gives you a working environment in which it is possible to access any of your named shell variables directly from the command line without the need for any other program. I like to think of it as like having RESOLVE loaded permanently without RESOLVE being there at all. One might think of it as another way to set up something similar to aliases, or perhaps more like a key redefinition program (though feasible only from the command line and not from within

a word processor).

Suppose you have several single-letter strings in your *.VAR file defined as filenames:

```
S   =   samantha.ltr
T   =   thomas.doc
B   =   bertha.z80
F   =   frankie.txt
```

In the course of today's work session you were often editing these four files, but perhaps switching among several different editors, such as VDE, LZED and WS. With SH.COM permanently loaded, if you wanted to look at SAMANTHA.LTR with VDE, you could type on the ZCPR3 command line simply "VDE %S". Then perhaps when you were ready to do some fancy printing with it and you needed WS4's print commands, "WS %S" would take care of that. Similarly with the other four files.

It's almost as if you had used your key redefinition program to define your "S" key to that string, or perhaps written an alias named "S". There is no need with SH to include RESOLVE on the command line; with SH.COM permanently loaded all strings preceded by a "%" are automatically processed as shell variables.

SH20.COM is also a history shell in itself similar to EASE, providing for the recall of up to 20 previous command lines, so we have the convenience of a history shell plus the new type of utility that expands shell variables. It is a very interesting experience to spend an entire computer session with SH20 as one's history shell; I believe you will find it fascinating and will find many previously unthought-of uses for its shell variable expansion facility. Keep in mind, though: due to its slowness it is most useful on RAMdisks and fast hard disks.

SHVAR

You might be wondering at this point how we get a large number of these variables into our *.VAR file so that SH or RESOLVE may expand them. The only way we have learned up to this point is with the GETVAR utility, which is somewhat slow as it requires the setting up of a question-and-answer situation. There are two additional utilities Rick Conn provided for this task, however, which make our job much easier: SHVAR and SHDEFINE. The former is command-line driven and best where we are defining only one or two variables at a time, and the latter is interactive and good when there are a large number involved.

Syntax for SHVAR is straightforward. The first parameter is the name of the variable and the second its definition.

```
SHVAR S SAMANTHA.LTR
```

will define "S" as "SAMANTHA.LTR" and put those into our *.VAR file.

I have found SHVAR indispensable inside an alias where we are echoing to the console a long text message, itself containing a variable string. An example will explain.

```
IF ARC $1
    ECHO THE ARCHIVE BIT ON FILE $1 IS SET
ELSE
    ECHO THE ARCHIVE BIT ON FILE $1 IS CLEAR
FI
```

This is a fairly typical alias and usually we think of this as the standard way to conceptualize something that we often want to express in ZCPR3 (or any other system, for that matter.) This format could be generalized as follows:

```
IF CONDITION A EXISTS
    ECHO ''CONDITION A EXISTS''
ELSE
    ECHO ''CONDITION B EXISTS''
FI
```

But SHVAR gives us another possible way to conceptualize what we want. Notice above that we have TWO separate 'ECHO' command lines in our alias which to an outside eye might seem a waste. Generally since we have 200 characters in our multiple command line buffer we can afford to be slightly profligate. In the first example above, our echoed text is fairly short so we do not mind having two ECHO strings in the alias, each with the symbol "$1." In writing aliases, though, we must always write with an eye towards conciseness. We must particularly keep in mind that though our alias uses the symbol "$1" which would seem to take up only two spaces, in actuality when the alias expands it could fill up to 12 characters in the command buffer if our filename parameter is, for instance, SAMANTHA.LTR. Furthermore, we must keep in mind that **both** occurrences of "$1" above will be expanded to 11 characters in this case, whether SAMANTHA.LTR does or does not exist! So that's an additional 22 characters of our buffer we must account for.

If we want to get a little more fancy, however, we cannot afford to be so wasteful. For instance:

```
IF ARC $1
    ECHO WITH THIS ALIAS WE HAVE JUST DETERMINED
         THAT THE ARCHIVE BIT ON FILE $1 IS SET
ELSE
    ECHO WITH THIS ALIAS WE HAVE JUST DETERMINED
         THAT THE ARCHIVE BIT ON FILE $1 IS CLEAR
FI
```

I am making the ECHO'ed text especially long here to make my point, but it is not inconceivable that one could write an ECHO string of this length. This otherwise perfectly legitimate alias will overflow the 200-character limit.

Of course we notice that the two long ECHO strings are exactly similar except for a single word. Why not instead put SHVAR to use for us:

```
IF ARC $1
    SHVAR A SET
ELSE
    SHVAR A CLEAR
FI
RESOLVE ECHO WITH THIS ALIAS WE HAVE JUST DETERMINED
            THAT THE ARCHIVE BIT ON FILE $1 IS %A
```

Get it? Note the '%A' at the end. See how RESOLVE precedes and defines the ECHO command line. This ensures that when ECHO sees the "%A" at the end of the string, it will be processed as a string variable. I like this method of processing a conditional. It feels more "logically correct" since we really only need to define one word and we can simply make this a variable; there is no need to repeat an entire string for one word. This method, however, is somewhat slower, as RESOLVE does take time to load. In certain circumstances such as here where command line buffer space is at a premium, however, this technique is indispensable.

Other Examples Using SHVAR

I ran into a similar situation when writing my alias PLF, Process Library Files. In the section of this multi-element alias in which I wanted to print the file, I had as my second input parameter the name of a compressed file. However, the file had already been uncompressed, and it was this file's type that I needed in order to print it. There is really no way to get this name from the name of the compressed file as the middle letter remains unknown. I decided to make a list of several commonly used filetypes and then use SHVAR to help me expand the possibilities.

To save space I renamed SHVAR.COM to S.COM. I have 'ELSE' permanently renamed to 'L' on my system. Finally I created the following ARUNZ alias (remember, '$tt2' means the fileTYPE and '$tt2' the fileNAME of the second parameter.)

```
plf7  if eq $tt2 zzz;s p  ;l      [store null to 'p']
      if eq $tt2 u?d;s p upd;l    [store 'upd' to 'p']
      if eq $tt2 z?0;s p z80;l    [store 'z80' to 'p']
      if eq $tt2 i?f;s p inf;l    ...etc...
      if eq $tt2 h?s;s p his;l
      if eq $tt2 n?t;s p not;l
      if eq $tt2 d?c;s p doc;zif
      resolve print $tn2.%p
```

As always, note the "%p" at the very end, our nugget of gold. Do you see how it works? The middle letter of the compressed filetype could be either "z" if crunched or "q" if squeezed, so I represent that with a ?". We set up "p" as the name of the string variable and define it with SHVAR ("S") according to the results of the IF EQ test on the filetype of the second parameter. Then, once we have its definition, in the last line we print it, with RESOLVE picking out the "%p" at the very end of the line and expanding it to the name of the filetype that we have defined with SHVAR.

I can't imagine any other way to do this in a single alias. I was very pleased that I was able to rely on such a splendid tool as SHVAR. I find this concept of "string variable" to be very handy in many situations.

SHDEFINE

So much for SHVAR, which being command-line driven is excellent for use in aliases. Sometimes, though, we will want to define a large number of variables in one setting. Enter SHDEFINE. This utility is menu-driven. The crucial command for our purposes is "E" for Edit, which allows us to interactively define as many variables as we wish. It is this almost never-used utility upon which I relied heavily in working with my project. We will use it later.

SHFILE

Now to the last utility in this series, SHFILE. If you use SHFILE once a year you're doing well, but when you have a need for it nothing else will do. I mentioned above that though the default name for the file in which our named variables are stored is SH.VAR, we can also create other named variable files. It is for this task that SHFILE was created.

Users of key redefinition programs such as NUKEY know that they can store a large number of "sets" of key definitions, one set to a file. We may want one set of key redefinitions when we are writing computer documentation, one for when we are writing letters to friends, and yet a third for our Ph.D thesis in botany. Each one is simply loaded as necessary with the key redefiner "loader," such as NUKEY.COM. We may want the letter "s" to produce "software" in the first circumstance, "Sandy O'Brien" in the second, and "serrate-leaved" in the last.

The concept here with SHFILE and named variables is similar. We may have several different applications for which we wish to use the same named variable. Instead of "loading" a new set of definitions with a "loader," however, we use SHFILE. The particular parameter provided to SHFILE informs ZCPR3 which will be the "current" named variable file, i.e. which one will be searched and used by whatever variable-expanding utility we are using, whether it be RESOLVE, SH, SHVAR, etc. If SHFILE is not specifically invoked all variable expansion is done to the file named SH.VAR, and only this single file is ever used.

Here's a good example of where we might use the combined power of SHFILE and SHDEFINE. Recent versions of ARUNZ allow for hooks into DateStamper, whereby the symbol $Dm will return the current month, $Dd the date and $dy the year, in two-digit numerical format. However, recently I wanted to create an alias that would tell me today's date with the **name** of the month and not just its numerical equivalent. Using **only** ARUNZ was out of the question as we would very quickly overflow the command line buffer as we can see (as always, I have renamed 'ELSE' to 'L'):

```
IF EQ $DM 01;ECHO TODAY IS JANUARY $DD, 19$DY
   L
IF EQ $DM 02;ECHO TODAY IS FEBRUARY $DD, 19$DY
   L
IF EQ $DM 03;ECHO TODAY IS MARCH $DD, 19$DY
   L
IF EQ $DM 04;ECHO TODAY IS APRIL $DD, 19$DY
    .
    .
    .
```

and we are already at our 200-character limit.

For fun we could see if we could do it with SHVAR.COM, renamed to S.COM, where our variable named "D" would be expanded to the name of the month:

```
IF EQ $DM 01;S D JANUARY
   L
IF EQ $DM 02;S D FEBRUARY
   L
IF EQ $DM 03;S D MARCH
   L
IF EQ $DM 04;S D APRIL
   L
IF EQ $DM 05;S D MAY
   L
IF EQ $DM 06;S D JUNE
   L
IF EQ $DM 07;S D JULY
    .
    .
    .
RESOLVE ECHO TODAY IS %D $DM, 19$DY
```

but again we are past 200 characters. SHDEFINE and SHFILE to the rescue.

Since this is a fairly specific use of the named variables, we don't want to waste space in our default SH.VAR as we want to keep it "clean" and leave it for other purposes. We are going to make the month numbers our actual variables and the month names our variable definition, the string to which it will be defined or expanded. If we define "01" to be "January" inside the default SH.VAR then we will not at some future point be able to define "01" to anything else, such as if we have an itemized list and we want "01" to represent the first item on the list, etc. It is more convenient to, as it were, load different sets of key redefinitions and keep strings for different applications distinct and separate.

We can use SHDEFINE to define variables within a file other than SH.VAR simply by providing its name as a parameter on the command line, so we enter: "SHDEFINE MONTHS". Then using the E)dit command we simply define "01" as "January," "02" as "February," "03" as "March," etc. "X" will exit and updates MONTHS.VAR on disk.

(Note, by the way, that the bottom part of the SHDEFINE help file:

```
Exit:   X. Exit and Update SH.VAR on Disk
        Q. Quit without Updating SH.VAR
```

is in error. It will say "SH.VAR" even when that particular VAR file is NOT the current named variable file or the one being edited. This is the result of the programmer Rick Conn incorrectly hard-coding the text "SH.VAR" into SHDEFINE.COM's help routine where it should instead be reading the name of the file being edited.)

Then we write an ALIAS.CMD alias—we'll name it TODAY—using SHFILE as follows:

```
TODAY   shfile months
        resolve echo today is %$dm $dd, 19$dy
        shfile sh
```

It is here that we use SHFILE to define "MONTHS.VAR" as the current variable file, so that RESOLVE will know that it is this file that it must search. Note, of course, the "%" preceding the "$dm", which indicates to RESOLVE to process what follows as a named variable. ARUNZ processes '$dm' to the correct month, e.g. '11', so we then have "%11" as our named variable. RESOLVE looks inside MONTHS.VAR and finds that the variable named '11' is defined as the string 'NOVEMBER.' It sends that to ECHO which then miraculously displays to the user:

```
TODAY IS NOVEMBER 12, 1988
```

The last line in the alias returns the system to SH.VAR, the default, as the current named variable file.

I find this use of the VAR files as a "data file" to be very creatively satisfying. Those who know how to write an assembly language COM file to do the same thing (DATE.COM, for instance) would use a routine inside their source code almost exactly as what we have in MONTHS.VAR.

Named variables are the poor man's assembly language.

The final example of SHDEFINE and SETFILE I will describe is a routine I used when I wrote a demo graphics program for MexPlus. It is a demo script for users of PC-Pursuit, the national satellite BBS hook-up for modem star travelers. It will particularly illustrate the importance of SHFILE and how we often want to assign different definitions to the same variable depending on our application, and that therefore we should keep separate named variable definition files for different purposes.

PC-Pursuit allows access to some 40 or 45 "city nodes," metropolitan areas users may access via modem. These nodes are identified by either a city name or a five-letter code. The script I wrote displays to the user both the name and the associated code of the last city called. Since it was essential to me that this information be retained even if the user exits completely from MexPlus, I opted to store this information in memory, and the logical place was in one of the ZCPR3 registers. But of course this presents a challenge since one cannot store (and subsequently access) a string of text characters but only a single numerical value in a byte of memory. Here we have SHDEFINE and SHFILE, and—as always—ARUNZ to the rescue.

I assigned a numerical value to each of the 40 or 45 PC-Pursuit nodes according to their alphabetical order, and stored this value in the MexPlus script. I then had the script poke ZCPR3 register C (3 bytes above the "official" register 9) with this value. Then I put SHDEFINE to work. I created PCPCITY.VAR with the command "SHDEFINE PCPCITY" and defined '01' as 'Atlanta,' '02 as 'Boston,' '03' as 'Chicago,' '04' as 'Cleveland,' etc. until all 40 city names were assigned.

I then created PCPCODE.VAR and used SHDEFINE to assign '01' to Atlanta's code, GAATL; '02' to Boston's code, 'MABOS'; '03' to 'ILCHI', the code for Chicago; '04' to 'OHCLE,' Cleveland's code, and so forth. PCPCITY.VAR and PCPCODE.VAR were now both in place and ready to be accessed by RESOLVE or SH.

For demonstration purposes in this article I created two aliases slightly different from that which I used to begin my script but which perfectly illustrate the concept. The aliases, which display the code and city name respectively of the just-logged PC-Pursuit node, are as follows:

```
PCPCITY:
    shfile pcpcity
    resolve echo last pcp city name was %$rt0b
PCPCODE:
    shfile pcpcode
    resolve echo last pcp city code was %$rt0b
```

In order to understand what's going on, let's take a look at that last string in each alias. Remember that Jay Sage has enhanced and expanded the symbols that represent the ZCPR3 registers. To represent "the value in register B displayed as two hex digits" we now have the symbol "$rt0b", and it is this that we are here using. The "%" in front is simply our old friend the symbol that tells RESOLVE or SH to treat what follows as a named variable. Suppose I had just logged off from the Lillipute Z-Node in Chicago. When the PCPCITY alias runs, PCPCITY.VAR becomes the current variable file and RESOLVE sees:

```
ECHO LAST PCP CITY CODE WAS %03
```

RESOLVE knows that '03' is the name of a variable it is being asked to expand. It then looks to PCPCITY.VAR, finds that inside that file the variable '03' is defined as 'CHICAGO', and ECHO displays:

```
LAST PCP CITY NAME WAS CHICAGO
```

When we run PCPCODE, SHFILE changes the current variable file to PCPCODE.VAR, and it is this file that RESOLVE then searches for the definition of '03'. This time we will be displayed:

```
LAST PCP CITY CODE WAS ILCHI
```

Notice the way that we are being very creative in transforming numeric values in memory into strings of text, here names of variables. The named variable feature allows us this flexibility and power. I'm sure you can think of many more uses for this wonderful feature of ZCPR3.

I think now we have enough background in the named variables and their associated utilities. Next column I will describe my big project about which I wrote at the beginning that inspired me to set pen to paper in the first place. I will whet your appetite here by saying that it not only uses SHDEFINE to enter the variables into the *.VAR file and SHFILE to define this file as current, but also another little-known ZCPR3 utility (SETFILE) that uses the "system file" feature that I will describe. To top it all off it is all snugly wrapped inside a ZEX file. Be sure to bring a good appetite next issue for this sumptuous ZCPR3 feast.

Write with any comments. Z you next time... ■

The Best-Kept Secret in tne Modem Industry!

Every active modem user would like to have a high-quality 2400 bps modem. After all, phone calls aren't free—and the sooner you can get your on-line business over with, the sooner you can have your computer back for other things. Buying a high-speed modem, however, is not a comfortable decision. On one side are the good-but-overpriced "name-brand" products of Hayes, Racal-Vadic, Prometheus, USR, Okidata, etc., on the other, cheap-but-how-good-is-it-and-how-long-will-it-last Taiwanese and Korean imports, usually sold by mail-order houses with questionable reputations for after-sale service. It's enough to keep you at 1200 or even 300 bps forever!

Our EMEX 2400 external modem represents a sensible alternative to confronting this dilemma. The EMEX in made in the U.S.A. by Incomm of Wheeling, Illinois, one of the most respected manufacturers of professional data communications equipment. Incomm does not advertise its products, preferring to sell them through a select group of technically-knowledgeable distributors. We get our EMEXs from one of these companies, a $100 million+/year organization with branches all over the country and a fully-equipped service department, headquartered right here in suburban Philadelphia.

Not only is the EMEX probably the only U.S.-built modem without an inflated price tag, it's also the only modem we know of that's "speed-upgradeable." That means that when the industry settles on a 4800 or 9600 bps standard, your modem can be factory-retrofitted to the higher speed (and MNP error correction as well) for a modest, under-$100 service charge. What's more, the EMEX is, to our knowledge, the only modem sold in the U.S. that carries a FIVE YEAR WARRANTY—not 90 days or a year. No wonder the EMEX is recommended for costly, multi-port UNIX systems as well as for "little" CP/M and Z-System computers, Big Bluish PCs, and other single-user systems.

The EMEX modem is not the cheapest on the market—after all, you can get a no-name 2400 bps unit for under $150. But, considering its exceptional quality and the value added by its unique up- gradeability and unsurpassed warranty, it's a true bargain at only $225 plus shipping and handling (usually $15 or less—the modem itself weighs less than 2 pounds, but the heavy-duty 3-prong power supply is hefty). The EMEX is truly Hayes-compatible—its command set is very close to that of the Hayes 1200 Smart-modem(tm), much closer than 2400 bps modems from USR or even Hayes itself. There are only three DIP switches to set, every thing else can be set up in terminal mode and saved to non-volatile memory. The EMEX has been tested with all the popular PD, shareware, and commercial communications software packages, including ProComm(tm), BOYAN, MEX(tm), CrossTalk(tm), etc. It is also known to work with RBBS-PC(tm), FIDO(tm), WildCat(tm), BYE and most other remote access system software.

To order your EMEX 2400, call North American One-Eighty Group at 215-443-9031 with your MasterCard or VISA handy—or write us at P.O. Box #2781, Warminster, PA 18974. If writing, be sure to include your name as embossed on your credit card, along with the card number and its expiration date. If paying by check, please allow $15 for shipping and handling—if that comes out to be five or more dollars less than $15, we'll mail you a check for the difference. If you have questions, we can be reached via modem on the "DHN*" system in Philadelphia (215-623-4040) or "Lillipute Z-Node" in Chicago (312-649-1730 or 312-664-1730), just leave a message for "Bruce Morgen" and we'll usually reply within 48 hours. Thanks for your time!

REL-Style Assembly Language for CP/M and Z-System

Part 2: Getting Started

by Bruce Morgan

Once you've acquired a compatible assembler and linker as described in the Part I, you'll be ready to break out you trusty ASCII editor and key in some simple REL-style assembly language programs. Note that the editor or word processor you use must be capable of producing a pure ASCII file, also known as "straight ASCII" or "flat ASCII." WordStar® and NewWord® can do this in their "non-document" mode, as can the standard CP/M line editor, ED.COM. Free-for-the-downloading programs like VDE266 by Eric Meyer and VDO25 by James Whorton will also do the trick.

We generally use LZED (the Little Z-System Editor) for short programs—as a matter of fact, when a source file becomes too big for LZED to load (about 40K), we generally figure it's time to break it up into LZEDable pieces. Other suitable commercial editors include PMate (no longer available, but the favorite of Jay Sage and Richard Jacobson, two very demanding and trusty users), Perfect Writer® (a derivative of Mark of the Unicorn's famous MINCE editor that Bridger Mitchell uses), and VEDIT.

Since we'll be presenting our examples in Zilog dialect, you might also want to equip yourself with a Zilog-to-Intel translation tool like Irv Hoff's XZI.COM (from XIZI-x.LBR) if you're using RMAC® or M80® in Intel mode.

Segments and the EXTRN Directive

REL-style assemblers and linkers can work with several relocation bases in a single module or program. Informally called segments, they are controlled with assembler directives (pseudo-ops) like CSEG, DSEG, ASEG and COMMON. This means that within a single source file, the assembler and linker can keep track of a few distinct sets of instructions and data.

This is an extremely powerful and useful capability, and one that advanced programmers can take spectacular advantage of. For now, we will be working strictly with the code segment, controlled by the CSEG directive. CSEG is the default mode of all the REL-style assemblers mentioned in Part I, so including the explicit directive in your source file is strictly optional. In CSEG mode, it is the linker rather than the assembler that determines the absolute origin of the program, so the first CP/Mish habit you'll have to discard is the inclusion of an "ORG 100H" directive—this can result in 256 bytes of useless filler at the start of your program after it has been linked.

Anyone with a modicum of experience with assembly language is familiar with the advantages of structuring repeatedly-used multi-instruction functions as subroutines. After all, why have the same code more than once if it can be repeatedly used via CALL instructions?

In a sizable conventional ASM-style assembly language program this very good practice usually results in dozens of subroutines cluttering up the source file, sometimes making it many times larger than the mainline code that defines the program's actual function. For example, a simple program to send a message to the screen could have this as its mainline code:

```
        CALL    PRINT
        DB      BELL    ; ding...
        DB      'You goofed!'
        DB      0       ; terminate
        RET
```

In the ASM style, you'd have to code or copy the PRINT subroutine into the source file, with MAC you could use the MACLIB directive to read in an assembly language LIB file that included the routine. The ASM method is just too darned much typing, and the MAC alternative will generally introduce a bunch of unCALLed-for code into the final COMfile.

REL-style coding using SYSLIB makes the entire program source only slightly larger than the mainline code:

```
EXTRN   PRINT

BELL    EQU     07H

        CALL    PRINT
        DB      BELL    ; ding...
        DB      'You goofed!'
        DB      0       ; terminate
        RET
```

The EXTRN directive tells the assembler that the PRINT routine isn't in the source it's currently working on, so the value of the label PRINT is left for the linker to resolve. If you use your editor to make the above code into DING.Z80, you could have M80 assemble it into DING.REL with the command line:

```
A>m80 =ding.z80/z
```

The "/z" trailing parameter is optional, since the code at this point is acceptable as either Intel or (rather loose) Zilog dialect. With RMAC, name the file DING.ASM and use this command:

```
A>rmac ding $-s pz
```

Either way, you'll shortly be in possession of DING.REL, a Microsoft-format relocatable object file. To make it into a CP/M-compatible program, you need to use your linker and SYSLIB.REL, the Microsoft-format library of relocatable subroutines that includes one called PRINT. SYSLIB's PRINT is a rather elaborate version of the ILPRT (In Line PRinT) routine commonly found in many CP/M programs. To build your DING.COM with the L80 linker, make sure DING.REL and SYSLIB.REL are on the same drive and in the same user area and type:

```
A>l80 /p:100,ding,syslib/s,ding/n/e
```

With RMAC's companion, LINK.COM, this is the command line to use:

```
A>link ding,syslib[snr]
```

In either case, the "s" parameter tells the linker to treat SYSLIB.REL as a library, linking in only the required routines rather than SYSLIB in its 24K entirety. The additional "nr" in the LINK command line suppresses creation of a symbol file, which is something you won't want to do during your own program development work—this file (e.g. DING.SYM) is used by a symbolic debugger like SID, ZSID, Z8E or WADE to allow use of symbol names as well as numeric addresses in disassembly and breakpointing operations. To make L80 generate a symbol file, use a trailing "/y" (no spaces) parameter.

What's The Big Deal?

Although REL-style's ability to relieve programming drudgery should be somewhat apparent, even with as simple an example as DING, what makes it a truly time-saving technique is the easy, free availability of extensive subroutine libraries like SYSLIB. Take almost any programming project for CP/M or Z-System/Bgii and SYSLIB (along with its more specialized brethren, Z3LIB, VLIB and Z33LIB) can shorten development time and, in many cases, significantly improve the quality of the finished product.

One of the most onerous tasks faced by CP/M assembly language programmers is "simple" command line argument parsing. Entire programs have gone unwritten because the programmer did not want to facing coding YAP (Yet Another Parser). With SYSLIB, such excuses pretty much evaporate. There's a whole family of routines that the SYSLIB help files call "parsing aids" plus such parsing-related goodies as numeric evaluators, complete drive/user/filename-to-FCB (CP/M File Control Block) translators, and specialty parsers for the Z-System environment.

For an example, suppose your program's design calls for getting the binary value of the number in the third command line token. ARGV, SYSLIB's UNIX-like argument parser could find the token, and the EVAL10 routine could convert the ASCII decimal number found there to a register value like so:

```
extrn   argv,eval10

tbuff   equ     80h
bell    equ     07h

        ld      hl,tbuff+2
        ld      de,argtbl
        ld      a,0ffh
        call    argv
        jp      z,argsok
        call    print
        db      bell
        db      'Too many args.',0
        rst     0       ; abort
argsok: ld      hl,(arg3)
        call    eval10
; Value is in DE now, go use it,
; test its validity, etc....

; ARGV's argument table
argtbl: db      3
arg1:   ds      2
arg2:   ds      2
arg3:   ds      2
```

This is a simplified example and assumes that the command line tail as parsed by command processor has not been corrupted. On entry, the HL register points to 82h, the first address where a command token might be found, DE points to the program's argument table data structure, and A contains a non-zero value (we could have stolen the 82h value in the L register and saved a byte) to signal ARGV not to null-terminate the arguments.

ARGV is then called, it does all the grunt work and fills in the defined spaces in the table with the addresses of the tokens. the defined byte at the head of the table is the maximum number of arguments ARGV is allowed to handle, it will return with the Z flag set if that number is not exceeded.

With the token's address loaded into HL, the EVAL10 routine is called. This routine returns (in the DE register) the binary equivalent of the ASCII decimal string pointed to by the HL register. When EVAL10 returns, HL is pointing at the first non-decimal character in the string, so an invalid number can be trapped by comparing HL's contents on return from EVAL10 with its entry value.

In the next episode of the REL-style saga, we'll to some file I/O, along with some Z-System magic with Z3LIB—heck, we might even start to build something useful while we're at it. We'll also delve into some linkage lore like the mysterious "$MEMRY" label and provide some handy patch points for RMAC and PROLINK, so stay tuned. ■

Real Computing

(Continued from page 28)

Next Time

In the next installment of this column, we'll discuss the NS32081 floating point unit. National's FPU is tightly integrated with the CPU and makes floating point painless and easy. There may be some other announcements to make as well. Until then, liberate your dreams! ■

Richard has moved since issue #35, and his new address and BBS number are as follows:

Richard Rodman
8329 Ivy Glen Ct.
Manassas, VA 22110
BBS (703) 330-9049

Advanced CP/M

by Bridger Mitchell

BackGrounder ii Update

BackGrounder ii is like no other CP/M program—it simply feels different. A touch of the "suspend" key and you pop into the background command processor, a touch of a user-defined macro key and you can switch to a second program, literally in mid-sentence. The built-in calculator, notepad, screen-dump, and cut-and-paste function turn out to be extremely handy desk accessories, especially because results on one screen can be exported to another task. But the magic of it all—black magic, perhaps—is the feeling that comes over you when you first experience the screen flashing back, cursor in place, with *no* trace of having been away!

BGii and the Z-System stand as twin pinnacles of advanced CP/M operating systems. At a conceptual level, they are orthogonal. By providing memory buffers for the command processor and applications and supporting conditional execution, ZCPR 3.4 allows tasks to *communicate sequentially*. By making the BDOS and command-processor recursive, BackGrounder ii creates two-way *communication between simultaneous tasks*, under user control. When combined—BGii running in a ZCPR 3.4 system—they elevate 8-bit computing into another dimension. The results are awesome.

Bringing BGii fully up-to-date to support the latest ZCPR version 3.4 has been a largely enjoyable task. I had put it off more than once, wanting to finish up DosDisk and then Z3PLUS.

When I finally returned to the BGii code I was pleasantly surprised to uncover several new coding shortcuts. They enabled me to squeeze in almost all of the "Z34" features and add some new conveniences, including enabling the user to rename the built-in BGii commands. Expert testing by Cam Cotrill and Jay Sage greatly firmed up several soft spots. It's now the production version and licensed users can order an update at low cost.

Environmental Programming

A customer of long-standing called the other night, as I was drafting this column. He enthused about BackGrounder ii, but then noted that "it sometimes finds bugs in *other* programs!"

Alas, bugs are always with us, even when we think we've got our own code pretty solid! This column is going to be about writing code that is respectful of the environment in which it is running. The sage (Sage?) advice collected here, and culled from the programming experience of many old hands, surely won't eliminate bugs. But it will greatly increase the chances of your programs living more harmoniously with a wide variety of CP/M systems.

Make a Good Start

The command processor starts your program by *calling* it. This means that you can speed up the flow of jobs by *returning* to the CCP when your program terminates, instead of causing a warm boot that reloads the CCP. To do this, you must *save the stack pointer* and stay clear of the CCP in the 2K of memory just below the BDOS.

Use a local stack for all but the simplest programs. The command processor's stack may not be deep enough for your functions, BIOS calls, and interrupts. And that stack could be in the TPA, part of the CCP that may be overwritten by your program or data.

Know the Territory

A shockingly large number of programs assume that they will always be run only in the environment for which they were written. Drop them into a different world and they almost always injure their host. So, please, join the environmentalists and take the responsible programmer's oath: *Do No Harm!* Survey the territory before plunging ahead, and pose these questions:

Is our host a Z80? An HD64180? A Z280? That determines which opcodes can we safely use. Is our host running CP/M Plus? or ZSDOS? What system calls are available? Is DateStamper running?

Is one of the drives set to MS-DOS format under DosDisk? If so, we must not make assumptions about internal data in the file control blocks on that drive or about the structure of the disk directory.

Is the host running a Z-System? With an extended environment? If so, we should allow for possible non-standard sized BDOS and CCP modules and get their addresses from the environment. If it is *not* a Z-System, we must avoid any references to Z-environment parameters; if we are a Z-tool, put out a short message of requirements and quit.

Finally, if we should need to know, can we determine what BIOS and type of machine our host is?

Figure 1 is a routine called TERRITORY that does these checks. It should be called at the very beginning of a program. If the host system has a Z-System command processor (ZCPR 3.3 or later, or BackGrounder ii) the program will begin with the HL register containing the address of the Z-System external environment.

TERRITORY first checks for a Z80-compatible processor, and then uses obscure differences in register operations to identify HD64180 and Z280 processors. The system addresses (BIOS, BDOS, and CCP) are determined from a Z-System extended external environment, if there is one, so that non-standard BDOS and CCP modules can be used correctly.

The BIOS check demonstrates how to detect an NZ-COM system and find the address of the original CBIOS. Several systems have bios-specific references to such things as function-key tables, foreign disk parameter blocks, and extended BIOS functions that cannot be located from the address at 0001h when NZ-COM is running.

Bridger Mitchell is a co-founder of Plu*Perfect Systems. He's the author of the widely used DateStamper (an automatic, portable file time stamping system for CP/M 2.2); Backgrounder (for Kaypros); BackGrounder ii, a windowing task-switching system for Z80 CP/M 2.2 systems; JetFind, a high-speed string-search utility; DosDisk, an MS-DOS disk emulator that lets CP/M systems use pc disks without file copying; and most recently Z3PLUS, the ZCPR version 3.4 system for CP/M Plus computers.

Bridger can be reached at Plu*Perfect Systems, 410 23rd St., Santa Monica CA 90402, or at (213)-393-6105 (evenings).

Figure 1. Determine characteristics of CP/M host's environment

```
bdos    equ     5

; Call this routine immediately.
; Enter with HL = value from command processor.
;
TERRITORY:
;
; Test for z80-compatible cpu.
;
        sub     a               ; sets even parity in 8080
        jp      po,ck_180
        ld      c,9             ; announce Z80 requirement
        ld      de,notz80msg
        call    bdos
        rst     0               ; ..and exit to warmboot
;
; Test for HD64180/Z180
;
ck_180:
        ld      bc,101h         ; prepare to multipy B=1 x C=1
        db      0EDh,04Ch       ; MLT BC opcode
        dec     b               ; if Z80, B will be unchanged
        jr      z,ck_280        ;
        ld      a,c             ; 180 leaves 16-bit result (1) in BC
        ld      (z180flag),a
;
; Test for Z280

ck_280: ld      a,10            ; Z280 doesn't use refresh register
        ld      r,a             ; load it
        ld      c,a             ; save it
        ld      b,a             ; cause some refreshes
loop:   djnz    loop
        ld      a,r             ; if value hasn't changed
        cp      c
        jr      nz,ck_rest
        ld      (z280flag),a    ; ..it's a 280
;
ck_rest:
        push    hl              ; save possible env address from ccp
        call    ck_dos          ; check type of BDOS
        pop     hl
        call    ck_z3           ; check for Z-System
        call    ck_dosdisk      ; check for DosDisk
        call    ck_bios         ; check type of BIOS
        call    ck_bg           ; check for BackGrounder ii
        ret
;
;
; Check for BDOS version
;
ck_dos: ld      c,12            ; get CP/M version number
        ld      e,'D'           ; with DateStamper id request
        call    bdos
        cp      30h
        jr      c,ck_ds
        ld      (cpm3flag),a    ; set flag if CP/M 3
        ret
;
; Check for DateStamper
;
ck_ds:
        cp      22h
        jr      nz,ck_xdos      ; .. not CP/M 2.2
        ld      a,h
        cp      'D'
        jr      nz,ck_xdos      ; ..no DateStamper
        ld      (dsflag),a      ; set flag
        ld      (dsclock),de    ; and save clock pointer
;
; Check extended dos version
;
ck_xdos:ld      c,48            ; use extended version number call
```

You can use the flags and addresses established by TERRITORY for your own requirements. You might also want to make a version of it into a simple diagnostic tool that prints out messages identifying exactly what the host system consists of.

Identify Yourself

Unless yours should be a silent program, announce yourself to the user with an appropriate message that includes a version number. All programs change, get updated, and gain features. You and other users need to be able to identify which model they're driving. Most Z-System tools use a standard format, which is well worth adopting for other programs:

```
A>PROGNAME Vers. 1.5-terse functional description
```

If you are the silent type, include sufficient version identification in the data area so that a debugger can be used to inspect the program. Alternatively, include the information in a help screen. Z-System tools use the standard "double-slash" command line to request help, another worthwhile convention:

```
A>PROGNAME //
```

Protect the Environment

Save the current drive and user number, so you can restore them on exit.

Explicitly *allocate memory* and check to prevent overflowing the available transient program area. The TPA is always the memory from 100h to the value that is stored at 0006, less 1 byte.

If there are no RSX's in memory, that value is the entry address of the BDOS and is the target of the jump instruction at 0005. This is the most common case; in CP/M 2.2 the CCP will occupy 6 + 800 hex bytes below that. But if an RSX has been loaded, its address will be at 0006. In that case the CCP will already be protected, and you can use all of the TPA and still return to the CCP.

So, if no RSX is loaded, if you intend to return to the CCP, and if you are not running under CP/M Plus, allow 2K of space below the BDOS. Figure 2 gives a routine that makes this calculation. It calculates the largest usable memory that will still preserve the CCP and returns the address of the first byte beyond that.

Applications have the right to have their register values treated systematically when they call on the operating system for services. CP/M is an 8080-based operating system, and from the beginning it put programmers on notice that it would not preserve the user's registers. That was logical, as the OS needed most of them for returning values.

The introduction of the Z80 and subsequent 8080-compatible CPUs led to more compact and more efficient BIOSes.

Unfortunately, more than one BIOS writer began using the additional *Z80 registers* without preserving their values for the user. The consequences have been erratic havoc—programs test out flawlessly on a variety of systems, then fail to start, or die mysteriously on another machine.

The environmentally-conscious rule here is: if your code will become part of the operating system—the BDOS, BIOS or an RSX extension—save and restore all Z80 registers you use. Why? Simply because it's a far greater burden on an application to preserve IX, IY, AF', BC', DE', and HL' in order to run on an arbitrary system, than it is for the system programmer to protect exactly those registers he needs to use.

"If your code will become part of the operating system, save and restore all registers you use!

The extreme case of environmental wantonness is a ROM-based-based BIOS for the early Osborne Executive, which used alternate Z80 registers for an interrupt service routine, without preserving them! A moment's thought should persuade you that there is *no way* an application could *ever* use those registers and run on that machine. Was the system designer so naive as to think that only 8080-code would ever be run on an Executive?

Expect the Worst

Test for all disk errors. Proceeding after one error (a full disk, a non-existent file, . . .) can wreak disaster.

Have a recovery strategy. Tell the user enough about the problem that he can alter the environment and try again. Give him a choice (insert a disk, restart the program, exit, . . .).

Limit the Damage

Before writing a file, check the disk space remaining. If there isn't enough room, give the user the chance to delete something, or to insert a new disk.

If a write error occurs, try to close the file to save as much data as possible. Offer the user a second chance, by changing disks or drives. Clean up file fragments after recovering.

Before printing, test to see if the printer is ready. If it isn't, ask the user to make it ready before retesting. Otherwise, if you start sending characters, the computer will hang and you can't inform the user what's wrong. A short routine to do this, derived

```
        call    bdos
        ld      (dosvers),hl    ; save version number and type
        ret

;
; Check for Z-System.
; Enter with HL = value from command processor.  If ZCPR 3.3
; or BackGrounder ii, HL -> external environment
;
;
ck_z3:  push    hl              ; save possible ENV address
        inc     hl              ; Offset to 'Z3ENV'
        inc     hl
        inc     hl
        ld      b,Z3ENVLEN
        ld      de,z3envsig
        call    match
        pop     de              ; recover de = ENV address
        jr      nz,set_std
        ld      hl,1Bh          ; Offset to self-reference address
        add     hl,de
        ld      a,(hl)          ; Check low byte
        cp      e
        jr      nz,set_std
        inc     hl
        ld      a,(hl)          ; Check high byte
        cp      d
        jr      nz,set_std
;
        ld      (zsysflag),a    ; set flag
        ld      (z3env),de      ; save environment address
        ld      hl,08h          ; get env. type
        add     hl,de
        ld      a,(hl)
        and     80h             ; test for extended type (>= 80h)
        ld      (extenvflag),a  ; and save result
        jr      z,set_std
;
; Set system addresses for a Z-System with an extended environment

        ld      hl,45h          ; -> bios address in environment
        call    addderef
        ld      (biosbase),hl
        ld      hl,42h
        call    addderef
        ld      (bdosbase),hl
        ld      hl,3Fh
        call    addderef
        ld      (ccpbase),hl
        ret

;
; Set system addresses for a standard system.
;
set_std:
        ld      hl,(0001h)
        ld      l,0
        ld      (biosbase),hl
        ld      de,-0E00h
        add     hl,de
        ld      (bdosbase),hl
        ld      de,-800h
        add     hl,de
        ld      (ccpbase),hl
        ld      a,(cpm3flag)    ; if not CP/M Plus
        or      a,a             ; ..all done
        ret     z
;
        ld      c,49            ; for CP/M Plus, w/o extended environment
        ld      de,getscbpb     ; get system control block address
        call    bdos
        ld      l,98h           ; offset to address of resident bdos
        call    deref           ; dereference pointer
        ld      l,0
        ld      (bdosbase),hl
        ld      hl,100h
```

```
        ld      (ccpbase),hl
        ret
;
addderef:
        add     hl,de           ; offset pointer by DE
deref:  ld      a,(hl)          ; dereference HL pointer
        inc     hl
        ld      h,(hl)
        ld      l,a
        ret
;
match:  ld      a,(de)          ; match`B bytes at DE, HL
        cp      (hl)
        inc     hl
        inc     de
        ret     nz
        djnz    match
        ret

ck_dosdisk:
        ld      c,113           ; get DosDisk id
        call    bdos
        cp      0FDh
        ret     nz              ; ..if not DosDisk, quit
        ld      (dosdiskflag),hl
              ; set flag (L) and drive with MS-DOS format (H)
        ret


ck_bios:
        ld      hl,(0001h)      ; -> normal CBIOS warmboot address
        ld      l,90            ; -> NZ-COM id in NZ-COM pseudobios
        ld      de,nzname       ; if NZ-COM id is exactly there
        ld      b,NZNAMELEN
        call    match
        jr      nz,ck_turbo
        ld      hl,(z3env)      ; ..get CBIOS (page) addr from
        inc     hl              ; ..z3env+2
        inc     hl
        ld      h,(hl)          ; ..get page
        ld      l,0
        ld      (biosbase),hl   ; ..and save correct ptr to CBIOS
;
; Check for Kaypro TurboRom
;
ck_turbo:
        ld      hl,0FFF8h       ; -> location of TurboRom signature
        ld      b,TURBOSIGLEN
        ld      de,turbosig
        call    match
        ret     nz
        ld      (turboromflag),a; set flag
        ret

;
; Check for BackGrounder ii
;
ck_bg:: ld      hl,(bdosbase)   ; -> location of BGii signature
        push    hl
        ld      de,-800h+5Bh    ; in BGii CCP
        add     hl,de
        ld      b,BGSIGLEN
        ld      de,bgsig
        call    match
        pop     hl
        ret     nz
        ld      (bgflag),a      ; set flag
        ld      de,-800h+1      ; -> BGii ccp entry +1
        call    addderef        ; get that address
        dec     hl              ; BGii task flag is 1 byte lower
        ld      (bgtaskptr),hl  ; save ptr to task flag
        ret
;
notz80msg:
        db      'Not Z80.$'
```

from BackGrounder ii, is shown in Figure 3. It's important to send a test character (a carriage return) in order to actually test the *printer* itself, because a serial printer channel will normally have a UART with a one-character buffer. If the UART's buffer is empty, it will report that *it* is ready to receive a character, even if the printer isn't.

Don't Take Shortcuts

The TERRITORY routine doesn't check for Z-Systems earlier than ZCPR version 3.3 (which supplies the external environment address in HL when it calls each program). It can be extended to do so by searching memory for the "Z3ENV" string and verifying that the self-reference address indeed points to the environment area being examined.

Note that if the string is found, but the address test fails, the search must be continued; it's quite possible to have more than one "Z3ENV" string in memory. Jay Sage tells me there is a group of programs that, regrettably, take a shortcut and stop searching on the first match. They fail to run on his system because it includes, quite appropriately, a directory named "Z3ENV".

The new Z3PLUS and NZ-COM systems have exposed shoddy programming practices, bugs that have been hibernating in widely used utilities. A common one results from the faulty assumption that the Z-System external environment address began on a memory page (xx00 hex). When this happens to be the case, then:

```
ld    l,offset
```

is a shorter route pointing to an environment parameter than:

```
ld    de,offset
add   hl,de
```

But when it's not, the path leads over the cliff.

Clean Up

Use a common exit point. This makes it easier to ensure that nothing is overlooked as your program grows to include new branches.

Close open files and *delete temporaries.*

Restore the default drive and user.

If you've used full-screen terminal features, *leave a clean screen below the cursor.* For some applications you may want to clear the entire screen. In other cases, having the final lines of data remain allows the user to make use of the information when she or he enters the next command; in that case, put the cursor on the bottom line and send a newline.

Finally, if you have not overwritten the CCP, restore the stack and return. Otherwise warmboot.

I certainly don't follow these guidelines slavishly, though I do pay them regard before releasing any software for wide testing. My own temporary programs, run in a known test environment, are often rude, slap-dash pasteups to get the job of the moment done rapidly. But I guard against giving them out to others. It's no favor to pass on code that may explode a friend's system at some unsuspecting moment. Which leads me to recall . . .

A Tale of Too-hasty Design

Some months ago a well-known programmer, author of CP/M several BIOSes, gave me a copy of the BIOS to a particular new computer. I was developing customized operating-system code for this box, and had a similar machine for the debugging and testing cycle. I had re-assembled the BIOS, made several rounds of modifications that were converging to a stable new system running on my box when, poof, I exited from a program and the system went to lunch. It wouldn't reboot, even from power up.

A systems programmer learns (from bitter experiences!) to sit quietly, write down everything he can remember, and think long and hard before he touches anything besides a pencil. Some clues to the crash may remain behind, on disk or in memory (although in this case memory was fully reset). In this case, the A: drive had been a ram disk with uninterruptible power, so I assumed that its system tracks had somehow been damaged.

I decided to make a systematic tour of the A: disk drive by booting up from a floppy system, and looking at different tracks with DU. (Yes, I try always to have a couple of bootable systems stored away on floppies for that black day that a hard-disk or ramdisk goes bad. You should too.) Sure enough, I found what appeared to be file data in the directory tracks. More investigation disclosed that the following tracks, which should have been the directory, had also been corrupted. I took a deep breath, made a mental note to write down what I could still remember of what I had typed in during the last two hours of iterations since backing up the experimental system code. I continued checking the disk. Suddenly, several tracks later, the disk appeared normal.

What could cause such systematic damage? Hypotheses poured forth, only to be discarded. Finally, I saw the glimmer of a pattern. The start of the ram disk—the start of banked memory—contained data related to the data at the very end of the disk. Wraparound! The BIOS had literally gone off the deep end of the ram disk and started writing on the front, clobbering the system, the directory, and the first files after that. As soon as a warmboot was attempted, the system loaded the corrupted system track and was dead.

(Continued)

```
getscbpb:
        db      3Ah             ; return address of scb
        db      0               ; ''get'' code
;
; Z-System environment signature
;
z3envsig:
        db      'Z3ENV'
z3envlen equ $-z3envsig
;
; NZ-COM's  signature in the pseudo-bios, at offset 90.
;
nzname: db      'NZ-COM'
nznamelen equ $ - nzname
;
; BackGrounder ii's signature

bgsig:  db      'BGii'
bgsiglen equ    $ - bgsig
;
; Kaypro TurboRom signature, at FFF8h
;
turbosig:db     'PPS'
turbosiglen equ $ - turbosig
;
;
; System flags (any NZ value means TRUE)
;
z180flag:       db      0       ; HD64180/Z180 cpu
z280flag:       db      0       ; Z280 cpu
cpm3flag:       db      0       ; CP/M Plus
dsflag:         db      0       ; DateStamper
zsysflag:       db      0       ; Z-System (ZCPR 3.3 or later)
extenvflag:     db      0       ; extended Z-System environment
bgflag:         db      0       ; BackGrounder ii
turboromflag:   db      0       ; Advent/Plu*Perfect Turborom
;
dosvers:        db      0       ; } a pair  version number (hex)
dostype:        db      0       ; }    'S' = ZSDOS, 'D' = ZDDOS, 0 = ZRDOS
dosdiskflag:    db      0       ; } a pair  DosDisk
dosdrive:       db      0       ; }         MS-DOS format drive
bgtaskptr:      dw      0
       ; BGii internal flag pointer: bit 1 set = upper task

;
; System Addresses
;
z3env:          dw      0       ; Z-System external environment
biosbase:       dw      0       ; BIOS
bdosbase:       dw      0       ; BDOS
ccpbase:        dw      0       ; CCP
dsclock:        dw      0       ; DateStamper clock
```

Figure 2. Find Top of Usable Memory that Preserves the CCP

```
; Return HL = top of usable memory + 1
;
top_of_mem:
        ld      hl,(0006)       ; load protect address
        ld      a,(cpm3flag)    ; if CPM Plus
        or      a               ;
        ret     nz              ; ..return it
        push    hl              ; for CP/M 2.2
        ld      de,(ccpbase)    ; if protect address is below CCP
        sbc     hl,de
        pop     hl
        ret     c               ; .. return it
        ex      de,hl           ; else return CCP address
        ret
```

Figure 3. A Printer-Ready Test

```
; Return: NZ if printer is ready
;
test_list:
        call    b_lstat         ; if printer is busy
        ret     z               ; ..return
        ld      c,0dh           ; send carriage-return
        call    b_list          ; ..to flush UART buffer
        call    wait            ; allow printer to process it
                                ; and re-test status
b_lstat:ld      de,2dh-3        ; call BIOS list-status entry
        call    bcall
        or      a,a
        ret
;
b_list: ld      de,0fh-3        ; call BIOS list entry
bcall:  ld      hl,(0001)
        add     hl,de
        jp      (hl)


wait:
;       ...                     ; any 10-20 mS routine
        ret
```

A hot theory. But where was the bug? I consulted with the author, and then he remembered— he'd given me the *large* ramdisk version of the BIOS. With less ram installed on my box the addressing had wrapped around.

I wasn't pleased to learn he'd overlooked putting in a *runtime* check for the amount of memory the system possessed. A simple test to make, yet because it was omitted, I now had several hours of reconstructing and testing files ahead of me.

This bug was in hibernation, waiting to byte just when a large disk filled up. Only extreme testing is likely to have disclosed it before it zapped a directory on a customer's machine. We were lucky it happened when it did. Yet it could have been prevented, by systematic design. Environmentally conscious programming would have done it.

Identifying the BIOS

How can a program determine who its host is? What hardware is available? Digital Research (DRI) had no BDOS function to return a version number in CP/M 1.4 and never did introduce one for identifying the BIOS. I have no idea why DRI failed to anticipate this question; a BIOS call to return version information seems now the obvious way to do things. And MSDOS is no better.

Anyway, we are stuck with CP/M's warts; there is no *portable* BIOS call that will identify an Ampro from a Kaypro from an S-100 box. Even if *a particular* manufacturer had the vision to include this entry point, it's catch-22. We can't safely use the call until we know that it's available!

The best approach is to identify the BIOS by reading memory bytes in the BIOS. Every BIOS should include a unique signature and version information. The signature can be an ASCII string, such as "PPS" (in the Advent/Plu*Perfect TurboRom BIOS) or "XBIOS" (in the XBIOS for SB180's), at a known offset from the start of the BIOS. Once the program has identified the type of BIOS, it knows it can make an extended BIOS call, if one has been provided, to obtain additional information. The XBIOS system, for example, provides information on the available hardware devices and their corresponding ASCII names in the system.

The TERRITORY routine includes a ck_bios routine to identify Kaypro systems with a TurboRom BIOS and systems running NZ-COM. Other checks can be added to identify other particular systems.

Unfortunately, a number of BIOSs were written by people who apparently never thought others might write software for their very computer! In these cases, the safest approach is to tell the user in a message that the type of BIOS cannot be determined, and ask him to confirm or enter the model manually before proceeding. ■

Computer Corner

(Continued from page 48)

customer support. WordStar now provides free support for six months. A move closer to WordPerfect's support, but not far enough for many businesses. Another business need of late is graphic support—WordPerfect yes, WordStar no. WordPerfect supports a number of graphic interfaces which make letterheads and newsletters considerably better. I have never found this a problem as I always paste up my work, integrating pictures and text at that time. I know the industry is moving toward being able to do both on one machine, but I still (at least for now) say I can do it faster with a waxer than a desk top publisher (that is however a different article).

One major point which could set the programs apart for some is the ability of WordStar to be changed. In WordPerfect you can set some of the variables, but most of the user interface is fixed (changed only at the factory). WordStar however has always provided a patch list and program to make changes. These changes can be important too. Suppose you are using a not so standard version of PC. WordStar patches shows you where and what variables to change so that you can handle those problems.

Now not everybody will want to change parameters or even be able to understand what you are doing. That of course is why WordPerfect is completely content to let the factory do it. I have known people however that had used a certain word processor for many years and changed all the key codes in WordStar to match their previous system. I guess I would call this the advanced system user.

That is about all the differences I can find. I must admit my study is not exhaustive but then each company is trying to be one up the other. So if you find a difference, I am sure it will not be there the next version.

Reviewing

In summary about the two programs, the point of division occurs, in when and how you were trained. If you started back in the beginning like I did, where you learned by yourself, you will find WordStar more to your liking. If you are a new comer to computers and started in some class or other, WordPerfect may be what you started on and will continue to be excellent for you. Remember, both of these programs are excellent and have few faults. I find WordStar designed for the casual and special user, while WordPerfect has become very popular with the larger middle group of trained workers.

The Home Front

What is happening on the home front is work and more work. I have spent far too many days lately looking for work. While not out pounding doors, I have been pounding nails around the homestead. Our ten acres is badly in need of work, and I dare say I will be doing roofing and carpentry long into the winter months. Until some of that work gets done, my evenings are spent working on software projects and not hardware. The WS/WP comparison was a series of nights. I have been having some problems with my Atari ST in trying to get it to do fractals under Forth. It is beginning to look like I should do the fractals on the NOVIX first, but I do so want to be able to compare PCs, STs and the NOVIX running essentially the same code and program.

I have just read the articles on the NEXT computers. I think they are excellent in design but can see Job's influence. Apple users will remember the problems they have always had with disk formats and disk hardware. Well here it is again, all the hardware for the disks is part of the main system. That has some advantages, but makes big headaches for others wishing to modify or use cheaper components. One feature little shown is the NuBus interface which the Mac IIs use as well. This should certainly add some fuel to their standard interface.

Speaking of standards interface, the AT and PS/2 battle still seems to be going on. I feel the modified AT bus will win out in the long run, with the PS/2 never reaching much impact, especially in after market additions. The standard bus keeps getting better as several companies are making PC and AT compatible system with that bus. I have even read where the VME bus (long the home of the 68000) will have PC compatible boards. This really goes to show how a product can develop a life of it's own. It may have started out the sole idea of IBM but everybody now calls it theirs.

Hardware Conclusions

I have a note here, which I have been forgetting to include. A long time ago I added 5.25 inch drives to the Atari ST. I have many brands floating around and found one of them unable to work. After much head shaking and looking, I discovered the input lines were terminated with 150 ohm resistors. Normal values are 470 or 680 ohms. Changing to the higher values solved all the problems. Those were Mitsubishi drives, but I feel that problem could exist with any system or drive type. Just check a problem line with a scope or voltmeter and make sure it drops below 0.8 volts and goes above 3.0 volts. At 150 ohms my signal lines were not dropping below 2.0 volts.

My hardware bits of wisdom for this issue have to do with soldering irons. One end is hot, the other cold, and one who grabs the wrong end seldom forgets which. ■

WordPerfect 5.0 Upgrade
81 North State Street
Orem, UT 84057
Phone (800) 222-9409
1. Send $60
2. Letter with Name/address
3. Indicate number of copies
4. Indicate 3.5 or 5.25 media
5. Include ORIGINAL TITLE page

WordStar Release 5.0 (PC ONLY)
(800) 227-5609
1. $129.00
2. Have serial of CP/M version

Back Issues Available:

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

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

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

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

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

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

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

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 L.B.
- Building a SCSI Adapter
- New-Dos: CCP Internal Commands
- Ampro '186 Networking with SuperDUO
- ZSIG Column

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

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

Issue Number 28:

• Starting your Own BBS
• Build an A/D Converter for the Ampro L.B.• HD64180: Setting the wait states & RAM refresh, using PRT & DMA
• Using SCSI for Real Time Control
• Open Letter to STD-Bus Manufacturers
• Patching Turbo Pascal
• Choosing a Language for Machine Control

Issue Number 29:

• Better Software Filter Design
• MDISK: Adding a 1 Meg RAM disk to Ampro L.B., part one.
• Using the Hitachi HD64180: Embedded processor design.
• 68000: Why use a new OS and the 68000?
• Detecting the 8087 Math Chip
• Floppy Disk Track Structure
• The ZCPR3 Corner

Issue Number 30:

• Double Density Floppy Controller
• ZCPR3 IOP for the Ampro L.B.
• 3200 Hacker's Language
• MDISK: 1 Meg RAM disk for Ampro LB, part 2
• Non-Preemptive Multitasking
• Software Timers for the 68000
• Lilliput Z-Node
• The ZCPR3 Corner
• The CP/M Corner

Issue Number 31:

• Using SCSI for Generalized I/O
• Communicating with Floppy Disks: Disk parameters and their variations.
• XBIOS: A replacement BIOS for the SB180.
• K-OS ONE and the SAGE: Demystifing Operating Systems.
• Remote: Designing a remote system program.
• The ZCPR3 Corner: ARUNZ documentation.

Issue Number 32:

• Language Development: Automatic generation of parsers for interactive systems.
• Designing Operating Systems: A ROM based O.S. for the Z81.
• Advanced CP/M: Boosting Performance.
• Systematic Elimination of MS-DOS Files: Part 1, Deleting root directories & an in-depth look at the FCB.
• WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII terminal based systems.
• K-OS ONE and the SAGE: Part 2, System layout and hardware configuration.
• The ZCPR3 Corner: NZCOM and ZCPR34.

Issue Number 33:

• Data File Conversion: Writing a filter to convert foreign file formats.
• Advanced CP/M: ZCPR3PLUS, and how to write self relocating Z80 code.
• DataBase: The first in a series on data bases and information processing.
• SCSI for the S-100 Bus: Another example of SCSI's versatility.
• A Mouse on any Hardware: Implementing the mouse on a Z80 system.
• Systematic Elimination of MS-DOS Files: Part 2—Subdirectories and extnded DOS services.
• ZCPR3 Corner: ARUNZ, Shells, and patching WordStar 4.0

Issue Number 34:

• Developing a File Encryption System: Scramble data with your customized encryption/password system.
• DataBase: A continuation of the database primer series.
• A Simple Multitasking Executive: Designing an embedded controller multitasking system.
• ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
• New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
• Advanced CP/M: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
• Macintosh Data File Conversion in Turbo Pascal.

Issue Number 35:

• All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
• A Short Course in Source Code Generation: Disassembling 8086 software to produce modifiable assem. souce code.
• Real Computing: The National Semiconductor NS32032 is an attractive alternative to the Intel and Motorola CPUs.
• S-100 Eprom Burner: a project for S-100 hardware hackers.
• Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
• REL-Style Assembly Language for CP/M and Z-System: Part 1-selecting your assembler, linker, and debugger.
• ZCPR3 Corner: How shells work, cracking code, and remaking WordStar 4.0.

THE COMPUTER CORNER

by Bill Kibler

It seems like so many years ago that I started writing for Art. It seems like even longer ago when I first used WordStar + e4. I think I got my first copy, version 2.8 beta copy, just after becoming their 25th employee. Back then MicroPro thought they could do no wrong. Everyone wanted a copy and they could care less what they had to go through to get one.

Times have changed. It is now eight years later and I have spent hundreds of hours using WordStar. It is hard to believe but I have been writing articles for Art for almost six years now. I have done this corner for over four years (seems like longer). WordStar is not the only word processor I have used either. For a while I had to use WordPerfect + e4 in my Masters program and have since taught several how to use it.

It is pretty hard these day not to find some magazine which hasn't reviewed the two. Most of the time it comes out that the reviewers like WordPerfect. I never have agreed with them, so I decided to buy both of their version fives and compare them.

WordPerfect 5.0

When WordPerfect 5.0 came out, they offered upgrades from 4.XX (whatever) for $65. I had just returned from the SOG and was visiting a friend who was using 5.0. He showed me where to get the upgrade and why he liked the new version. I mumbled something about still not better than WordStar and he shook his head, saying "got em beat by a long shot." Most of that was after he explained how they helped him on the phone to do things it was suppose to do but didn't for him. We spent lots of time talking about how WordPerfect has 144 phone lines and operators 24 hours a day seven days a week. Hard to beat support like that even if the product didn't work, which is not the case.

I have my copy now, took about three week after I mailed them a check and the inside front cover of my version 4.0. What I got was 10 new disk, some sales literature, a pamphlet listing changes, new manual pages (sorted and divided differently than before), and a 400 page workbook. The workbook and their tutorial program give you guided hands on practice with the most common word processing activities. I have mainly played with their newsletter example which I will have more to say about later.

WordStar 5.0

Since I bought version 4.0 for CP/M, I have received a newsletter from MicroPro. An 800 number was listed for updates and customer service. When their version 5.0 came out for the MSDOS/PC systems, they covered it and stated they had a special upgrade price for all their users. Well, that price is not as good as WordPerfect, but $129 is much better than their full list price. True that is twice what the WordPerfect price is, but then I like WordStar better, or at least that was my premise.

I called the 800 number on Wednesday, and the lady said it would take up to four weeks. On Friday UPS dropped it at my door. That means they put it in UPS's hands the same day I called, which is what I call service. I just wish the operator had been a little more knowledgeable about the true shipping schedule. I could tell that the two companies had been reading the same complaint columns in computer magazines as I got 10 disks, a short pamphlet on what is new, and a bound workbook like manual for version five. A big difference is that WordStar has also included some other software in that set of ten disks. You get a telecommunications program, a file finder program, and PC-Outline by Brown Bag software. The manual has 550 pages and is broken down several ways; training, alphabetize topics, extra programs, references section (WordPerfect's manual is the same also).

Some Comparisons

A long time ago I explained how I felt documentation should be prepared, and both companies have pretty much followed what I said. It is not that I am an authority, it is just, that is what users need to understand and to properly use programs. Any company staying in business long enough will find those facts out, some faster than others. Of the two manuals, I might lean toward WordStar over WordPerfect on one fact—it is all bound in one book. If you are learning some feature and need to seek the reference section WordPerfect has it in a different book. That is a pretty minor difference if you ask me, and shows just how close the two are in quality.

The place that you find the most difference is the user interface. Both programs now support almost all printers made. Each has special abilities to handle the laser printers. The number of fonts possible are endless in most cases. They both have features to display the page completely.

My complaint about WordPerfect from day one still exist, and is why I still prefer WordStar. Untrained and casual users will find WordStar easier to use, especially with their new help level 4. This new level uses pull downs and does away with the old "classic" menus. The main menu is replaced with three choices; *files, other, additional.* This gives the user a known place from which to start. In WordPerfect they start at the blank screen approach. If you haven't been trained in WordPerfect, it is impossible to find your way around the program. I must say that most users do not complain (at least to me) about the hundred or so options they need to learn to use the program.

The WordStar approach is similar to the Xerox papers (the source of Macintosh's interface) where you are led through a series of menus to the desired function. Typically this is never more than two keystrokes to get the desired action, but it can be more. WordPerfect on the other hand uses no such menus. Yes many of their choices do appear in menu selections, but the documentation and approach is not based on that idea. I have not found a causal user of WordPerfect. Either you use it as your sole processor or not at all. It reminds me a bit of many people and their love hate relationship with UNIX. Power users love it, casual user never know what is going on.

The Major Differences

A couple major differences do exist which many organizations can't do without. We talked before about the

(Continued on page 45)

AUSTIN COMPUTER SYSTEMS

YOUR CHOICE

Austin Computer Systems: Heavy on Power and GE Service, Light on Price

AUSTIN 386/20 CACHE

Standard Features:

- Intel 80386 Double Sigma Processor runs 20 MHz
- 30 MHz Throughput Performance
- American Made Motherboard Expands to 8 MB, 32 Bit RAM
- 64K Static RAM Cache @35 Nano's
- 1–32 Bit, 6–16 Bit, 1–8 Bit Slots
- DeskTop Chassis – Tower Optional –
- Dual Floppy/Harddisk Controller
- 3.5" 1.44 MB or 5¼" 1.2 MB Floppy Disk Drive
- Keytronics Enhanced 101 Keyboard
- 2 Serial Ports & 1 Parallel Port
- Clock with Battery Backup
- 80287 & 80387 Math Co-Processor Support
- Users & Technical Reference Manual
- Built-in Setup & Diagnostics Program
- Fully DOS 4.0 & OS/2 Compatible

- **VGA Color Monitor**
- **VGA 640•480 Analog Display Adapter**
- **80 MB 28 mls Fast Access Hard Drive**
- **1 Full MB of 32 Bit RAM**
- **GE 1 Year On-Site Service Contract**

$3795.00

Complete as shown

Austin 386-20	MONO 720X350	EGA 640X350	VGA 640X480	VGA DELUXE 800X600
40 MB*	$3095	$3495	$3595	$3795
80 MB*	$3395	$3695	$3795	$3995
230 MB*	$4795	$5195	$5395	$5495

*ALL HARDDRIVES ARE FAST ACCESS=40 MLS or LESS

- **EGA Display Monitor**
- **Paradise EEGA Autoswitch Display Adapter**
- **40 MB Fast Access Harddisk**
- **1 FULL MB of RAM**
- **Supports EMS LIMM 4.0 Standard**
- **GE 1 year On-Site Service Contract**

$2295.00

Complete as shown

Austin 286-12.5	EGA 640X350	EEGA 640X480 W/MULTI-SCAN
40 MB* 1:1	$2295	$2445
80 MB* 1:1	$2595	$2745

*ALL HARDDRIVES ARE FAST ACCESS=40 MLS or LESS

AUSTIN 286/12.5 CACHE

Standard Features:

- True Intel 12.5 MHz 80286 Processor
- Dual Floppy/Harddisk Controller with 1:1 Interleaving & 32K Harddisk Cache
- American-Made Motherboard Expands to 4 MB, 16 MB total
- Shadow RAM buffers slow BIOS into Fast RAM
- 3.5" 1.44 MB or 5¼" 1.2 MB Floppy Disk Drives
- Phoenix BIOS
- Keytronics Enhanced 101 Keyboard
- 2 Serial Ports & 1 Parallel Port
- 1 PS/2 Mouse Port
- Clock with Battery Backup
- 80287 Math Co-Processor Socket
- FCC Class B Approved
- Users & Technical Reference Manual
- Diagnostic & Utility Programs
- Fully DOS 4.0 & OS/2 Compatible

GE Computer Service

CALL AND ORDER TODAY!

1-800-752-1577

Fax (512) 454-1357

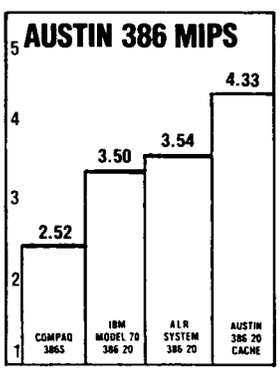
Overseas (512) 458-5106

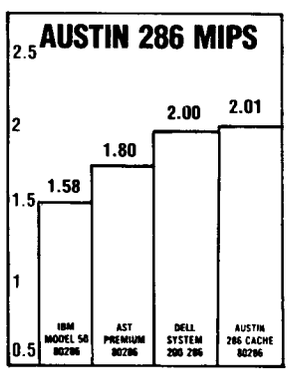
MONDAY–FRIDAY: 8AM–7PM CENTRAL
SATURDAY: 10AM–5PM

AUSTIN COMPUTER SYSTEMS

7801 N. LAMAR • SUITE E-198
AUSTIN, TEXAS 78752

MIPS Test and Bottomline: Performance You Can Really Count On





AUSTIN 286/12.5 BOTTOM LINE

FEATURES	AUSTIN 286 12.5	ULTRA-COMP	CLUB	CMO	ZEOS	COMPU-ADD
One-year on site service included	YES	NO	NO	NO	NO	NO
Uses surface mount construction	YES	NO	NO	NO	NO	NO
Supports up to 4 mb of ram on the motherboard	YES	NO	NO	NO	NO	NO
Supports 1:1 interleaving	YES	NO	NO	NO	YES	NO
6 layer motherboard	YES	NO	NO	NO	NO	NO
640 · 480 EEGA auto-switch display adapter	YES	NO	NO	NO	NO	NO
Built-in mouse port	YES	NO	NO	NO	NO	NO
Motherboard made in U.S.A.	YES	NO	YES	NO	NO	NO