The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 57 September / October 1992 US$3.95

Z-System Corner

Dr. S-100

Home Automation with X10

Real Computing

File Transfer Protocols

MDISK at 8 MHZ.

Introduction to Forth

Shell Sort in Forth

Z AT Last!

ISSN # 0748-9331

TCJ The Computer Journal

Issue Number 57 Sept / October 1992

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Bill D. Kibler

Technical Consultant
Chris McEwen

Contributing Editors
Brad Rodriquez
Matt Mercaldo
Tim McDonough
Frank Sergeant
Clem Pepper
Richard Rodman
Jay Sage

The Computer Journal is published six times a year and mailed from *The Computer Journal*, P. O. Box 535, Lincoln, CA 95648, (916) 645-1670.

Opinions expressed in *The Computer Journal* are those of the respective authors and do not necessarily reflect those of the editorial staff or publisher.

Subscription rates within the US: $18 one year (6 issues), $32 two years (12 issues). Foreign (surface rate): $24 one year, $44 two years. Foreign (airmail): $38 one year, $72 two years. All funds must be in U.S. dollars drawn on a U.S. bank.

Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 535, Lincoln,CA 95648.

Registered Trademarks

EDITOR'S COMMENTS

Welcome to issue number 57. This time TCJ is back to normal minimum size. Time and money (the problems of any business these days) have me keeping the size from getting too large. Actually I am saving up for the big celebration issue number 60.

NOTICE TO ALL WRITERS

Issue number 60 will be our TENTH YEAR aniversary issue. Since there are only two more issues to go, all our writers (and hope to be writers) need to review what they want to say and do for this special issue. I am looking for retrospective articles. Articles that compare systems of 1982/83 vintage to today's.

For "CLASSIC" system people, I am looking for articles that chronicle the rise and fall of various vendors. A good example might be the BIG BOARD system which got Dave Thompson doing Micro Cornicopia magazine. That board became the Xerox 820 and Kaypro.

MICRO-C DISK LIBRARY

I have a tenative agreement with Dave Thompson to acquire the rights to sell the Micro C floppy disk library. If all goes as planned, by next issue I should have all the disks in house and be able to supply those disks. The library consists of 5 1/4 and 8 inch disks for Xerox, Kaypros, Big Boards, and PC based systems. This move is part of my CLASSIC system support concept.

Many disks in the Micro C library are not available elsewhere. This is important if you are trying to bring up an older system. A number of CP/M BIOS source files are in the library. Along the same grounds, I will as time permits, be going back through old issues, putting listings on disk. This will take considerable time, so I do not expect to have any available till summer of next year.

ISSUE 57

In this issue our regulars are back in full strength. I must congratulate them for being able to produce so much, for so little reward, and being under so many other obligations. Jay Sage and Rick Rodman have big columns, and Frank Sargeant has what so many have been asking for, a FORTH intro tutorial.

A number of articles were unable to meet press deadline, because of their own product deadlines (the D/A article series is being dealyed due to product shipment and last minute modifications.) Spread throughout the issue, you will find some new features like, CLASSIFIED, SUPPORT WANTED, NEWSLETTER REVIEW, and so much more. So read on and enjoy! Bill Kibler.

DELAYS, DELAYS, and more DELAYS

The bad side of the changeover has been delays. Issue number 57 was shipped in early October and not our normal September 15 mailing. Some back issue orders have taken 3 months to complete. Subscriptions, address updates, and renewals have all been delayed in one form or another. To our readers we apologize for the DELAYS. We are working as hard as we can to correct any and all problems. Systems are being set up to streamline the operation and prevent problems. Some systems are in place, many are still waiting their turn. We ask all readers to be patient with us as the transition continues into next year.

What can you do to help? From our readers we ask that you write and not call whenever possible. Written responses are less prone to errors, especially when they contain full addresses and names. Please do not send only checks or half filled out forms. We cross check everything received for errors. Short and cryptic correspondence does not help. We have recently received incompleted credit card orders, requiring a second contact and more delays. Please proof read and provide phone numbers (only with credit card orders.)

The best help we can get from you the readers of The Computer Journal is your financial support. All these changes are taking money. We ask that you read and review your subscription labels, and when possible pay early. That helps save us the work and expense of sending out renewal forms. TCJ will no longer send issues that have been lost due to not receiving a change of address request in time. We use the post office 3547D service. They charge $.35 to tear off the cover (and throw away YOUR issue) when you do not offer to pay forwarding charges. All we get from the postal service is your new address. If you figure in all the costs, your lost issues, form 3547D, a second issue, mailing expenses,our cost nears $4.00 to get your second issue to you. We typically recieve 10 to 20 3547Ds each month. Only you can help us with this problem.

Lastly please support our advertisers. They support TCJ by helping to offset the cost. So when contacting them, please say you saw their ad in TCJ. Our only alternative is to raise our subscription rates, which are now very low. Your help and comments are always welcome.

READER to READER

Letters to the Editor

All Readers

MINI Articles

92/08/11

From: L.CAMERON5
To: B.KIBLER

Sub: TCJ issue 56

Dear Mr. Kibler,
I have a number of comments that I would like to make on issue #56 of TCJ inspired by your editorial comments:

I have spent almost two years now putting together my vintage IMSAI computer and have enjoyed it tremendously. When I first bought it, it was a mess and I have lovingly restored it to perfect working condition. This computer introduced me to electronics; before starting to work on it I didnt know the right end of a soldering iron! I would love to see more articles on the S-100 buss. How about printing a list of people who stock S-100 buss compatible cards? I had a hell of a time locating cards for my IMSAI and actually took out a ''wanted'' ad in Nuts and Volts mag to be able to find them.

My IMSAI is running a Z80 Cromemco processor board, IMSAI BASIC and IMSAI assembler from 2708's, 48K RAM, and a Hayes S-100 300baud modem (I'm writing from my IMSAI). Currently I am working on getting up and running an Altair. I guess you could say this is my hobby and I would like to be able to contact other antique computer junkies thru your magazine! Sometimes I feel like I am the only person crazy enough to be doing something like restoring old ''obsolete'' computers but I get a kick out of it; I feel like I am preserving a piece of something that has been forgotten about, but literaly changed our society!

I think your changing the focus of the magazine to more hardware related projects is a good move. I cant honestly say that I have gotten anything out of my trial subscription but that may change I hope with the next issue. The software articles you published seemed to be directed toward an extremely small group of your readers. I agree with your comments about Elektor electronics trying to reach to broad of a cross section of interests and in my opinion publishing unfocused and watered down articles, but your magazine was at the other extreme: too narrow!

In conclusion: ''platform independent'' good idea; continued S-100 stuff: good idea; more focused on hardware and less on software: definetely a great idea.

Larry Cameron

p.s. anything on GEnie Z80 related that I can download and make use of on my IMSAI?

=END=

Larry:

Thanks for the GENIE message. On S-100, you will be happy to see our new Doctor S-100. We have also started classified columns to help you find and sell items that only other ''Classic system'' lovers would want. I prefer the term ''CLASSIC'' in place of ''obsolete'', but I think most of our readers still remember with fondness their first put it together project. TCJ's readership is rather wide, so there may be some articles over your head at first, but by looking up old articles and just hanking in there it will all make sense one day soon. To help you out, we are starting a few more beginners columns to try and bring your skills up to date. Thanks for the good words. BDK

9/15/92

Dear Mr. Kibler

Congratulations on becoming the editor of the computer journal. I have been a subscriber for several years and have enjoyed the articles very much. I was very interested to read the letter by Herb Johnson of Trenton, New Jersey expressing his interest in S-100 bus systems. As a long time user of Compupro equipment I would very much like to see articles about the S-100 bus.

I also wanted to express my opinon of the new Computer Journal. I was somewhat disappointed to see the small size of the new magazine. It seems to have dimished to less than half of its previous size. I was also disappointed not to see the column on Z-system software by Bill Tishey. Since The Computer Journal is one of the last 8-bit computer magazines, I have always looked forward to the articles about Z-system programs and other information about 8-bit software and hardware.

The other reason that I felt compelled to write was the poor quality of some of the writing. I was particulary disturbed by some of your articles and comments. Your writing is riddled with grammatical errors and misspellings and some of the sentenences do not make sense. Mr. Kibler, there is no letter ''a'' in the word excellent and [possessives are formed by adding apostrophe 's' such as in ''IBM's implementation'' and ''Netware's

NETBIOS." Also, referring to page 4, people do not "do" articles any more than the "do" lunch. I really don't think you need to include your initials at the end of your comments to identify yourself as the writer. By the way, the sentence on page 2 should be "are my comments" not "is my comments."

I am by no means an expert writer but I do find the multitude of errors unacceptable, especially since your first sentence is about the 'improved" journal. I expect the editor of any magazine to have at least basic writing skills and be able to spell. I don't wish to be excusively critical and, if you would like, I would be very happy to proof read any of your writing before publication.

Please let me know if I can be of any help. Best of luck with the magazine.

Regards, Chris Christsen, S.B. CA

Chris:

Thanks for all the good and bad comments. I really do try and catch all the mistakes possible. When I put issue 56 together, my main concern was making it to press ASAP. Far too much time had passed between issues and making deadline was foremost. That meant my third and four proofing just didn't happen. I would love to use your help, but I do most all the work between my job and teaching. TCJ would never make it to press if I had to ship everything somewhere else to be checked. Thanks anyway.

The size and missing articles were mostly due to the change, people on vaction, and everyone being overworked. We try to keep the magazine at 60 pages and feel that #56's size will not happen again. Since adding so many new writers and columns, even if someone takes time off, there will be plenty to read without them. I am looking forward to hearing from you again after I have had a chance to introduce all the new writers and topics. I hope by this time next year to have a much better handle on everything you mentioned. Thanks again. BDK
P.S. After your comments on my initialing my writing I checked several national magazines. The editors all put their initials after their comments and I do it now so that later editors (I plan on having my "contributing editors" respond in later issues) can be distinguished from mine.

8/11/92

Dear Bill Kibler,

I really like your ideas about TCJ for the next 10 years. I think Forth is a great language and a great way to learn about programming. Your emphasis on platform independent projects is excellent.

De-emphsizing programming is okay. I think software should be treated more like hardware anyway. By this I mean put the software and processor in a black box and use it as a functional block.

Would you be interested in projects that involve i2c bus from Signetics? These chips make for a fairly platform independent approach.

I got the 68HC16 kit and it looks good. I also got the 68HC05K1 kit and it is excellent. Maxim also puts out really great kits. Some vendors are really getting the idea.

Are any of your issues available in text form only on a disk? This would be more usefful to me than a hardcopy. I can read 3.5 inch Mac and IBM format.

I primarily work with MacIntosh systems and single board systems such as the New Micros 68HC11.

Thanks for TCJ

Gus Calabrese, Denver CO

Gus:
Seems like I am on a roll with the NEXT TEN article, now all I need to do is make it all happen. Maybe you could help make it happen with a few articles on those Motorola kits. I have been planning on doing them myself, but not a chance these days. How about it? Got any great little learning project for those trainers? TCJ will be suppling code on disk in the future, but for now it has to be hardcopy. I can and will be uploading some code to GENIE, but proably not till next year when I have more time. On doing an article about the i2c, it sounds great, in fact I have several people who have suggested the same. My policy is to print as many article on a single topic as possible, only as long they are not exactly the same. So contact me about what you want to do and I will make sure it hasn't already been done. Thanks again Gus. BDK

9/8/92

Dear Mr. Kibler:

I was pleased with your article about the next ten years for the TCJ. I am a retired Electrical Engineer who got into computing just after the WW-II, using analog machines, onlly to miss the fun of digital machines due to other demands on my time. Now with time to play "catch Up", your publication is most interesting to me. I was sorry to see Micro-Cornucopia cease publication.

As a member of a local computer club, I edit the small newsletter, a copy of which is enclosed, since I commented on the TCJ in a manner I hope is pleasing to you.

I am looking forward to Herb Johnson's articles since I have several S-100 machines in various stages of dis-repair. I also have some MS-DOS equipment.

I have enjoyed the embedded controller articles.

I would like to see some articles on adaptation of MS-DOS machines and printers to barcoding of mail. It seems to me that it is high time that the Post Office Department extend reduced postage rates to all mailers, not just volume mailers with varying degree of political clout.

Best Wishes,
Eliot C. Payson, Litteton, CO.

Eliot:

Hopefully Herb's articles will get your S-100 systems running like new again. Maybe we will even be able to get you *running ZCPR on them as well. Your clubs newsletter was great and thanks for the good words. Barcoding hasn't yet been covered in TCJ, guess it is an oversight on our part. Maybe someone out there reading this can correct the problem. One ofTCJ's writers is snowed under with other projects (and our article as well), but when he sees the light of day, I will try and get him to report on how they do barcoding in Forth on a 80451. BDK*

8/15/92

Dear Bill,

I read your computer Corner article about your trials and tribulations using Netware (known as NetPig to it's friends).

I've been down the same road that you are currently on. I ran a two computer network for 18 months using The $25 Network ($25N). It is a super product and I highly recommend it. I upgraded the systems to The Little Big Lan (LBL) (aka The $75 Network). It is also a superior product and I also recommend it. LBL supports both serial and Arcnet cards. If you can put up with the speed restricitons of a serial network and you have two or three netnodes then the $25N will work fine. If you think that some day you will grow the network and will require the additional speed that Arcne gives you then LBL would be a viable choice.

In my case I wanted to set up a network that was:

inexpensive,
high speed,
did not cost a lot of money,
used twisted pair,
was cheap
did not require a dedicated server,
low per node charge,
was easy to use,
economical,
easy to install and maintain
frugal,
allowed multiple peers in the network.

The last requirement was to allow me to use the single tape drive to back up the local disk on each node. I also wanted to offer each system's resources to other users on the network.

My choice of networks is Artisoft's LANtastic. It is a very inexpensive peer to peer network product. LANtastic supports a number of different network boards in a variety of ways. There are some network boards that LANtastic will talk to directly. There are others that you need the Adapter Independent (AI) version. With the wide number of different network adaptors, most of them supported in some way by LANtastic.

I was able to obtain a large supply of AT&T Starlan cards. Starlan is a 1MB twisted pait network. I have a ton of unshielded twisted pair (UTP) in the house, I wanted to use the cable rather than pulling new cable to each node. Sadly Starlan is not supported by LANtastic. Something about the age of the boards (pre transistor?) and the strange memory addressing scheme. After a number of tries with techsupport it looked like a lost cause.

Then Artisoft announced LANtastic for Netware. LANtastic for Netware uses the IPX drivers and NETBIOS. You load just the LANtastic REDIR and where needed the SERVER programs. The LANtastic packets ride inside the IPX packets, the network delivers the packet to LANtastic.

```
***************************
*     THE HARDWARE        *
***************************
*          IPX            *
***************************
*        NETBIOS          *
***************************
*         REDIR           *
***************************
*         SERVER          *
***************************
```

A friend and I did some digging and found Starlan IPX drivers. We also found a very old (1985) copy of NETBIOS. We loaded them into a test system, loaded LANtastic and everything worked. The neat thing is that you do not need Netware, just the IPX drivers and NETBIOS.

The network works quite well. I've been running it for about five months and I've had zero problems in using it. It is transparent to all of the programs that I use and most major applications, for example Windows 3.1, see that it is there and can cope with it.

Like you I've been digging on the BBS systems and I have found a number of programs that work with LANtastic. Most of the "well behaved programs" are on the Artisoft BBS and on Compuserve. I picked up a very nice LAN monitor that tells me about packet flow across the network. Since we are all very lose the need for E-Mail is very small, LANtastic has a very good E-Mail program built into it. It also has a "chat" program, pretty useless when you can see the other person, but it may be a feature to you.

Let me address the cost. The cost for LANtastic for Netware is $299. This allows you a 300 node network. The cost of the Starlan boards range from $100 each to two for $25. (I'll let you decide which boards I purchased.) Each node in my network cost about $75. Depending on the cost of your adaptors the cost may be lower or higher. The key factor is that IPX drivers can be found for most boards and that my activity is supported by LANtastic.

Ok, last plug for LANtastic. It has been a PC Magazine "Editors Choice" the last few times it has been reviewed. The other great feature is Artisoft Techsupport. I call, I get answers or I get a baffled tech, who finds another tech, who calls me back. We've thrown some really gorky questions to them and they have responded with some super answers. (That actually was two plugs for Artisoft, one ofTechsupport's big problems is my inability to count.)

I'd like to address some of the comments you made. I'll refer to the paragraph numberr of your article.

Paragraph 9. You reference two books about NETBIOS. Another good source is Ralf Brown's interrupt list that he publishes four times a year. It describes the NETBIOS structure and I've found it to be very accurate.

Paragraph 10. LANtastic is also loaded as TSR's. You can use the MARK and RELEASE programs to remove them. Due to the tiny size and the ability to load them high, I doubt that you will need to recapture the memory.

Paragraph 12. It's interesting that most of the mags state that LANtastic is not good for networks more than a few hundred users. It was really interesting that a survey of networks shows that the average number of nodes in the network is fairly small.

Paragraph 13. You hinted at an ODI serial driver. I'd like to find a serial IPX or ODI driver for my laptop. Let me know when the code is done, I'd be happy to field test it.

Paragraph 14. The are a number of versions of NETBIOS out there. The one that I use is from 1985. You may find that an older version will work better for you. There is a new Novell NETBIOS that was released in June that may give you a big improvement.

Ok, this has gone on long enough. I'd suggest that you try LANtastic. It has worked very well for a number of us. We even use L4N at work, we've had so few problems our biggest problem is finding the manuals when we want to change anything.

I agree with you once you get more than one system, it's time to network them. With the number of low cost, easy to use products, it makes sense to network.

Keep up the good work with TCJ!!!

Foster Schucker, Eagle PA

Well Foster that was some letter (actually a mini article.) I am sure all our readers are glad to get that review, but would be more interested in knowing where the Starlan cards can be had for $25 (send me a couple and I will pay you back later!) Like I said in my article, Artisoft technical support makes their products one of the best. We just started using the Netware NETBIOS 3.10 and it solved most of our problems. However one problem that had been fixed, popped up again. I really do not know why so many people who have only two or three computers buy 10 or 20 node Netware products. My feelings are the sales people (used cars sales people before becoming a computer person) just didn't tell them of all the alternatives. Currently I don't have time to work on a serial ODI, but maybe one of our readers does. Any takers??? I have been thinking about how to do LANs on CP/M or any mixed platform system. Maybe I can get the $25 network people to tell us how to talk to their system from other machines. Till then, thanks for the great review. BDK

TCJ:

Bill,

I've enjoyed The Computer Journal for many years now and I applaud your taking-over in light of Chris' illness. I've read your column in every issue (though it is not my cup of tea) and learned much from it. Although Jay Sage is my favorite author, I am writing to ask what happened to the Paul Chidley column. I am in the process of building a YASBEC and I wait with baited breath for the latest developments on what I think is the most interesting project since Bill Godbout started putting out kit computers. I have read your article about the magazine's direction and I can somewhat agree, but this mag was the premier vehicle for information on the subject and Z-SYSTEM in general. Please don't drop another segment of your audience as so many other magazines have. I have been toying with the idea of floppy controller and some ports for the YASBEC as a system addition and I can understand your bias toward homebrew and microcontrollers, I have learned enough to feel that I can do it on my own, given time, others may feel this way too. Give them a home as we've been thrown out of so many other places.

Respectfully,
Jim Thale, Deerfield, IL

Guess I missed the mark a little Jim. My intentions are to make TCJ the ONLY magazine to turn to for people who want to tinker with "CLASSIC" systems. Please feel at ease that we will NOT become another PC only magazine. Jay is back in full this issue, but I have not been able to get anything yet on YASBEC. I will press everyone for something in the next issue, but since they don't get paid for it, I can't press too hard. I am not sure what you or I meant on a "bias", but I feel very strongly that if you want to learn, you have got to get in there and "tinker" with your hands. The fact that you feel like you can do the tinkering now says that maybe TCJ has helped at least one reader to become more skilled. Thanks for the letter. BDK

Dear Chris:

You will no doublt be heartened to hear that I've received each issue to date, already opened. TCJ obviously creates interest as it passes through sorters hands in the mailroom. I'm not going to suggest a mailing envelope, as obviously costs would escalate. There's not a lot of room at the page edge either. I can imagine howls of protest if a staple was found to obliterate some cryptic bit of assembler listing at page edge so perhaps the use of your simple sticker has been well thought through. #54 arrived with the extra taped edges, made from what appears to be mailing labels. These worked well.

To things Morrow, my MD3 to be exact. Still running on floppies, but at least I've managed to mount Quaddrives, so we have nearly 800KB per drive. I may well take up your generous offer on Socrates BBS to help us find specific programs. There are about 3 I have in mind, and just perhaps will be obtaineable through or on the BBS.

I look forward to suggested up coming articles on generic SCSI add-ons and confess that the enthusiastic and detailed reviews of the Z-System components have me hooked! Though this 8 bit software is a "must have", I'm a little unsure of just what exactly the purchase of NZCOM will get me, and how complete and up to date will the utilities be. How much therefore, would I be required to purchase, to get a full Z-System??

As I also log onto Sypko Andreae's M.O.R approx once per month, and by your American standards, earn very little; you'll appreciate that I can spread myself only very thinly. Calls therfore, to Socrates can also be only brief, and perhaps once per month.

Your acceptance of MasterCard will make renewal much easier, and the purchase of p.d s/ware disks off the bbs... is this service still available? I imagine, that for most of us distant subscribers, all but the smallest d/loads (or up) will be far too expensive.

Thank you for your time, and a most definite BIG thank you for all the team who help produce such a readable and informative TCJ.

Sincerely, Paul MacDiarmid, Rotorua, New Zealand.

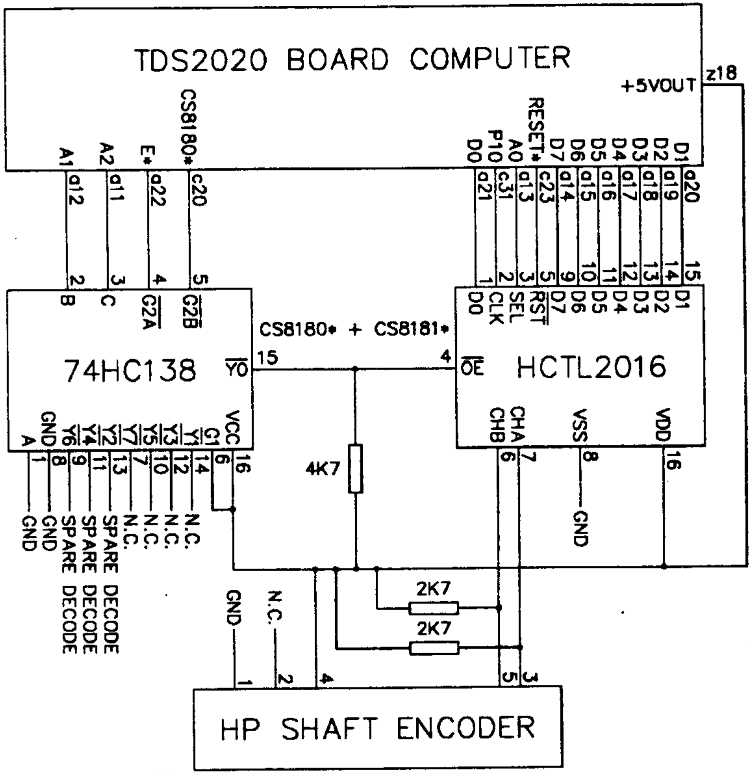
This letter was passed to me by Chris, and sorry to say Paul but the BBS is no longer on line. I am not sure if Chris will put it back on line from his new place in Washinton state or not. Jay Sage has a BBS you might try in place of Chris' (617-965-7259). Most of us have moved to the CP/M section of GENIE and I will be starting to upload things to it soon. I have passed your request on to Jay about NZCOM and what you really need to bring a system up. Supposedly it has all been done in past articles, so maybe a back issues order is what you need at this point. I am trying to get some beginners type reviews, as we are getting so many first time CP/M users. I currently do not have any Morrow writers, but if you need help for something specific try our new and free, help needed section. On mailing envelopes and the mail in general, have I stories to tell you. For your information and all our readers, I have had many problems with getting the magazine out the local post office. TCJ is much more than they are use to. It will proably be issue 60 that gets mailed without any troubles, so hang in there and stay with us while I get the new problems resolved. BDK

Mail your letters to :

TCJ
P.O. Box 535
Lincoln, CA 95648-0535

NEWSLETTER REVIEW

TCJ will reprint articles of interests to our readers from NEWSLETTERS we receive. To have your newsletter reviewed, inculde a letter indicating permission to reprint articles and include us on your regular mailing list. We reserve right to select or NOT select articles for reprinting. TCJ does not endorse or support any products presented. This service is provided as a means of assisting our readers find alternative sources of information.



```
( HP Shaft Encoder software) DECIMAL
0 VARIABLE PREVIOUS ( copy of last reading)
0. 2VARIABLE COUNTER ( 32-bit signed accumulator of current
   shaft position)
$8180 CONSTANT ENCODER ( change for other chip selects)
: SETUP ( set Data Direction Reg to put 10MHz clock on c31)
   $0F $FF80 C! ;
CODE GET            ( -- n   read shaft position)
   R3                  CLR,       ( ensure top byte is 0)
   R4                  CLR,       ( ensure top byte is 0)
 B ENCODER ))       R3 MOVFPE,   ( get most sig. byte)
 B ENCODER 1+ ))    R4 MOVFPE,   ( get least sig. byte)
 B R3                  SWAP,      ( transfer to top byte)
   R3               R4 ADD,       ( merge to 16-bit count)
   @-R7             R4 MOVO,      ( push count on stack)
 END-CODE
: POSITION          (  --    update 32-bit position counter)
                    ( withshaft encoder reading)
   GET DUP PREVIOUS @ SWAP - COUNTER 2@
   ROT M+ COUNTER 2! PREVIOUS ! ;
: EXAMPLE ( test routine, exit with ctrl/c)  SETUP
   BEGIN   POSITION COUNTER 2@ 14 D.R CR 500 MS ?TERMINAL
   UNTIL ;
```

Reprinted from the newsletter of: Triangle Digital Services Limited, 223 Lea Bridge Road, London E10 7NE, Tel: 081-539 0285. In USA & Canada: The Saelig Company, 1193 Moseley Road, Victor NY 14564, Tel: (716)425-3753. On CompuServe: USA Sales 71042,17, other sales /technical 100065,75.

Regular Feature

ZCPR Support

Language Independence

The Z-System Corner

By Jay Sage

Programming for Language Independence

At the time I wrote my last column I was just about to depart for Germany, where -- besides vacationing -- I was going to attend ZedFest Europe 1992. I promised a full report in this issue, but I'm afraid that is not going to happen in the way I originally envisioned. After I got back, a critical situation at work kept me at the lab until after midnight nearly every night for four weeks. It was exhausting, but fortunately I got the results I was hoping for and was able to report on them at the conference from which I have just returned. With all that intense work, I had no time to think about computing, and my mind is still in a bit of a blur. I said I was going to take a roll of black-and-white photos, and I must have done it, but I have no idea right now where the pictures are. So a full photographic report will have to wait.

ZedFest Europe 1992

Despite the mental blur, I recall clearly that ZedFest Europe 1992 was as enjoyable and successful as last year's (see issue 53). Uwe Herczeg has sold his share in the computer store where we met last year and is now affiliated with a new company, whose offices occupy the second floor of the home of the parents of one of the owners. We met there this year, and it was quite an experience for me -- the house was built in the 1500s! That's something we don't come across here in the United States.

Most of last year's participants returned. Helmut Jungkunz, the organizer, was there, of course, and so was Tilmann Reh, designer of the CPU280 board. Juergen Peters again drove all the way down from Hamburg. Unfortunately, because of a misunderstanding about the date of the meeting, Franz Moessl from Italy did not make it. However, the international character was upheld by Wim Nelis, who spent 24 hours on trains to get there from Holland. He provided the Dutch for the source code I'm going to show you shortly.

Some other new people joined us as well. Matthias Voelker showed up with an incredibly souped-up Commodore C128 (8 MHz Z80H CPU, 20 Mb hard disk, 3.5" high-density floppy drive, and more); I have encouraged him to write some articles for TCJ. Alexander Schmid was there with a beautiful CPU280-based computer: one 3.5" HD floppy, two 5.25" HD floppies, an 80 Mb IDE hard drive running using Tilmann Reh's board, a Hercules-compatible graphics card, an IBM keyboard and monitor, and a large static-RAM disk. You'll hear a little more about Alexander later.

Multi-Language Programming

At last year's meeting we mostly got acquainted and talked about some issues in general terms. At this year's ZedFest, however, we got down to real work and fleshed out a new coding concept that I will share with you in this column. Last year, as I described in issue 53, we discussed the matter of international compatibility in our programming, but we never really got around to doing anything specific about it. This year we decided to write some code on the spot.

Since essentially all Z-System programs have been produced in English, native English speakers don't have much sense of the difficulties posed by programs written in a language they don't know. It became clear at last year's ZedFest that if Z-System is to become more popular in non-English-speaking countries, we must make our programs readily adaptable to other languages.

This statement, of course, is not limited to Z-System programs. Many of the large software houses, such as Microsoft, offer their products in versions for other countries. I'm not sure how they do this, but my guess is that they compile special versions for each language. I am going to describe a very simple technique that allows the same basic program to be used in multiple languages and to be reconfigured by the user. I will even describe how the Z-System might support programs that run automatically in a designated language. The PCBoard bulletin board software for MS-DOS computers (this is what I use on the DOS BBS I run for my Boston Computer Society user group) is the only program I am aware of that has this kind of flexible language adaptation.

A Sample Program

Listings 1 and 2 show two versions of a sample program, each using a conventional approach to the display of messages. Listing 1 follows the technique that was most common in standard CP/M programming. It makes use of an operating system call. The DE register pair is loaded with the address of the message string, which is terminated by a dollar sign; the C register is set to the operating system function number for printing a string.

Listing 2 uses PRINT, the in-line print routine from the SYSLIB subroutine library. I have not shown the source code for that routine (with SYSLIB you don't have to worry about the source), but here's how it works. PRINT gets the address of the string by popping the program counter off the stack. It then displays the characters embedded in the code until it encounters a null (i.e., zero) byte. Then it pushes the following address onto the stack and returns. This programming style has been very popular in Z-System programming, since the in-line message text makes the code easier to read and modify.

There are a couple of requirements that must be met to make it easy to generate programs in a multitude of languages. First, we have to gather all the messages into one place, as in Listing 1. In-line strings would be very hard (though not impossible) to modify. This is similar to what was done with configuration data in order to support the ZCNFG general-purpose configuration utility (see Al Hawley's articles in issues 52 and 53). As with the configuration data, it is convenient to put the text strings near the beginning rather than the end of the program. Second, we have to make provision for the fact that the message strings may be of different lengths in different languages.

Listing 3 shows a first cut at achieving these objectives. The messages are now placed before the actual code (and after any ZCNFG configuration block and/or an internal environment for compatibility with standard CP/M -- see my column in issue 52). Extra space is provided for each message by the ORG directives. Now, instead of the SYSLIB routine PRINT, we use the SYSLIB routine PSTR, which prints the null-terminated string starting at the address in the HL register pair. A Dutch-language overlay for the version of the program in Listing 3 is shown in Listing 4.

Before going on, let me add a note suggested by Howard Goldstein. Although I am using PSTR in my examples, the same techniques apply to the other SYSLIB printing routines, such as EPSTR and VPSTR. In fact, the examples here really should use EPSTR, which does basically what the BDOS function call does: prints a string directly to the screen. The PSTR routine has extra capability that we don't really need here. The same techniques also apply to routines that send output to a printer or even a file.

The overlay can be used in either of two ways. The source can be assembled to a HEX file and then installed over any existing version of the program using MLOAD or MYLOAD. Alternatively, the source can be assembled or loaded to absolute binary and then installed using the ZCPR3 GET command or a special utility. I will illustrate this in more detail later, where I will show some of the additional power that this approach allows. Of course, both approaches can be supported at the same time. I anticipate that source code for the language overlays will be provided with program releases.

When I proposed this solution at the ZedFest, Andreas Kisslinger had an immediate objection: it wastes space. Every message, individually, has to have extra space to accommodate the longest message that might be required in any language. He argued that it would be much better to supply a table of pointers to the messages, and I agreed. An implementation of this refined approach is shown in Listing 5, with the corresponding Dutch overlay code in Listing 6. Now we have to replace the PSTR calls with a call to a special subroutine, which we have named PRINT$. This routine first loads into DE the address stored at the address stored in HL (double indirection). Then it moves that address into HL and calls PSTR.

At this point, I had my own objection: the language overlays contained absolute addresses (the pointers to the message strings). This meant that type-3 programs (these are Z-System programs that are executed at addresses other than 100H -- see my column in issue 55) would require a separate HEX or binary overlay for each load address. Even worse, type-4 programs (these are Z-System programs whose load address is computed by the command processor at run time) could not be implemented at all. This, of course, was unacceptable.

The solution to that problem was to modify the pointer table so that it contains not the absolute addresses of the message strings but rather the offsets of each message from the beginning of the message block. Once this is done, it matters neither where in the program the message block is located nor where in memory the program is executed.

Before I show this code, I want to add one final refinement. I feel that it is a good idea (from the standpoints of safety and functionality) for modules to identify themselves. Listing 7 shows the final form of our test program. I have written this as a type-3 program with the special ID string at offset 3 into the code that identifies the program as one that has been written according to Z-System standards.

This same identification approach is applied to the language block. First, there is a leading byte with the opcode "RST 00H" (a call to address 0000). This ensures that, should there ever be an attempt to execute a language overlay, the result will be a warmboot. Second, there is an ID string of 'Z3TXT' followed by a null byte to identify the code module as a message text overlay (the RST instruction helps, too). Finally, there is a filename in the form needed for a file control block (FCB). The name portion indicates the program for which this is the overlay, and the extent portion identifies the language. I anticipate that the binary language overlay file will be given this name. The final Dutch overlay is shown in Listing 8.

Automatic Language Adaptation

This approach affords the possibility of automating the language adaptation and configuration process further. For example, I assembled Dutch and German versions of the overlays for the test program into the files TEST.DUT and TEST.GER (in binary form, not HEX). I renamed the complete English version of the program (a type-3 version with execution address 100H) to TEST.BIN and wrote an alias TEST with the following script:

```
get 100 test.bin
if null $1
  go
else
  if exist test.$1
    get 180 test.$1
    go
  else
        echo L%>anguage overlay
%<TEST.$1%> not found.
  fi
fi
```

I could then enter the command "TEST" to run the program with English messages or "TEST DUT" to run with Dutch messages or "TEST GER" to run with German messages.

There are still more ambitious possibilities, but I did not have time to write code to test the concept. However, I think, this might be the ideal way to implement language independence. A byte in the Z-System environment descriptor (the Z-System memory module that contains a description of the operating environment -- see my column in issue 54) could be set to indicate the language to use. For example, 0 might indicate English; 1, German; 2, Dutch; etc.

One approach would be to include in the program code for loading its own overlays. That code would check the environment's language byte and the currently installed message text code. If a new language needed to be installed, the name of the file could be generated from the current overlay and the language byte, and then that overlay could be loaded from disk. Here is a first cut at pseudo-code to perform the task.

```
    Get the language byte from the ENV and translate it to text (e.g., ENG, DUT, etc.).

    Compare to currently installed language as indicated by the filename in ID string in the program.

    If installed language = specified language, then start executing program.

    Otherwise search for the appropriate language overlay file.

    If not found, display an error message in the current language and either terminate or (preferably, I think) run with the current language.

    Otherwise, load the language overlay to the appropriate address.

    Run the main program code.
```

Alternatively, an extended command processor could be used to load programs, or the command processor could be enhanced to install language overlays. Since these tools would not already know where a language overlay (if any) is stored in a program, it would have to search for it. This is where the ID string 'Z3TXT' becomes essential. The following pseudocode would be added at the beginning of what we showed above:

```
    Scan from start of program looking for the ID string C7,'Z3TXT',0.

    If not found, assume that the program does not support multiple languages, and just run it

    Remember (1) the address where the overlay starts and from the header (2) the file name and (3) the language.
```

If this scheme were implemented, it would be possible to run a remote access system (a Z-Node) that would allow a user to specify his/her language preference and then to have programs run automatically in that language. However, typical users will probably be satisfied (and served more efficiently) by installing the overlay for their language directly into the program's COM file.

A Call For Assistance

There are lots of programs in existence that need to be converted to this new multiple-language capability. If you are the author currently maintaining a program, I ask that you try converting your program to the form I have described, as shown in Listing 7. For programs that are not currently being maintained by anyone, I invite any programmer to take a shot at making the conversion (you can send me the code on a diskette or via GEnie mail or my Z-Node if you would like me to check it out).

The process is not terribly difficult for most programs. You have to perform three basic steps: (1) the code where the messages are sent to the screen or printed have to be converted to calls to PRINT$ (or a similar routine for printer output); (2) the messages have to be collected into a module along with a pointer table; and (3) the subroutine PRINT$ (and possibly a similar routine for printer output) has to be added to the code. If you assemble and link the modified version and it works, chances are very good that you did it right.

Alexander Schmid, who was at the ZedFest, has already tackled one of the most difficult programs: ZFILER. Helmut Jungkunz sent me the code, which can be converted very neatly between English and German. I want to make a few minor changes, but by the time this column reaches you, the new version should be out on Z-Nodes and ZSUS diskettes.

Post-Script

Having just mentioned ZSUS, the Z-System Software Update Service, I would like to inform people who have subscribed and have been wondering where their disks are, that I hope to have the service back in operation shortly. When Chris McEwen was forced to give up TCJ, he also had to discontinue ZSUS, and my schedule has not afforded me the time to sort through all the records and get the production reorganized. That will be happening soon, and I hope there will be a flood of programs with internal ENV modules for operation under standard CP/M and language modules for flexible multilingual operation.

For those who want to see how NOT to do this (language independence), try Microsoft Systems Journal Nov-Dec 1991. These 30 pages show the prob-

lems you will need to overcome to try the same idea in Windows. I will also ask some of TCJ's Forth writers to show how simple it can be done in Forth. BDK

Listing 1

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

cr          equ   13
lf          equ   10
bdos        equ         0005h       ; BDOS entry address
printf      equ         9
            ; BDOS function code for message display

; --------- Here is the program code

            ld          de,hello
            ld          c,printf
            call        bdos

            ld          de,done
            ld          c,printf
            call        bdos
            ret

; --------- Here are the messages

hello:      db          'Hello, world!',cr,lf,'$'
done:       db          'Program finished',cr,lf,'$'
            end
```

Listing 2

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

extrn       print
; SYSLIB routine for in-line messages

cr   equ   13
lf   equ   10

            call        print
            db          'Hello, world!',cr,lf,0
            call        print
            db          'Program finished',cr,lf,0
            ret
            end
```

Listing 3

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

extrn       pstr                    ; SYSLIB routine
cr   equ   13
lf   equ   10

st0:        jp          start
; ZCNFG and ENV data here
; --------- Here is the messages block

org         st0+80h

hello:      db          'Hello, world!',cr,lf,0
            org         hello + 25
done:       db          'Program finished',cr,lf,0
            org         hello+80h

; --------- Here is the program code
start:      ld          hl,hello
            call        pstr
            ld          hl,done
            call        pstr
            ret
            end
```

Listing 4

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

cr          equ         13
lf          equ         10
            org         180h
hello:      db          'Hallo wereld!',cr,lf,0
            org         hello + 25
done:       db          'Programma is beeindigd',cr,lf,0
            end
```

Listing 5

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

extrn       pstr                    ; SYSLIB routine
cr   equ   13
lf   equ   10

st0:        jp          start
; ZCNFG and ENV data here
; --------- Here is the messages block

            org         st0+80h
msgs:
hello$:     dw          hello
done$:      dw          done
hello:      db          'Hello, world!',cr,lf,0
done:       db          'Program finished',cr,lf,0

; --------- Here is the program code
            org         msgs+80h
start:      ld          hl,hello$
            call        print$
            ld          hl,done$
            call        print$
            ret

print$:     push        de          ; Save DE
            ld          e,(hl) ; Load address at HL into DE
            inc         hl
            ld          d,(hl)
            ex          de,hl
                        ; Get message address into HL
            pop         de          ; Restore original DE
            jp          pstr
                        ; Now let PSTR print the message
            end
```

Listing 6

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

cr          equ         13
lf          equ         10
            org         180h
msgs:
hello$:     dw          hello
done$:      dw          done
hello:      db          'Hallo wereld!',cr,lf,0
done:       db          'Programma is beeindigd',cr,lf,0
            end
```

Listing 7

```
---------
extrn       pstr                    ; SYSLIB routine
cr   equ   13
lf   equ   10

st0:        jp          start
            db          'Z3ENV'     ; Z-System ID string
            db          3           ; Type-3 program
            dw          0    ; Space for ENV address
            dw          st0         ; Load address

; --------- Here is the messages block
org         st0+80h
msgs:       rst         00h
            db          'Z3TXT',0
;           '--------'
            db          'TEST    '  ; Program ID
;           '--'
            db          'ENG'       ; Language ID
hello$:     dw          hello-msgs
done$:      dw          done-msgs
hello:      db          'Hello, world!',cr,lf,0
done:       db          'Program finished',cr,lf,0

; --------- Here is the program code
org         msgs+80h

start:      ld          hl,hello$
            call        print$
            ld          hl,done$
            call        print$
            ret
print$:     push        de     ; Save DE
            ld          e,(hl)   ; Load offset at HL into DE
            inc         hl
            ld          d,(hl)
            ld          hl,msgs    ; Get starting address
            add         hl,de
                        ; Add to get message address
            pop         de       ; Restore original DE
            jp          pstr
                        ; Now let PSTR print the message
            end
```

Listing 8

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

cr   equ   13
lf   equ   10
            org         180h
msgs:       rst         00h
            db          'Z3TXT',0
;           '--------'
            db          'TEST    '  ; Program ID
;           '--'
            db          'DUT'       ; Language ID

hello$:     dw          hello-msgs
done$:      dw          done-msgs
hello:      db          'Hallo wereld!',cr,lf,0
done:       db          'Programma is beeindigd',cr,lf,0
            end
```

In Issue #58

- Multitasking in Forth 83
  Brad Rodriquez explains the ins and outs of mulitasking.

- C based program to Calculate Monostable and Timer Values
  Clem Pepper gives us a simple C program so we can put away all those complex charts.

- Z-System, REAL computing, and more!

TCJ is looking for articles that cover the following:

- IDE drive specifications and software protocols.

- YASBEC interfacing project reports. Tell our readers what you did and how it works.

- Network interfacing from CP/M or Z80 systems. We have many readers who would like to hook their classic systems into existing LANs. Let us know if you have done this or have a vendor who can.

- Special ports. TCJ readers want to hear from those who have ported products to different systems. Have you ported MINIX to S-100 or a Radio Shack Model 16? Did you put F83 in ROM? ZCPR on a TRS-80?

- Cross platform development. Have you used your classic system to do 68HC11 development. Tell our readers about your problems and how you overcame them.

Regular Feature

Intermediate

CLASSIC Support

Dr. S-100

By Herbert R Johnson

The editor, Bill Kibler, and I have corresponded about what amateur computerists want and need to support their "older" or "antique" computers, while providing something of merit for new amateurs and new developers. One of my special interests, professionally and personally, is in the first generation of a series of microcomputers called "S-100" systems. To kick off, I'll introduce you to buses in general, and the S-100 in particular for "historic" reasons. I'll suggest why you should be interested, and begin to discuss historic differences between S-100 computers. I'll try in each column to give sources for S-100 stuff. Then, it's *your turn* to write to me with questions and sources of more computers, software and docs!

In the beginning, the Bus

In the mid 1970s, a number of small companies built the "first generation" of personal computers. Like their predecessors, the minicomputers, these computers consisted of a box of electronic cards each of which shared a common set of signals. A consistent definition for these signals, along with a description of the "logical" operations associated with each signal, and a physical description of the connector and connections, is generally referred to as a computer's "bus".

Why is a bus useful? A designer of hardware, armed with the bus specification, could design a new board not only to fit the bus, but also to "fit" the device connected to this new card. Also, other developers could design "standard" cards for standard devices such as terminals and printers. Finally, cards with limited and older technology (like small memory chips) could be replaced with expanded and newer technology (larger or faster memory chips).

There were several "bus" designs during the 70s, even as there are several today. Each bus provides some paths for data, some paths for addresses (locations for data and instructions), and control signals to determine the timing and operations on that data. So, why should I rave about the S-100 bus? First, it was one of the most common and enduring bus designs of the 70s and 80s. Translation: there are a lot of surplus S-100 systems out there! Second, the technology of that period - the digital chips - are simple, well documented, cheap, and generally available even today. Third, they are slow enough (by today's standards) that most electronic hobbyists can buy or build inexpensive test equipment to maintain or even redesign S-100 systems.

Fourth, S-100 systems are still powerful enough computers to do reasonable tasks; yet not so powerful as to be unlearnable. Fifth, most of those systems were intended to be "learning systems" (microcomputers were new to EVERYONE then) so they were thoroughly documented. Finally, the software and hardware universe for 8080/Z80 based S-100 systems is fairly stable: a lot of old but useful software, yet a handful of recent developments (Z-system for instance) that combine to offer something for every taste.

Like many engineers and technicians in the 1970s and early 1980s, I learned about affordable computer hardware and software on the early IMSAI, Altair, Cromemco and other computer manufacturers who based their systems on the S-100 bus. Today's new engineers, techs and hobbyists can also use these systems for learning and development, but they will need some help in finding the old docs, hardware and software, not to mention the new software and technology being developed today.

This column, with the help of your cards and letters, can provide a common ground for all of us who are still hard at work in our basements with these beasts which, like the extinct dinosaurs, still fascinate many of us. If you aren't interested in S-100 systems as such, you might find that this computer "bus" is enough like your favorite computer that you can learn from, and contribute to, our discussions.

Send your letters to me directly, either by mail or the electronic addresses given at the end of my column, and I'll try to respond directly to you. I'll assume your letters can also be printed in this column unless you say otherwise, so we can all benefit. In the meantime, I'll start the column with a quick rundown of S-100 "standards".

S-100 "standards"

Now that I've raved about the benefits of a common bus, the first thing you must know about S-100 cards is that they are NOT all alike! The "standard" was initially whatever pins were used by the Altair computer (from MITS). It evolved as the number of manufacturers grew, and stabilized when a real, written-up and voted-upon standard was approved by a working group of the IEEE (Institute of Electrical and Electronic Engineers). Their standard, called "IEEE-696", was also adopted by ANSI (American National Standards Institute). Between the first Altair computer in 1976 and the last of the Compupro's in the late 1980's, I find it convenient to think of three flavors of S-100 cards:

IMSAI/ ALTAIR compatible cards were based on the original design by MITS, Inc. that appeared in Popular Electronics back in 1976. One hundred pins, 50 per side, with 8 bits of data into the processor, 8 bits out (yes, separately!), and 16 bits of address for 2**16 or (about) 64K memory addresses. A number of signal pins were provided for processor, I/O and memory operations; several lines for *front panel operation* (this is critical!); and several for interrupts, processor halt, and processor wait. Altair boards had some additional lines for memory enable and disable.

These cards were designed for 1 MHz to 2 MHz 8080 and Z80 operation. Floppy disk controllers were designed from individual "discrete" logic for 8" drives. No hard disk controllers. Memory cards were only a few "K" (thousand bytes) or even less. Most of these cards are too antique, even for our antique computers!

Post-IMSAI compatible cards (Cromemco was a typical manufacturer of such cards) retained most of the above lines except for front panel operations, particularly *single step*. Specifically, pins 20 and 60 were grounded, which for me defines this class of cards. These pins must be *un-grounded* to use these cards on a front-panel based system. I simply tape over the pins on the connector, rather than cut traces and pads.

Some of this class of cards have useful bank-switching memory and/or re-addressing schemes which "stretch out" the 64k memory space for RAM and ROM use. These cards can run at 2MHz, perhaps at 4MHZ. 68000 cards and 5-inch and 8-inch class hard disk controllers were available, and 5-inch floppy controllers as well. Many used fairly sophisticated 40-pin controller chips for floppy and hard disk drives. Even later versions combined these functions onto the processor card, creating a *single board* computer that could still operate all your old "peripheral" cards.

IEEE-696 compatible cards (Compupro is one manufacturer of commonly-found cards) used the original data lines for "double duty" as 16 bit bi-directional data paths. They also added another 8 lines of address for a total of 24 address bits. The standard described the appropriate bus signals for switching processor cards or "bus masters" which could also include "temporary" masters (take control of bus from another CPU card.)

These cards do not work conveniently in front-panel IMSAI-based systems, but they typically run *fast:* 4Mhz, 6MHz, or better. These cards were designed in the mid-80s and are now becoming surplus. They include cards with one to two megs of RAM, RAM disk, 16 or 32-bit processors (68020, 80286, even 80386), and "slave" processor cards.

Moral for today

Whenever possible, stay within *one class of S-100 technology*. Better still, stay with the *same manufacturer*! Incompatibilities, if not lack of detailed documentation and source code, makes "mixed" systems a challenge only for the experienced! Of course, if you are experimenting, dig in and *learn the differences!*

Sources:

Herb Johnson. As far as I know, I'm the only regular vendor of *S-100 cards* around, and I'm getting kind of skeptical. Send me an SASE (or at least a stamp!) for my catalog. I typically sell cards and disk drives, but I may have complete systems. Hard disk controllers are rare, but I might design a SCSI or IDE card if enough people ask for it (hint) at the right price (hint). *Documentation* is a problem, as some vendors are still around and thus retain copyrights. No problem with IMSAI and Altair, as they are long dead, and I have some other stuff as copies. *Software* is in the same boat, except I can sell some legitimate CP/M 2.2 diskettes, and some bootable disks as well.

IEEE-696 standard. Its full name is IEEE Std 966-1983. Contact the IEEE Secretary, Standards Board, 345 East 47th St., New York NY 10017. Check your library for the magazine *IEEE Computer*, July 1979, for a "rough draft" version. Contact me first, however, as I might have a source for a book on the subject.

Lambda Systems. While not strictly a S-100 vendor, David McGlone is a Z-80 "specialist" across several hardware platforms, and sells some commercial software and CP/M bootable diskettes. His journal, *Z-Letter,* is reorganizing and he should be encouraged: call or write him for details. Hint: he has a 2-part series on the IMSAI/Altair S-100 bus specification among his back issues.

And, of course, check the advertisers in Computer Journal.

Biography

Herb Johnson is an Electrical Engineer by training, but programs what used to be called microcomputers instead. When not chasing dump trucks to the scrap yard for those "blinking light" computers, he tinkers with CCD cameras for astronomy. Call his basement lab at (609)-588-5316; or via FidoNet 1:266.22 as Herb Johnson; or Compuserve as 70303,1024. Mail to CN 5256 #105, Princeton NJ 08543

Support Requests

To request support or assistance on a given project, please write our support people directly. TCJ's Technical Editors place their address in their columns so that you can get faster response by writing to them directly. Your requests will be included in later articles. Our editors reserve the right to refuse service or redirect you to a more appropriate source. Please include as much background information as possible.

Special Feature

All Readers

CP/M Automation

Home Automation with X10

by Rick Swenton

Here is an article that explains about using the appropriate system for the job. I think Rick shows what I have been talking about by using a CLASSIC system for modern day projects. The article also says all there is to be said about the current state of house and business controls. I do not think I have seen such a complete review as Rick provides. Great research job! BDK

HOME AUTOMATION

If you are an electronic hobbyist like me, you are likely involved in some kind of electronic wizardry in your home. Soon, you are bitten by the remote control bug. It is all around you! VCR, Stereo and TV infrared controllers, radio controlled garage door openers, photo electric and motion detectors, even people detectors for smart lighting control have invaded our homes.

Soon you discover a need to control AC line powered devices like lamps and appliances. You also wish you had switch wiring located at places you never thought you would ever need them. Enter the X10 Powerhouse system.

The X10 system is a versatile way to control lights and appliances by sending control signals through the existing house wiring. Using X10, you can control lights or appliances from anywhere in your home without the need to install special wiring. All you need is an AC outlet at each end.

Two basic parts are needed for a minimum system: A transmitting device and a receiving device. There is a wide variety of each in the X10 system. Transmitting devices come in the form of control units with ON/OFF/DIM switches as well as radio controllers, infrared controllers, clock timers and an interface unit which can be operated with a contact closure or low voltage input.

Receiving units come in the form of lamp modules or appliance modules which plug into the wall receptacle, replacement wall switches, replacement duplex outlets, and interface modules with relay contact outputs.

The transmitting devices are called "controllers". The receiving devices are called "modules".

The formal name for the system is called "X10 Powerhouse". This is the same system sold under other names such as Radio Shack's "Plug 'n' Power". Compatible systems are sold by Sears, Stanley (Lightmaker), Heathkit and X10(USA) itself.

The most versatile X10 component is the CP-290 computer interface. This is a smart interface which connects to your computer's serial port and allows you to send commands to your X10 system. Before I discuss the details of the CP-290, perhaps you might like to know more about the X10 system and its components. After you go wild installing all these X10 components around your house like I did, you will soon discover how nice it would be to interface your home to your computer with the CP-290 interface.

Here are some of the features, in detail, of the X10 system:

All devices share some common features. The X10 system supports 16 separate unit codes in 16 separate house codes to yield a maximum support of 256 modules. You could have more than one module on the same address if you desired. By X10 convention, the house codes are lettered A through P and the unit codes are numbered 1 through 16. This means that the address of a device can range from A1 through P16. The original concept of the house code was to cope with things like people in an apartment building using X10 systems fed from the same power line. Each apartment could use a different house code but each would be limited to 16 devices. The success of such a system relies on the mutual cooperation and coordination of the users. In a single home located a sufficient distance from the neighbors, the house codes could be thought of as "zones" in the single house. For example, I have each floor in my house assigned to a specific house code. This means that an "All Units OFF" command issued from a first floor controller would only turn off all units located on the first floor.

Controllers:

There are several X10 control devices which send commands over the power lines. The smallest is the MC460 Mini-Controller. This device can control 8 modules (only the first 8 unit codes, it can not address modules 9 -> 16). It has switches for "All Lights On", "All Off", "Dim" and "Bright". The SC503 Maxi-Controller has all these features too but supports all 16 addresses instead of only 8. The TR2700 Telephone Responder allows you to control your first 8 unit codes over the phone and the RC5000 controller allows you to control the first 8 unit codes by radio remote control.

For timer controllers, there is the MT522 which allows you to manually control 8 modules and timer-control 4 modules to go on or off at specific times up to twice a day. It also has a security mode to simulate random on/off operation while you are away from home. The CR512 has all of the features of the timer-controller built into a clock radio. Both have battery back-up.

Another device which falls into the category of controller is the BA284. X10 calls this unit a Burglar Alarm Interface. Radio Shack calls it a Universal Interface, and I like this name better. This unit is a small box which plugs into any outlet. You would use this unit to control X10 modules with an external voltage or contact closure. You can configure this unit to control a single module of any type. You can also configure it to transmit the "All Lights On" command in addition to a single selected unit code. Finally, you can configure it to flash all your lights on and off, such as would be the case if you interfaced it to your alarm system. This interface is very versatile.

In general, the house code is selected with a thumb wheel or screwdriver operated rotary switch on controllers. While controllers can control all 16 house codes, it is mostly inconvenient to change the house code as though it was a user-operated feature. Usually, you dial-in the house code on a controller and leave it set.

Modules:

The LM465 Lamp Module is a small box which plugs into any AC outlet and can control up to 300 watts of incandescent light. It can dim or brighten the lamp and responds to the "All Lights ON" and "All OFF" commands.

The LM486 (2-prong) and LM466 (3-prong) Appliance Modules are also small boxes which plug into the AC outlet. They control their loads with a relay so they can operate things like TV's and air conditioners, up to 15 Amperes. Appliance modules do not respond to the "Dim" or "All Lights On" commands but they do respond to the "All Units Off" command.

The SR227 is a duplex outlet which can control a 15 A load. This unit is a replacement for your existing wall outlet and unless you know what you are doing, you would be better off enlisting the help of an electrician. Like the appliance modules, the duplex outlet module does not respond to the "Dim" or "All Lights On" commands but it does respond to the "All Units Off" command.

The WS467 and WS4777 (3-way) are wall switch modules. They replace your existing wall switch and can control 500 watts of incandescent lamps. Consult with your electrician on this one too. If your wall switch controls wall receptacles in the room, keep in mind that these modules can only handle light bulbs and they may overheat or could even cause damage to anything other than a light plugged into the receptacle. Like the LM465 Lamp Module, the Wall Switch Modules can dim or brighten the lamp and respond to the "All Lights ON" and "All OFF" commands.

The HD243 (15 AMP) and HD245 (20 AMP) are heavy-duty plug-in modules to control high current 220 volt loads like air conditioners.

Finally, there is the TH2807 Thermostat Set-Back. You think I meant "Set-Back Thermostat" but I didn't! This unit is actually not a thermostat. It is simply a heater. You install it on the wall under your existing thermostat. When it is connected to an appliance module, you can use your control of the appliance module to operate the heater. When the heater heats the air around your thermostat, it "fools" it into thinking that the room is warmer than it really is, so it doesn't call for heat.

Of special interest to us hardware hackers are the BA-284 Universal Interface and the BA-506 Universal Module. The Universal Interface is essentially a controller which operates with a voltage or contact closure instead of the push of a button. The Universal Module is essentially a plug-in module but instead of an AC receptacle for the load, it has two screw terminals with dry relay contacts.

Typical Applications

The X10 system can be put to work in many applications around the home or business. In my house, there are several lights which are timed to turn on and off automatically. Almost every room is equipped with an X10 wall switch. This means that the room lights can be turned on and off from the wall switch and from any X10 controller. Having an X10 controller located at the exit door allows me to use the "All Units Off" command to be sure the kids didn't leave their bedroom lights on. It saves me a trip to the second floor. Another application is a timed vacation schedule which is programmed to make the house appear to be occupied because the lighting in every room closely -- but not exactly -- resembles our normal activity. The timer security mode varies the timed events each time they are sent. My favorite application involved stealing a commercial idea which allowed your garage door opener to also control a remotely located light. I used a BA-284 Universal Interface and connected it to the lamp control relay in my garage door openers. I used an opto-coupler on each opener lamp relay and simply connected their output transistors in parallel to form an "OR" gate. When either opener lamp comes on, the large fluorescent lamps in the garage come on too.

A microswitch on each door is connected in parallel with the opto-coupler transistors. The switches are installed to be activated by the door in the up position. When the door is up, the garage flourescents stay on, even after the opener lamp times-out.

At the office, I have an appliance module connected to the FM radio which plays music-on-hold into our phone system and background music into the office. All the equipment is located in the back room except for the X10 Mini-Controller located at the front desk. When the office staff reports to work, they can skip the daily excursion into the back room and turn on the radio from their desk.

At church, I installed an appliance module on the sound system amplifier. In addition, I installed a universal module (dry contact closure) in parallel with the output of the FM wireless microphone receiver. Our church is located at a very high elevation and the FM wireless mic receiver picks-up many other signals when our portable mic is not powered-up. Now, from several locations in the church, we can disable the wireless mic receiver if we are not using it. You may ask why not just use an appliance module on the receiver's AC power line. Well, turning on the power causes a thump in the sound system as the receiver comes on. Putting the dry contacts in series with the audio leaves an open circuit (noise potential) at the amplifier input. Placing a short on the audio definitely mutes the audio from the receiver as well as any potential noise.

In reality, X10 applications are unlimited. You can turn on your sprinklers to water your lawn. You can control your heating and air conditioning. Using a special X10 interface, you can send and receive X10 commands using a dedicated controller such as the one Jay Sage has been describing. This interface, called the TW523, consists of only the very basic circuits to interface to the power line. Your host software must to all the timing in order to send or receive X10 codes. The nice part is that this box plugs directly into the wall receptacle. You connect to it with a modular phone jack. It isolates your controller safely from the power lines. Purchase of the TW523 gives you permission to use the proprietary X10 format signals.

One of my personal goals is to develop a custom system, similar to Jay's, but use X10 as the primary access to AC power control. I also envision a CRT screen used as a status board to scroll through displays of heating/air conditioning, lighting, and watering. It will also have a phone line interface to allow commands to be received remotely or even entered from any phone in the house. It will have a clock to schedule programmed events including security lighting and temperature set-backs.

Ken Davidson authored two articles about the power line interface. They appeared in the May/June 1988 and September/October 1988 issues of Circuit Cellar Ink. These articles are essential if you intend to integrate the interface into a dedicated controller system.

The Ultimate X10 Controller for Computer Junkies - The CP-290

The CP-290 is an intelligent interface to control your X10 modules with your computer. It connects to your serial I/O port and communicates in a manner very similar to an intelligent modem. Using the CP-290 you have the ability to turn units on or off directly from your computer keyboard as well as the ability to program the CP-290's event timer. There are 128 individual programmable events which can be set to turn things on or off an any predetermined time for selected days of the week. The CP-290 maintains an internal 24-hour 7-day clock. You do not need to keep your computer on or even connected to the interface. The interface will keep on transmitting the programmed event information over the power lines at the correct times. There is an internal 9-volt battery to maintain the programmed information during power failures.

The CP-290 comes with software. Only Apple-IIe, Commodore, Macintosh or IBM-PC software is available from X10. Other computers are supported by private companies but none, to my knowledge, have any commercial software available for use under CP/M. This in understandable since each CP/M computer accesses its serial I/O ports differently. At the time I purchased my CP-290, I only owned CP/M computers so the IBM or Macintosh software was of no value. It was obvious that if I wanted a comprehensive software package to run my CP-290 interface on CP/M systems, I would have to write one myself.

If you looked at the Programming Guide which came with the CP-290, you would quickly realize that it is no simple matter to write a program to control the interface. The microprocessor in the CP-290 is a relatively simple controller with only 8K of RAM. The firmware command instructions are not consistent. All have a different number of bytes and most bytes have bit-mapped definitions. This means that for a program to talk to the CP-290, not only must it know how many bytes to send, it must also know which particular bits within certain bytes must be on or off.

Eventually I was able to develop a series of programs in Z80 assembly language to access the features of the CP-290. As time went on, I consolidated the individual utilities into one single program. Finally, with the help and encouragement from Al Hathway and Biff Bueffel, we honed and polished X10.COM into a full-featured program for CP/M and Z-System.

To give you a taste of the program, see the Menu Display on the next page.

X10.COM can run in menu mode or from the CP/M command line. Here is an example of a typical X10 command entered from the command line. This command will turn-on a light with address D12 and dim the lamp to level 9.

```
X10 NOW D12 DIM 10
         |   |   |
        (1)  (2) (3)
```

(1)-- House/Unit Code - A ->P, 1 -> 16 or ALL
(2)-- Function - ON, OFF or DIM
(3)-- Dim Level - 1 (dim) --> 16 (bright)

The CP/M command line is an ideal way to pass all the X10 commands to the CP-290 interface through ZCPR3 Aliases or other scripts.

It should be noted that the Libraries were used extensively during the development of X10.COM. SYSLIB, VLIB, Z3LIB, DSLIB and ZSLIB provided routines which greatly enhanced the power and versatility of X10.COM without having to re-invent the wheel at every turn. The Libraries made programming in Assembly Language a pure joy! This allowed us to create a program which ran well under ZCPR3 / NZ-COM as well as plain CP/M.

The hardware specific software for X10.COM is contained in an overlay file similar to IMP and MEX. We have provided overlays for many popular systems as well as a generic overlay to help you create a custom installation. The overlay also contains a spot for the terminal definitions. The terminal definitions are only required to run X10.COM under plain CP/M. We have an internal environment and TCAP in X10.COM because we wanted to create ready-to-run COM files for Heath and Kaypro which would run under plain CP/M. If you are running ZCPR3, X10.COM will use your currently defined terminal in the ZCPR3 TCAP.

X10 Mega-Systems

At the very high-end of X10 systems are commercially available products not provided by X10 USA but which use all X10 products as system accessories. These systems are usually found in the million-dollar mansions with the separate Audio/Video room which has electric drapes on the wall screen for the projection TV and a custom built 200 slot laser disc juke box.

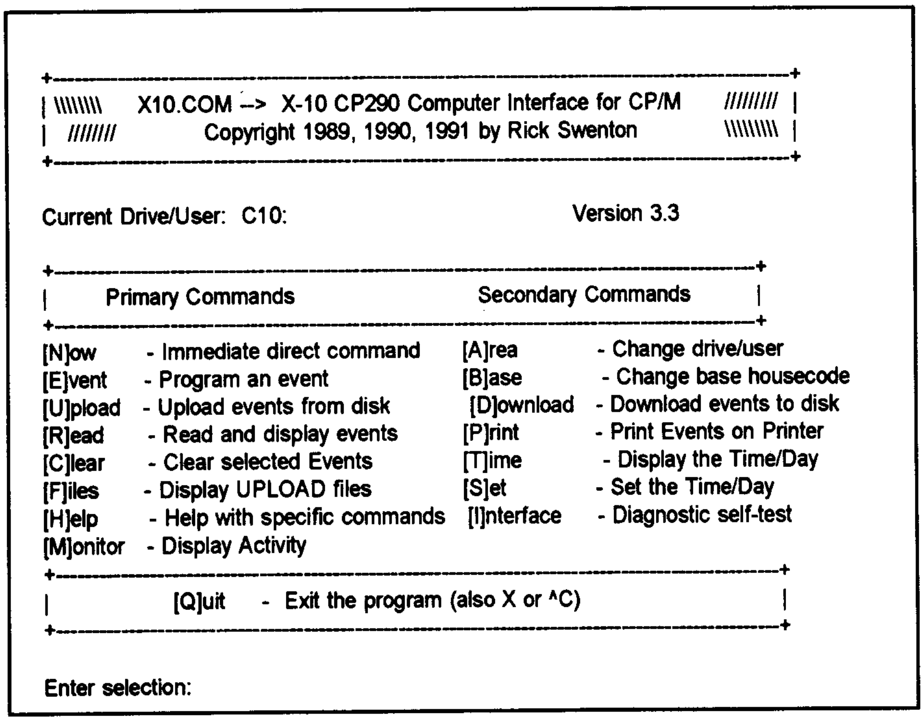
The Enerlogic ES-1400 is a microprocessor based intelligent home control system. It uses an IBM (tm) compatible PC as the human interface but does not need the computer to be on or even connected to perform automated tasks (just like the CP-290). What's special about the ES-1400 is that it is a TWO-WAY INTERFACE. It can also receive commands from other controllers and then make an intelligent decision based on the command just received. This open's the door to macro commands where the touch of one button somewhere in the house will cause the ES-1400 to "remap" that command into a series of other commands which can occur now or at some time in the future.

Some of the suggested applications are to use motion sensors to turn-on selected room lighting when you are home but have them activate the alarm system when you are away.

The Enerlogic ES-1400 sells for about $370.

The JDS Telecommand System 100 is a microprocessor controlled telephone interface. This unit will allow you to send any X10 command from any Touch-Tone (tm) telephone. This includes all phones inside your home as well as dial-up from outside. Think about it. You could call your home from the cellular car phone and start the air conditioning on that hot summer day while you are on your way home from work!

The Telecommand System 100 sells for about $450.



```
+----------------------------------------------------------------------+
| \\\\\\\    X10.COM --> X-10 CP290 Computer Interface for CP/M  ///////// |
|  ///////        Copyright 1989, 1990, 1991 by Rick Swenton     \\\\\\\\\ |
+----------------------------------------------------------------------+

Current Drive/User:  C10:                          Version 3.3

+------------------------------------------------------------------+
|     Primary Commands                    Secondary Commands        |
+------------------------------------------------------------------+
[N]ow      - Immediate direct command    [A]rea      - Change drive/user
[E]vent    - Program an event            [B]ase      - Change base housecode
[U]pload   - Upload events from disk     [D]ownload  - Download events to disk
[R]ead     - Read and display events     [P]rint     - Print Events on Printer
[C]lear    - Clear selected Events       [T]ime      - Display the Time/Day
[F]iles    - Display UPLOAD files        [S]et       - Set the Time/Day
[H]elp     - Help with specific commands [I]nterface - Diagnostic self-test
[M]onitor  - Display Activity
+---------------------------------------------------------------------+
|           [Q]uit   -  Exit the program (also X or ^C)                |
+---------------------------------------------------------------------+

Enter selection:
```

Starting at $2,000, Home Automation Inc. has the system for you. This is a microprocessor controlled "whole-house" system. This single unit provides two-way X10 interface, scheduling, heating/air-conditioning interface with temperature control, macro commands, 21-zone coverage for fire, burglary or emergency, alarm system with auto dialer and digital speech messages (name, address, type of emergency, etc). It also has local and dial-up telephone control.

The Future

X10 is the simplest way to control things without installing wiring by using existing AC power lines. The X10 system has been around for many years, so it is a proven system. However, X10 is not compatible with the new CEBus standards which are emerging today. CEBus (Consumer Electronics Bus) is a new way for devices in the house to communicate with each other, regardless of the manufacturer. CEBus can use power line, phone line, coax, infrared, RF or any combination to communicate. On the power line, it uses a scheme similar to X10 but at a higher speed and capacity. With CEBus, you could in theory program a scheduled event into your VCR from the front panel of your microwave oven! CEBus opens the door to phones, thermostats and alarm switches sharing the twisted pair wiring, appliance control sharing the power line wiring, and the audio/video sharing the coax wiring with routers interconnecting the three media and translating events between them. Of course, the consumer products need to be manufactured with the CEBus capability built-in to take advantage of the system.

Compatibility with CEBus could diminish your X10 investment in the near future. However, there are clever people out there who would enjoy developing a bridge between X10 and CEBus. It shouldn't be that hard to do.

Conclusion

The X10 system is a powerful and versatile system to control lights and appliances around the home without the need to install special wiring. The CP-290 Interface makes it easy to control your home by programming events on your PC clone or better yet, your new YASBEC system! Once you get started with the basic X10 system, it is very difficult to break the habit of expanding the system on a weekly basis. It took a little time for my family to acclimate to the unconventional light switches and power controllers. Now, it is a way of life.

Publications:

Electronic House
747 Church Road, G-11
Elmhurst, IL 60126-1420
(219) 256-2060
$14.95 per year MC/VISA/AMEX

Circuit Cellar INK
4 Park Street Suite 20
Vernon, CT 06066
(203) 875-2751
$17.95 per year MC/VISA

Other:

There is a special user area in GEnie dealing with Home Automation. GEnie is an on-line subscription computer service. In the Home Automation area in GEnie, you will find a roundtable -- a two-way dialog of people interested in home automation as well as computer programs available for downloading.

To sign-up for GEnie, call client services at 1-800-636-9636 or write

GE Information Services
401 North Washington Street
Rockville, Maryland 20850

The program X10.COM is available for downloading on GEnie. Version 3.3 is called X10-C33.LBR which contains ready-to run COM files for Heath, Kaypro, Ampro and SB-180 and the Users Manual. X10-S33.LBR contains the source code files including the overlays and extended TCAP files.

X10.COM is also available on the TCC BBS (203) 673-8752, Jay Sage's BBS (617) 965-7259 and several RCP/M systems on the west coast.

Biff Bueffel is currently preparing release 3.4 of X10.COM and should be available "real soon now".

The following notes pertain to the Cross-Reference table:
* DAK is currently having financial problems and may be in the midst of bankruptcy proceedings. It's possible that some good deals may be available

Cross-Reference of X10 Products

Product	X10	HeathKit*	Radio Shack	Crutchfield	DAK*
Lamp Module	LM465	BC-465	61-2683	009LM465	9779
Appliance 2 prong	LM486	AM-486	61-2681	009AM468	
Appliance 3 prong	LM466	BC-466	61-2684	009AM466	
Wall Switch	WS467	BC-467	61-2683	009WS467	9780
Wall Switch 3-way	WS4777	BC-4777	61-2686	009WS4777	
Wall Receptacle	SR227	BC-227	61-2685	009SR227	
220V 15A Appliance	HD243	HD-243			
220V 20A Appliance	HD245	BC-245			
Thermostat Set-Back	TH2807	BC-2807			
Mini Controller	MC460	MC-460	61-2677A		
Maxi Controller	C503	BC-503	61-2690	009SC503	4622
Radio Controller	RC5000	BC-5000	61-2675	009RC5000	4712
Telephone Controller	TR2700	BC-2700			
Universal/Alarm	BA284	BC-284	61-2687		
(Interface external system inputs such as alarm systems)					
Universal Module		UM-506			5690
(Interface external system outputs such as sprinklers or drapery openers)					
Whole House Control		URC-5000			
(Infrared hand-held VCR-style remote control for X10 and TV/VCR/Stereo, etc)					
Command Center		URC-3000			
(The receiver portion needed with the hand-held URC-5000)					
Sensor Chime		SL-5321			
Timer	MT522	MT-522	61-2679	009MT522	4973
Timer/Radio	CR512				
Computer Interface	CP290P	BC-290P	905-2087	009CP290P	

now or in the future. There's also some risk.

HeathKit has ended production of electronic kits. At this time, they are still in the home automation products business.

The Computer interface CP290P comes with a serial cable which can directly connect to the H89 modem port 330. The other H89 serial ports can be used but the gender of the connector needs to be changed from female to male and pins 2 and 3 need to be swapped.

You can order any product directly from X10 (USA) Inc. They accept major credit cards. You can also stop into your local Radio Shack store for most of the common products.

X10 (USA) INC.
91 Ruckman Road
Closter, NJ 07624-0420
(201) 784-9700
(800) 526-0027

DAK Industries
8200 Remmet Ave.
Canoga Park, Ca 91304
(800) 325-0800 - Order
(800) 888-9818 - Tech Info

Crutchfield
1 Crutchfield Park
Charlottesville, VA 22906
(800) 446-1640

HeathKit
Benton Harbor, MI 49023
(800) 253-0570

Radio Shack
1500 One Tandy Center
Forth Worth, TX 76102
(817) 390-3011

Home Control Concepts
P.O. Box 27983
San Diego, CA 92198
(800) 828-8537 Order (cards ok)
(619) 484-0933 Info and Support
HCC is a high volume distributor carrying a wide variety standard and EXOTIC X10 Products. They do have a $100 minimum order but prices are low!

CLASSIFIED and FOR SALE

For Sale: Televideo terminal boards. Have 3 of 950 and 1 of 925, still in working order. I am willing to remove all electronics from the case and ship to you. $25 each, plus UPS shipping fees. Call Bill at *TCJ*, (916) 645-1670.

For Sale: HP2621 terminal with printers. These HP terminal were working, but recent moving has damaged some cabling. Will take out all boards and ship to you for $25 each, plus UPS shipping. Call Bill at *TCJ*. (916) 645-1670.

For Sale: Lexidata 3400 RGB CAD system. High resolution graphic station without any docs. Will sell boards (6809 video CPU) or entire system. Can ship boards, but cabinet and power supply too heavy. Entire system includes monitor and would require motor freight, $250. Small parts or boards are negotiable (else will part it out). Call Bill at *TCJ*. (916) 645-1670.

For Sale: GIMIX 6809 SS-50 floppy disk controllers. Have six to sell at $25 each. These are like new and docs maybe had for a fee. An entire system is available with software and documents. If interested contact Bill at *TCJ*. (916) 645-1670.

The Computer Journal Classified section is for items FOR SALE. It is priced and setup the same as *NUTS & VOLTS*. If you currently have an ad running in *NUTS & VOLTS*, just send us a copy of the ad and your invoice from them, along with your check, and we will publish it in the next *TCJ*.

CLASSIFIED ads are on a pre-paid basis only. The ad rate is $.30 per word for subscribers, and $.60 per word for others. Those rates are for the first 150 words and double beyond 151 words. There is a minimum $4.50 charge per insertion. Photos and drawings are extra. Contact *TCJ* for details (or see an issue of *NUTS & VOLTS*.)

SUPPORT WANTED

Wanted: SAGE/Stride CP/M68K or UNIX software. Have a SAGE IV, (actually II but labeled wrong - a real classic) with 500K of memory. Also looking for hard drive for same. Want source code for CP/M BIOS and BOOT disk. Have entire P-System with source for that BIOS and ROMs. Complete set of DOCS and will copy what you might need in exchange for software and or support. UNIX may not run on my system, but feel that I need to collect this information before it disappears for good. Call Bill at *TCJ* (916) 645-1670.

Wanted: Help with NORTHSTAR ADVANTAGE, these hard sectored disk systems are still running, but hard to get disk for. Need modem program and other support software on hard sectored disk. Any modifications to go soft sector out there??? Contact Jack care of *TCJ*. (916) 645-1670.

Wanted: Software and advice on HP 125. Is there anybody in my area that can help me? How about local clubs or organizations? Leslie Jones, 1206 Overdale Rd., Orlando, FL 32825, (407) 240-2189.

The Computer Journal SUPPORT WANTED section is a FREE service for our readers who need to find old or missing documentation or software. No FOR SALE items allowed, however exchanges or like kind swapping allowed. Please limit your request to one type of system per entry. Provide the make, model number, and vintage when possible. A phone number or BBS contact is recomended as well as home or business address. *TCJ* reserves the right not to print any request it feels that does not meet the guidelines or is more an classified ad than a request for support.

Communications

All Readers

Modem Transfers

File Transfer Protocols

by Steven G. Westlund

Here is a great article that covers all anyone needs to know about transfer protocols. Steven does such a great job, I have used this as a handout for my data communications course. You might think you have already learned all there is about protocols, but I think you will be surprised by some of his information. BDK

COMPARISON OF FILE TRANSFER PROTOCOLS FOR MICROCOMPUTERS

If you venture into the world of telecommunications with your microcomputer, you probably take advantage of the vast array of public domain software and shareware available on bulletin board systems (BBSs) across the country. You may have discovered that a significant amount of connect time can be consumed in transferring those files to and from your system. It can become rather expensive when dialing long distance or using an on-line service. If this sounds familiar, take a closer look at your file transfer protocol options. It just might save you time and money.

FILE TRANSFER BASICS

File transfer can be defined as the process of sending a file over a communications line from one computer to another. Files can consist of text characters, binary data or program source code. The computers involved can range from home and office micros to minicomputers and mainframes used by businesses, government and on-line services.(1)

In a typical file transfer process, the sending computer reads a file from a mass storage device, such as a floppy disk, fixed disk, or RAM drive, and transfers the data through a serial port to the sending modem. (The word "modem" means modulator-demodulator). The sending modem converts the data from digital to analog form and transmits it over a communications line, usually a voice grade telephone line, to the receiving system. The receiving modem accepts the data and converts it from analog back to digital form. This modem transfers the data through a serial port to the receiving computer, which stores it in a file on a mass storage device on that system.

File transfers can take place in two directions. Uploading is the process of sending data to another computer. Downloading is the process of receiving data from another computer. Both systems must use the same protocol, which is the set of rules that govern the exchange. These rules define the handshaking and data formats that manage error detection and correction. In other words, the protocol indicates when to transmit, when to receive, and how to communicate to the other system that something wasn't understood.(2)

ASCII FILE TRANSFER

An ASCII file transfer uses the simplest type of protocol. Commonly invoked to transfer straight text files, it is generally limited to files containing 7-bit ASCII characters. The ASCII protocol uses transmitter on (XON) and transmitter off (XOFF) characters to signal when to start and stop sending data.

In an ASCII transfer, data is sent character-by-character to the receiving system. Data buffering can be used on both ends to speed up the process, since it reduces the times a transmission must be halted while the sending or receiving computer accesses its disk. The XON/XOFF protocol allows either system to start and stop the transfer when this occurs.

Parity checking is a type of error detection available on systems using 7-bit ASCII. The parity bit is transmitted at the end of a 7-bit character. Two types of error checking methods are employed: EVEN/ODD and MARK/SPACE.

EVEN/ODD parity determines if the sum of a string of bits should always be even or odd. An even parity check works as follows: The sending computer, when framing a character, counts the number of bits equal to one and then adds the parity bit. The value of this bit will be zero when the total is an even number. When the addition of bits is odd, the parity bit will be equal to one, to make the total even (hence - even parity checking.) So, with even parity, the total number of ones should always be even (including parity bit.) With odd parity, the total should always be odd. The receiving system checks for this result. When finding a discrepancy, it flags the data and requests a retransmission.

MARK/SPACE parity is even more basic. Mark parity always places a one in the parity bit with the receiving computer expecting it to be there. Space parity calls for a zero parity bit. There are some basic problems with this method of error detection. Parity checking is not available on systems that use 8-bit ASCII. When employed, it is not very reliable. This is because parity schemes are only successful in catching one-bit errors. Two bit errors can mask themselves.(3)

ASCII transfer is generally an acceptable protocol when transferring text over a quality telephone line. However, when telephone connections are noisy, data can be lost. This is because ASCII does not provide for the retransmission of garbled data. The ASCII protocol's primary limitation is its inability to send binary files and programs. For this type of exchange to work, the files must first be converted from binary to hexadecimal. Following transmission, they must be converted from hexadecimal back to binary. This complex procedure convinced early microcomputer users of the need for an error-detecting file transfer protocol designed to work with binary data.

THE XMODEM PROTOCOL

In 1977, Ward Christensen developed a protocol that was designed to provide reliable file transfer of text and binary files between microcomputers. The protocol, known as XMODEM, or the Christensen Protocol, was generously placed in the public domain. Since its introduction, XMODEM has been the most popular file transfer protocol used by microcomputers. It is a standard option on BBSs and in communications programs offering non-proprietary protocols.

In a XMODEM transfer, a file is sent in 132-byte frames, also called blocks. Each frame begins with an ASCII control-A mark character, followed by the block number. The data field is 128-bytes long, which is the size of a standard 8-inch CP/M disk block. For error detection, a checksum byte is added to each frame. Figure 1 illustrates a basic XMODEM block.

A file transfer is initiated when the receiving system sends a negative acknowledgement (NAK) character. The file is sent just as it is stored on disk, without conversions. The control fields, and usually the data field, are composed of 8-bit ASCII characters. The checksum is calculated by adding up the ASCII values of all the bytes in the block and dividing that total by 255. The single digit remainder is the checksum character. If the receiving system calculates the same checksum, it transmits an acknowledgement (ACK). Otherwise, it sends a NAK.

The sending computer waits for a return ACK or NAK to determine whether to send a new block or resend the last one. The transfer process normally ends when the sending system transmits an end of transmission (EOT) at the end of the file and receives acknowledgement. The transfer also can be aborted by either user, too many retries or a time-out. The checksum method of error-detection offers 95 percent reliability.(4) This is much better than the ASCII parity check, but still leaves room for improvement. It only takes one-bit error to disable a computer program.

XMODEM EXTENSIONS

A common XMODEM option, the cyclic redundancy check (CRC), improves the way the protocol checks for errors. This extension is usually identified on a BBS as XMODEM CRC, and may be offered in a communications program. Both systems, of course, must have this capability, and a program supporting XMODEM CRC can usually detect if the other system offers it. CRC is selected when the receiving system sends the character "C" to initiate the transfer. If nothing happens, it tries a few more times before sending a NAK, which initiates a XMODEM checksum transfer. The cyclic redundance check adds a second checksum byte to each block to enhance error detection up to 99.6 percent.(5)

Another extension of the protocol, called MODEM7, adds batch file transfer capability. This permits a group of files to be sent in one operation when the operator requests several files or uses a wildcard asterisk character in a file name in order to transfer all files of a particular type.

The XMODEM 1K protocol represents a significant enhancement in terms of throughput. This extension increases the blocksize from 128 to 1024 characters and adds CRC error-checking. With a good connection, XMODEM 1K can improve throughput over regular XMODEM by as much as 87 percent.(6)

XMODEM LIMITATIONS

XMODEM, with all its extensions, still has certain inherent limitations. Because XMODEM requires 8-bit control characters, it won't work over a 7-bit ASCII channel. Control fields, like the block number, would translate incorrectly. This means that XMODEM can't be used over certain public packet-switched networks, such as Telenet. It also won't work with certain host computers, like IBM mainframes, that require the use of character parity.(7)

The XMODEM block data field was designed for CP/M. On MS-DOS systems, it pads the last block in a file with null characters when a file doesn't end on a 128-byte boundary. This alters the true file length, increasing the time required for the transfer.

THE YMODEM PROTOCOL

The YMODEM protocol was developed by Chuck Forsberg, who was responsible for several XMODEM enhancements. YMODEM, actually defined by Ward Christensen in 1985, was named after Forsberg's "Yet Another Modem" (YAM) communications program.(8)

YMODEM incorporates many of the XMODEM extensions, including 16-bit CRC error-checking and 1024-byte blocks. A special version, called YMODEM Batch, performs multiple file transfers like MODEM 7. YMODEM is available on many BBSs and communications programs for both CP/M and MS-DOS microcomputers.

Be aware that not all versions of YMODEM out there are compatible, This is because of some confusion between YMODEM and XMODEM 1K. A few communications programs claim to support YMODEM when they actually offer XMODEM 1K. This can cause frustration when a transfer is canceled because the program does not send or receive the correct YMODEM pathname block.(9) However, the odds are that your YMODEM transfers will work fine. Most YMODEM implementations are compatible.

THE WXMODEM PROTOCOL

The WXMODEM protocol was created by Peter Boswell for use by People/Link subscribers. WXMODEM stands for windowed XMODEM. A windowed protocol is a full-duplex protocol that doesn't require the sending computer to wait between blocks for an ACK or a NAK. WXMODEM can send up to four blocks without an acknowledgement of any kind. The window is the difference between the block being sent and the block for which the last ACK or NAK was received. The sending computer is always one to four blocks ahead of the receiving computer. When a NAK is received, WXMODEM knows what block must be retransmitted.(10) Like the original XMODEM protocol, WXMODEM transmits 128-byte data blocks. Figure 2 illustrates a WXMODEM window.

THE ZMODEM PROTOCOL

Chuck Forsberg initially created the ZMODEM protocol for transferring files over the Telenet packet-switching network. Because of its performance, ZMODEM has become the protocol of choice by many BBSs and individual users. The ZMODEM protocol differs from XMODEM in how it deals with packet-switching networks and mainframes. ZMODEM can transmit files over a 7-bit ASCII channel. It uses 512-byte data blocks and, unlike XMODEM or YMODEM, preserves the exact file length if it doesn't end exactly on a 512-byte boundary. Like WXMODEM, ZMODEM is a full-duplex windowed protocol that does not have to wait between packets for an ACK or NAK from the receiving system. It also allows the user to increase the size of the input buffer space from the 1024-byte default to a maximum of 8192 bytes. This can result in faster transfer times, especially on floppy disk systems.

ZMODEM's advanced features include a 32-bit CRC, crash recovery, flexible control of selective file transfers, and security verified command downloading. The 32-bit CRC code, written by Gary S. Brown, is an improvement over the 16-bit version used by XMODEM and YMODEM.(11) After a broken connection is reestablished, crash recovery will resume a transfer from the last acknowledged packet. I have found this feature to be a lifesaver. Nothing seems as frustrating as downloading a large file over a long distance telephone line, only to have the connection broken right before the transfer is completed. With other protocols, you would have to dial back and start over from the beginning. With ZMODEM, once the connection is reestablished, the transfer can continue from the last acknowledged packet.

Forsberg offers a shareware version of the protocol, known as DSZ, for MS-DOS systems. Ron Murray's public domain program, ZMP, provides CP/M users with the capability. Both ZMP and DSZ can be downloaded from many BBSs across the country.

THE KERMIT PROTOCOL

The Kermit protocol was developed in 1981 by Frank da Cruz and his team at Columbia University. With a growing number of microcomputer users and the decentralization of computing responsibilities, the university was faced with the problem of how to get its mainframes, minicomputers and micros to communicate with each other. Because compatible communications packages were not available at the time for all its systems, Columbia made the decision to develop a communications protocol that would meet the needs of the university. Yes, the protocol was named "Kermit" after the famous frog of Sesame Street. Kermit also happens to be a Celtic name that means "free." This is appropriate, since it was placed in the public domain.

Kermit was initially developed for CP/M and MS-DOS microcomputers, the DECSYSTEM-20 and IBM 370-Series mainframes. Pleased with the initial results, da Cruz began presenting the protocol at computer users conferences. Columbia University openly shared the Kermit programs, source code and documentation with interested computer users. As other organizations developed Kermit for their systems, they sent the new implementations back to the university. The process continued. By 1986, Kermit was available for about 200 machines and operating systems.(12)

Kermit is an ASCII character-oriented protocol capable of transferring 7 or 8-bit ASCII files. When necessary, it can convert 8-bit characters to 7-bit. This is accomplished by translating ASCII values greater than 127 into a pair of 7-bit bytes, which can be transmitted safely. Kermit breaks the data file into packets that are normally 96-bytes in length. The packet size can vary because of changes in transmission conditions. Basic Kermit packets are composed of a header, file data and checksum.

The header begins with a control-A character marking the beginning of the packet. The next three bytes of header information indicate the packet length, the sequence number and the packet type. The length byte is an ASCII decimal value equal to 32 plus the total of the remaining characters in the packet. The sequence number byte is an ASCII decimal value from zero to 63, and is used to detect lost or duplicated packets. Sequence numbers wrap around to zero after each group of 64 packets. The type field is an uppercase letter indicating the purpose of the packet. Special packet types are used to send initialization parameters, file headers, ACKs and NAKs. They also signify the end of the file, a break in transmission, or an abort in the transfer process.

In a data packet, the data portion is followed by a block check. This single byte arithmetic checksum represents all the characters in the packet between the control-A mark and the block check. Like XMODEM, Kermit uses the checksum to detect errors. Figure 3 illustrates a basic Kermit packet. This format represents a normal Kermit packet that all programs should support. Like XMODEM, Kermit also has options that vary the length and block check fields.

KERMIT EXTENSIONS

Though normal Kermit packets are 96 characters, longer packets can be used. The sender must set this capability in the initialization packet and furnish extended length header fields. The extended header is placed between the packet type and data fields. It contains packet length and header checksum fields. The maximum length of a long packet is 1000 characters.

Normal Kermit programs send and acknowledge packets one at a time. An extension of the protocol, known as Super Kermit, employs full-duplex, sliding window capability. Like WXMODEM, it allows multiple packets to be sent before an ACK is received. Super Kermit can send as many as 31 packets before receiving an ACK.(13)

Kermit offers an optional form of data compression known as run-length encoding. This extension replaces any data character, appearing more than three times in a row, with a shorter prefixed sequence identifying the character and the number of occurrences. This form of data compression is most effective when transmitting fixed-blocked files with trailing blanks, program source code, indented outlines, or binary files with repeating zeros.

The MS-DOS and CP/M versions of the Kermit can be downloaded from many BBSs across the country. Many public domain, shareware and commercial communications programs also support the protocol. MS-DOS users can find Kermit as an option in Procomm, a shareware communications package by Datastorm Technologies, Inc. David Goodenough's Qterm program provides Kermit support for CP/M systems.

PERFORMANCE TESTS

Table 1 compares the performance of several public domain file transfer protocols. The tests were conducted from my residence using an IBM PC-XT running Procomm Plus communications software with a 1200 baud asynchronous dial-up connection to a local BBS. The communications parameters were set for eight character bits, one stop bit and no parity check. The Super Kermit programs employed sliding windows and data compression. ZMODEM capability was added to Procomm through an external program, DSZ.COM.

As the test results show, maximum throughput was achieved by the YMODEM protocol. ZMODEM and XMODEM 1K were also high performers, while Super Kermit was the slowest protocol tested. Another aspect to consider is the size of the file after the transfer. In this category, only ZMODEM and Super Kermit kept the original file size intact. All the other protocols increased the length in order to end on a 128-byte boundary.

The tests were performed under favorable operating conditions. If a high level of line noise or transmission delays had been a factor, the results would have been different. ZMODEM and Super Kermit, with full-duplex windowing capabilities, are designed to handle transmission delays. Kermit and XMODEM CRC, due to smaller packet sizes, can be more efficient than the other protocols on a noisy line.

CHOOSING A PROTOCOL

When signing on to a BBS, make note of the protocols it supports. Choose one based on its throughput, error-detection, error-recovery and effect on file integrity. Of the protocols discussed in this paper, ZMODEM has the most to offer. It has excellent throughput, employs 32-bit CRC and crash recovery, and does not alter file length. If ZMODEM is not available, look for YMODEM, then XMODEM 1K, XMODEM CRC and XMODEM checksum. ASCII is only recommended for transferring text files when other options are unavailable.

Use Kermit when you need to send 8-bit characters to a mainframe computer. Super Kermit and WXMODEM perform best when on line with commercial systems, such as the Source and People/Link. CompuServe offers its own protocol, known as CompuServe B. It is also available in the public domain, and should be used when on that system.

Public domain and shareware communications programs generally offer a better selection of non-proprietary protocols. Some, like Procomm and MEX, have evolved into more powerful commercial versions. Procomm Plus (MS-DOS), MEX-PC (MS-DOS) and MEX-PLUS (CP/M), support many of the file transfer protocols available in the public domain.

So, if you haven't given much thought to the file transfer protocol you've been using, try one of the high performance protocols the next time you download a file. You just might be surprised at the results.

BIOGRAPHY

Steve Westlund has been working with microcomputers since 1978. He is interested in telecommunications and applications programming in the CP/M and MS-DOS environments. Steve is employed by Washington University as a Senior Project Leader. He is responsible for the design, development and implementation of information systems. Steve also teaches programming classes at Belleville Area College and is finishing work on his Masters degree in Informat-ion Management. Steve can be reached on Bitnet at C08920SW@WUVMD.

REFERENCES

(1)Michael A. Banks, The Modem Reference (New York, 1988), p. 220.

(2)Al Stevens, "A Phone Directory and XMODEM added to SMALLCOM",Dr. Dobbs Journal, (April, 1989), p. 110.

(3)Michael S. Booner, Micro to Mainframe Data Interchange (Blue Ridge Summit, PA, 1987), p. 38.

(4)Alfred Glossbrenner, "The Downloading Zone", Personal Computing, (March 1988), p. 90.

(5)Michael A. Banks, The Modem Reference (New York, 1988), p. 235.

(6)Glossbrenner, p. 90.

(7)Frank da Cruz, KERMIT, A File Transfer Protocol (Pittsburgh, PA, 1987), p. 304.

(8)Chuck Forsberg, DSZ- a ZMODEM,

True YMODEM, XMODEM File Transfer Program (Portland, OR, 1988), p. 4.

(9)Forsberg, p. 4.

(10)Banks, p. 235.

(11)Forsberg, p. 34.

(12)da Cruz, p. 9.

(13)Glossbrenner, p. 92.

File: 51134 bytes (binary)			
Protocol	Min:Sec	Bytes/Sec	Bytes After Transfer
XMODEM	8:13	103	51200
XMODEM CRC	8:30	100	51200
XMODEM 1K	7:20	116	51200
YMODEM	7:16	117	51200
ZMODEM	7:18	116	51134
Super Kermit	10:12	83	51134

Table 1. File Transfer Protocol Performance Tests

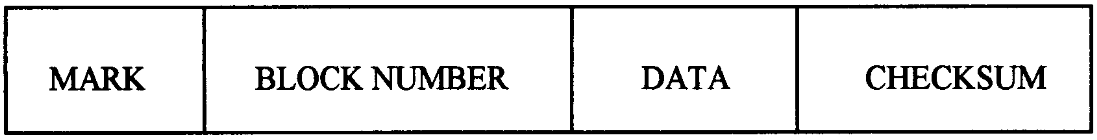


Figure 1. Basic XMODEM Block

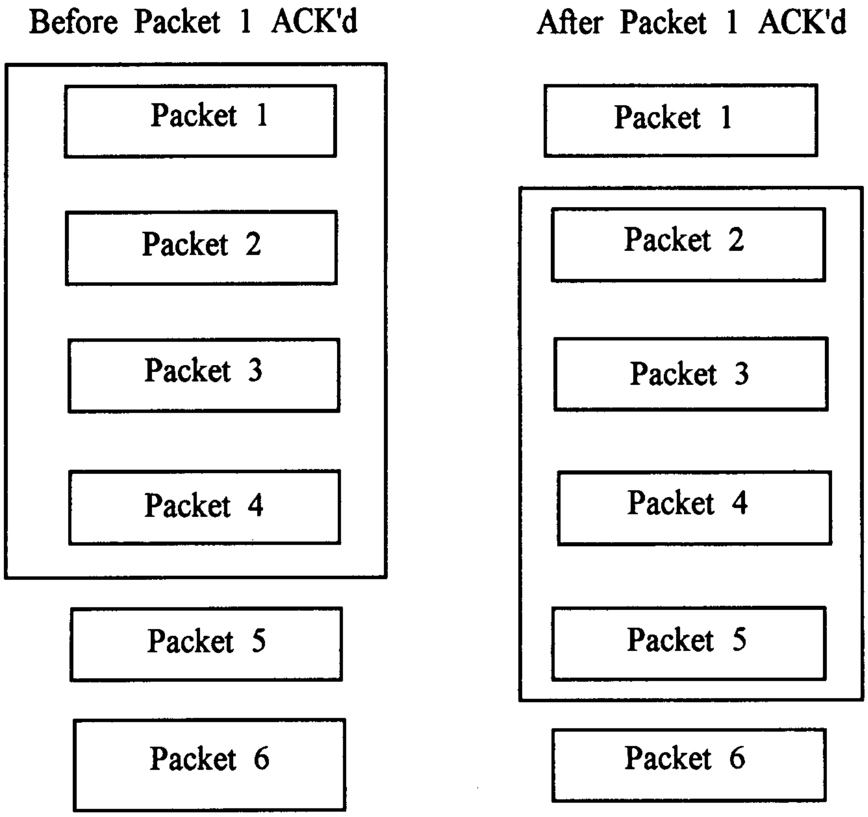


Figure 2. A WXMODEM Window

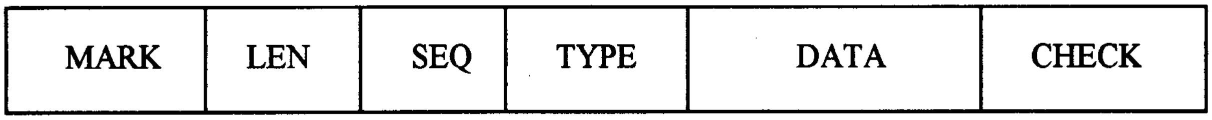


Figure 3. Kermit Packet Fields

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

OS/2 MINIX and SCSI

OS/2 2.0, Minix within OS/2, and SCSI

OS/2 2.0!

OS/2 2.0 is finally out: the full 32-bit, 386-specific operating system that does everything better and faster. Now you can run all of your existing PC software - at once!

You can have multiple DOS boxes - all the DOS boxes you want - and they can each have all the EMM and/or extended memory you want, all emulated through the magic of the 386. Plus, they all execute quasi-simultaneously with OS/2 sessions and with each other. Plus, the quality of the emulation is better than under 1.3. Heck, you can even boot a real DOS inside OS/2... or Minix (more on that later).

One of the nicest features is that you don't have to explicitly create a DOS box. All you have to do is enter the command name at the system prompt. If it's a DOS or Windows program, it will create a box for you.

The Windows aspect of the DOS box emulates Windows 3.0. Actually, it really **is** Windows 3.0. It runs in Standard mode only. If you simply type in a Windows program name, it'll run in a "Windows Full Screen" box. However, you can get it to run on the regular desktop, by creating a Workplace Shell object for it.

Workplace Shell is what we used to call Presentation Manager, and maybe still do. It's a little different from previous OS/2 and Windows GUIs; it uses a lot more direct manipulation (drag and drop).

But you could have read most or all of the aforesaid in PC Week. TCJ readers want to know about what's under the hood.

Well, you can write true 32-bit code now. Remember how many times the "640 K barrier" has been broken? It was broken by EMS, right? By EMM? By DOS 5? By Windows 3.0, certainly? Horse hockey!

There never was a 640 K barrier. Anybody who programs PCs can tell you right off that the real barrier has always been 64 K, just like on the Z-80. Yes sir, you can plug 32 megabytes of RAM right onto the motherboard, but as soon as you have a data object bigger than a measly sixty-four kilobytes, it's like changing your vacation from Baltimore to Bulgaria.

Anyhow, you can now write true 32-bit code and try to forget all of that segmentation garbage. But, you may have existing 16-bit libraries or DLLs. What then? No problem, you can still use them!

How can the operating system have mixed 32 and 16 bit code? Well, it's quite tricky. The selectors for 16-bit code are set up in a "tiled" fashion, so that there is a direct conversion between a segmented 16-bit address and a flat 32-bit address. So that the compiler knows to generate code for this, you have to use special compiler directives "_Seg16" and "_Far16". IBM's releasing a compiler toolkit, called Workset/2, and other compilers are on the way.

Of course all of your existing 16-bit OS/2 and DOS compilers will work just fine. They won't produce 32-bit code. However, since the operating system is internally more efficient, they'll often compile faster under OS/2 than they did under DOS!

What are the down sides? Some folks have had trouble installing OS/2 2.0 on some SCSI disk controllers, although I haven't. The default mode for installation is to run your hard disk ROM BIOS (chip on the controller) in a kind of DOS box. It takes a long time to install - it comes on 21 high-density floppies. The key to getting good performance is to have lots of RAM. Of course, RAM has continued to get cheaper and cheaper.

As an aside, many magazines, and even newspapers, made a lot of noise "comparing" Windows 3.1 to OS/2 2.0. That's a lot like comparing a twelve-speed bike to a Maserati. Yes, Windows 3.1 is somewhat better than Windows 3.0. But so what?

Microsoft is touting its future Windows NT as a new product which will preserve people's investment in Windows development by using an API which is "almost" compatible with Windows. This is a weird concept; Windows' API is awkward and messy, with funny and inconsistent naming, whereas OS/2's is cleaned up. Microsoft should know

- I'm pretty sure they're the ones who cleaned it up.

Minix within OS/2

Not only can you run DOS programs in a DOS box under OS/2, but you can also boot a *real* DOS disk from within .OS/2. The instructions for doing so are on page 101 of the Installation Guide. Basically, by entering the program command line "*" (a single asterisk), a special "virtual DOS machine" (virtual 8086 session) is created. These sessions emulate the entire PC hardware, including extended and expanded memory. Also, if your C: drive is FAT, you can access it from the booted DOS. Sorry, DOS can't access an HPFS partition.

Since the session is really an emulation of the PC hardware, you're not restricted to booting just DOS. You could, for example, boot Concurrent DOS or CP/M-86. And most importantly, you can boot Minix!

Minix will run in real mode, which means you're restricted to 640K. After creating the program object (call it "Minix"), putting a star in the program name, and setting DOS_STARTUP_DRIVE to A: (in DOS Settings), you're ready to go. Put the Universal Boot Disk (#3) in the A: drive and double-click. From there, you'll need disk #4 (Root File System) and a copy of disk #5 (/usr).

With DOS, you don't need to change the disk while booting, so you can make images of the system disk(s) on your hard drive with a program VMDISK. Since Minix requires that you change the floppy, you'll have to use the floppy drive, at least until you set up the hard disk partition(s) (/hd2 and maybe /hd3) for Minix.

If you're running with floppies, you'll need a non-write-protected copy of disk #5. Actually, I only work with copies of disks. I *never* use original disks with any program other than DISKCOPY. Copying these Minix floppies is a real pain. Since they have no BPB, DOS and OS/2 only copy 40 tracks, which is only 360K instead of 720K. I have two programs, GETDISK and PUTDISK, which I use to copy Minix floppies. Of course if you can get Minix up on a hard drive, you can use Minix to copy them.

Running operating systems inside other operating systems takes a little getting used to, especially when it's a multitasking OS like Minix. It's a feature which really has potential though. I always hated rebooting to change operating systems - now there's no need to!

SCSI

A great tool for experimenting with SCSI devices is the PC-532 monitor's "raw" command. You set up buffers for the command and any data you want to send or receive, issue the command, and then look at the buffers. Any sense data or messages that have come in are right there.

Quick introduction to SCSI: You send a command, which may be 6, 10 or 12 bytes long, depending on what command it is. You may send some data along with it (data out). Normally, everything goes fine and you may get data coming back, if you asked for any (data in). If there's a problem (a "check condition"), the controller will usually do a "sense" for you and retrieve some data describing the condition (sense data).

The SCSI bus can have 8 physical devices attached to it ("PUNs"). Each PUN can have 8 logical units (LUNs). Hardly anyone uses LUNs. The PUNs are usually numbered 0 to 7. PUN 7 is usually your interface card ("host adapter").

The commands are quite intelligent. You can ask a device to tell you what it is, how big it is, and other relevant details, and control all aspects of the device's operation. There are no more sectors, cylinders and heads. Instead, disk drives are just a sequence of logical blocks numbered starting at zero. There are SCSI disk drives, tape drives, scanners, graphics boards, video digitizers, printers...

Enough introduction. As usual, the software and documentation out there are almost criminally inadequate given the exciting potential of the hardware and protocol design. If you want to do anything, you pretty much have to start off at square zero. So what if you're not lucky enough to have a PC-532? What do you do if you have to use (shudder) a PC?

After sifting through the various claims out there, I've come to the conclusion that, in the PC world, there is only one SCSI API that is usable and documented, and that's Adaptec's ASPI.

ASPI is a pretty simple interface. You set up a structure containing various parameters such as the PUN to talk to, where your data buffer is, and which direction the data is supposed to go, plus the SCSI command, then make a far call, and you get a status back and sense data if it failed. ASPI works with either Adaptec or Corel SCSI boards, and maybe others too.

Some people may tell you that you don't need to understand the "complexities" of SCSI and interpose their own translation layer. But all they've done is substitute a different, proprietary set of complexities for the standardized complexities of SCSI, which actually aren't very complex anyhow.

Besides, sooner or later you're going to have a device that you need to toggle the bits and bytes on, and their "high-level" interface isn't going to support it, so you're going to have to get down into the SCSI level anyway.

SCSI is everything that IEEE-488 was supposed to be and lots more. Any new peripheral device should be designed with a SCSI interface. SCSI, not Ethernet, is the next RS-232. (I know, there are length limitations. These things take time.)

Now to access ASPI from Windows in protected mode, you have to use what is called DPMI. DPMI ("DOS

Protected-Mode Interface") is a messy and confusing kludge cobbled up to translate between "paragraph-style" (8086) segmented addresses and "selector-style" (80286) ones. It involves setting up a structure, setting some registers, and doing an INT 31 hex.

On the topic of DPMI...

Microsoft has a new C/C++ 7.0 compiler out. This product has two serious flaws. First, it doesn't support the 16-bit mode of OS/2 any more. This is no great loss, since OS/2 2.0 is out. The more serious drawback is that it uses some "Microsoft-proprietary extensions to DPMI" which require a special memory manager. I can't decide whether this is just bone-headed design - after all, we're only talking about a *compiler* here, not an operating system - or a deliberate, sleazy effort to prevent the compiler from working under any non-Microsoft OS. Either way, *avoid this product.* If you have 6.0, don't upgrade. If you don't, get Borland or Zortech.

The reason these PC compilers get so complicated is because of the Intel processors' segmentation. Of course the fact that C is such a muddy mess complicates the matter, but: Microsoft C/C++ 7.0 comes on 11 high-density floppies. The Prendeville compiler, for the clean NS32 architecture, fits on a single 360K floppy - and it includes all source, too.

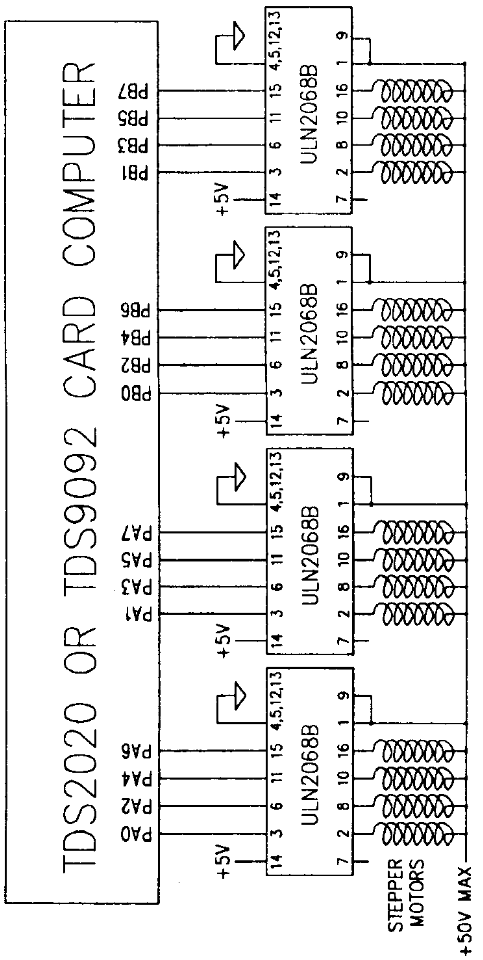
Next time

Well, I've used up my allotted space, so some of the news from the PC-532 and Linux fronts will have to wait for next time. Another topic I hope to discuss is the advent of the Gnu tools - gcc, gplus, and more - under OS/2. Running in full 32-bit mode, a regular PC is a whole new machine.

Where to call or write

Adaptec BBS: +1-408-945-7727

```
/* getdisk - copy a disk into a disk file
        901103 rr from readabs.c

        For Datalight Optimum-C
*/

#include "dos.h"
#include "stdio.h"
#define NUM_SECTORS 9
#define SECTOR_SIZE 512
#ifndef I8086L
** error must be compiled large model
#endif

struct dbt {
        unsigned char           specify1;
        unsigned char           specify2;
        unsigned char           motorturnoff;
        unsigned char           sizecode;   /* sector size code */
        unsigned char           lastsector;
        unsigned char           gaplength;
        unsigned char           datalength; /* usually 255 */
        unsigned char           formatgaplength;
        unsigned char           formatdatavalue;
        unsigned char           headsettle;
        unsigned char           motorstartup;
} *diskbasetable;

unsigned char buffer[ NUM_SECTORS * SECTOR_SIZE ];
static  void  usage( char * );
static  void  reset_disk_system( void );

/* -- main program -- */

main( argc, argv ) int argc; char *argv[]; {
        int             n, drive, head, track, sizecode, retry;
        int             i;
        union REGS              regs;
        struct SREGS            sregs;
        unsigned char           origsizecode;
        FILE            *outfile;

        if( argc != 3 ) usage( "Incorrect number of parameters supplied" );
        switch( *argv[ 1 ] ) {
        case 'a':
        case 'A':
                drive = 0;
                break;
        case 'b':
        case 'B':
                drive = 1;
                break;
        default:
                usage( "<floppydrive> must be A: or B:" ); p73
        }
        outfile = fopen( argv[ 2 ], "wb" );
        if( outfile == ( FILE * ) 0 ) usage( "Can't open output file" );
        /* get the disk base table data pointer */
        diskbasetable = * ( struct dbt ** ) 120L;  /* vector 30 */
        origsizecode = diskbasetable -> sizecode;
        printf( "\r\norigsizecode was %d", origsizecode );
        head = track = 0;
        sizecode = 2;

        memset( buffer, 0, NUM_SECTORS * SECTOR_SIZE );
        reset_disk_system();
        printf( "\n\nGetting disk...\n\n" );
        for( track = 0; track < 80; ++track ) {
                for( head = 0; head < 2; ++head ) {
                        printf( "\rtrack %d head %d", track, head );
#if 0
                diskbasetable -> sizecode = sizecode;
                if( sizecode == 0 )
                        diskbasetable -> datalength = 127;
                selse   diskbasetable -> datalength = 255;
#endif
                for( retry = 0; retry < 3; ++retry ) {
                        sregs.es = sregs.ds = getDS();
                        regs.h.dl = drive;          /* 0 = A, ... */
                        regs.h.dh = head;           /* 0 or 1 ... */
                        regs.h.ch = track;
                        regs.h.cl = 1;              /* sector 1 */
                        regs.h.ah = 2;              /* read sector(s) */
                        regs.h.al = NUM_SECTORS; /* # sectors */
                        regs.x.bx = ( unsigned int ) &buffer[ 0 ];      /* buffer */

                        int86x( 0x13, &regs, &regs, &sregs );     /* invoke BIOS */
                        n = regs.h.ah;              /* get errors */
                        if( n == 0 ) break;
                        reset_disk_system();
                }
                if( n ) printf( "\n** Disk error %d ** %d secs read\n",
                        regs.h.ah, regs.h.al );

                /* write the data gotten to the output file */
                if( fwrite( &buffer[ 0 ], SECTOR_SIZE, NUM_SECTORS,
                                outfile ) != NUM_SECTORS ) {
                                printf( "\nError writing output file\n" );
                                fclose( outfile );
                                exit( 1 );
                                }
                        }
                }
}
static void usage( s ) char *s; {
                printf( "Usage: getdisk <floppydrive> <filename>\n" );
                printf( "Error: %s\n", s ); p73
                exit( 1 );
}
static void reset_disk_system() {
                union REGS  regs;
                regs.h.ah = 0;                  /* reset disk system */
                int86( 0x13, &regs, &regs ); /* invoke BIOS */
}
/* end of getdisk.c */



/* putdisk - copy a disk file onto a disk
                901103 rr from getdisk.c

                For Datalight Optimum-C
*/

#include "dos.h"
#include "stdio.h"
#define NUM_SECTORS 9
#define SECTOR_SIZE 512
#ifndef I8086L
** error must be compiled large model
#endif

struct dbt {
                unsigned char           specify1;
                unsigned char           specify2;
                unsigned char           motorturnoff;
                unsigned char           sizecode;  /* sector size code */
                unsigned char           lastsector;
                unsigned char           gaplength;
                unsigned char           datalength; /* usually 255 */
                unsigned char           formatgaplength;
                unsigned char           formatdatavalue;
                unsigned char           headsettle;
                unsigned char           motorstartup;
} *diskbasetable;

unsigned char buffer[ NUM_SECTORS * SECTOR_SIZE ];
static  void  usage( char * );
static  void  reset_disk_system( void );

/* -- main program -- */

main( argc, argv ) int argc; char *argv[]; {
                int             n, drive, head, track, sizecode, retry;
                int             i;
                union REGS              regs;
                struct SREGS            sregs;
                unsigned char           origsizecode;
                FILE            *infile;

                if( argc != 3 ) usage( "Incorrect number of
                parameters supplied" );
                switch( *argv[ 1 ] ) {
                case 'a':
                case 'A':
                                drive = 0;
                                break;
                case 'b':
                case 'B':
                                drive = 1;
                                break;
                default:
                                usage( "<floppydrive> must be A: or B:" ); p73
                }
                infile = fopen( argv[ 2 ], "rb" );
                if( infile == ( FILE * ) 0 ) usage( "Can't open input file" );
                /* get the disk base table data pointer */
                diskbasetable = * ( struct dbt ** ) 120L;  /* vector 30 */
                origsizecode = diskbasetable -> sizecode;
                printf( "\r\norigsizecode was %d", origsizecode );
                head = track = 0;
                sizecode = 2;
                memset( buffer, 0, NUM_SECTORS * SECTOR_SIZE );
                reset_disk_system();
                printf( "\n\nPutting disk...\n\n" );
                for( track = 0; track < 80; ++track ) {
                                for( head = 0; head < 2; ++head ) {
                                                printf( "\rtrack %d head %d", track, head );
                                /* read the data gotten to the output file */
                                if( fread( &buffer[ 0 ], SECTOR_SIZE, NUM_SECTORS
                                infile ) != NUM_SECTORS ) {
                                printf( "\nError reading input file\n" );
                                fclose( infile );
                                exit( 1 );
                        }
#if 0
                        diskbasetable -> sizecode = sizecode;
                        if( sizecode == 0 )
                                diskbasetable -> datalength = 127;
                        else    diskbasetable -> datalength = 255;
#endif
                        for( retry = 0; retry < 3; ++retry ) {
                                sregs.es = sregs.ds = getDS();
                                regs.h.dl = drive;      /* 0 = A, ... */
                                regs.h.dh = head;       /* 0 or 1 ... */
                                regs.h.ch = track;
                                regs.h.cl = 1;          /* sector 1 */
                                regs.h.ah = 3;          /* write sector(s) */
                                regs.h.al = NUM_SECTORS; /* # sectors */
                                regs.x.bx = ( unsigned int ) &buffer[ 0 ];  /* buffer */

                                int86x( 0x13, &regs, &regs, &sregs );
                                /* invoke BIOS */
                                n = regs.h.ah;          /* get errors */
                                if( n == 0 ) break;
                                reset_disk_system();
                        }
                        if( n ) printf( "\n** Disk error %d ** %d secs writ\n",
                                        regs.h.ah,  regs.h.al );
                        }
                }
}
static void usage( s ) char *s; {
                printf( "Usage: putdisk <floppydrive> <filename>\n" );
                printf( "Error: %s\n", s ); p73
                exit( 1 );
}

static void reset_disk_system() {
                union REGS              regs;
                regs.h.ah = 0;          /* reset disk system */
                int86( 0x13, &regs, &regs ); /* invoke BIOS */
}

/* end of putdisk.c */
```



This is one suggested interface to four stepper motors, or just use 16 ZVN2106A MOSFETs. Use the library routines S4STEP.TDS and 4STEPPER.TDS to drive the motors individually in the background while the main program gets on with user interface via LCD and keyboard.

Reprinted from the newsletter of: Triangle Digital Services Limited, 223 Lea Bridge Road, London E10 7NE, Tel: 081-539 0285. In USA & Canada: The Saelig Company, 1193 Moseley Road, Victor NY 14564, Tel: (716)425-3753. On CompuServe: USA Sales 71042,17, other sales /technical 100065,75.

MDISK at 8 MHZ

By Terry Hazen

CLASSIC Suppoirt

Intermediate

Ampro Hardware

Terry sent me a letter with his article which I think introduces his topic perfectly. BDK

The MDISK at 8Mhz, is an article that tells how to make an Ampro Z80 Little Board with an MDISK 1 megabyte RAM disk combination run at 8 Mhz. The MDISK is an add-on board for the Ampro that provides space for 1 megabyte of RAM that may be configured for use as a RAM disk. MDISK also provides the Ampro with bank-switching capabilities for operating system expansion, such as use of Cam Cottrill and Hal Bower's new banked ZCPR40-ZSDOS2-B/P BIOS system.

MDISK at 8 is a follow-up to several earlier articles. The original MDISK hardware and software articles ran in TCJ29/30 and George Warner did an article in TCJ54 on converting the Ampro from 4 Mhz to 8 Mhz.

Finally, I'd like to say how much I like TCJ. I look forward to each issue, especially the ZCPR3, YASBEC and other Z80 hardware and software features. I only wish Lee Hart could return with more advanced Z80 software ideas. I learned a whole lot from his contributions.

I wish you success with TCJ. It's certainly a big job!

Regards, Terry

Adapting A 4 Mhz Ampro Little Board And MDISK 1 Megabyte RAM Disk For Use At 8 Mhz

George Warner ended his TCJ #54 article on modifying 4 Mhz Ampro Z80 Little Boards to run at 8 Mhz by saying that he hadn't been able to make an Ampro/MDISK 1 megabyte RAM disk combination (TCJ #29,30) run reliably at 8Mhz. I'm happy to say that it only takes a few simple chip changes to make an 8 Mhz-modified Ampro/MDISK run reliably at 8 Mhz and to provide you with a very speedy and roomy RAM disk workspace.

First, however, a correction to the basic Little Board Model 1B (the board with the built-in SCSI controller) 8 Mhz conversion directions. In the second column of page 31, direction number 3 for providing 8 Mhz to the Z80 actually applies only to Model 1A. The correct Model 1B procedure is to cut the component-side trace taking the 4 Mhz clock signal from U3 pin 13 to Z80 pin 6 at a point directly adjacent to pin 6 of the Z80. You then provide the 8 Mhz system clock to the Z80 by jumpering U3 pin 14 to Z80 pin 6 on the circuit side of the board.

Adding the MDISK RAM disk board to the Ampro Little Board changes the overall RAM timing a bit. Making the board combination boot and run at 8 Mhz requires a few chip changes on both boards to speed up some operations while slowing down others so that the overall RAM timing is brought back in sync at 8 Mhz. I've specified type 74F chips when faster chips are required. It's possible that other fast chip types may be used in place of the 74F's, but I didn't have any on hand to try. Feel free to experiment. If the boards will boot and run at 8 Mhz, the timing is ok. It's as simple as that. If the timing isn't right, the drive light on the boot drive won't light or you won't make it all the way though the boot process.

On the Ampro Little Board, 6 chips need to be snipped out and the leads carefully desoldered and removed. Replace them with sockets and add the following new chips:

Model 1A	Model 1B	New Chip
U19	U35	74LS00
U28	U38	74LS02
U18, U27	U26, U37	74LS74
U20, U29	U22, U23	74F157

On the MDISK board, replace the two socketed 74LS245 buffer chips (U51, U53) with 74F245's. The 150ns DRAM chips used at 4 Mhz will still work fine at 8 Mhz. If your board has 200ns DRAM chips, you will probably have to replace them with faster chips. You should now be able to boot and run at 8 Mhz.

What will this 8 Mhz conversion do for you? Even if you use your Ampro/MDISK mostly for word processing or other terminal-oriented work, you'll see a big difference between a 4 Mhz Ampro/MDISK and the 8 Mhz version. You'll see even greater differences when running more computation-oriented applications. Here are a few real world task speed comparisons between 4 Mhz and 8 Mhz Ampro/MDISKs, with the application programs running on the RAM disk. See the chart on the next page.

Now for the gotcha's! Ampro-specific utilities that talk directly to the floppy disk controller chip, such as AMPRODSK and MULTIDSK, contain fixed delays that assume a 4 Mhz clock. These utilities may not operate properly at 8 Mhz, particularly on a Model 1B. Because of this problem, I've rewritten a set of Ampro utilities to automatically check the clock speed when they are run and set their disk delays accordingly. They also operate properly under NZCOM. See AMP8ZU13.LBR on your favorite Znode.

Ampro's last official BIOS release, BIOS38, seems to work all right on my own 8 Mhz floppy disk machines, but it has fixed floppy disk controller delays which may cause floppy disk sector write problems at 8 Mhz on some Model 1B systems. Model 1A floppy disk operations don't seem to be as sensitive to clock speed for some reason.

With improper delays, it's possible for the floppy disk controller to lose data. When it does, it simply writes zeros for several records. Since this would be very unfortunate if you're doing a directory write, you should initially verify floppy disk driver operation by writing to an unimportant disk before attempting to write to an important one! Since the datestamping process also involves disk writes, turn off your datestamping before doing any initial testing just to be on the safe side.

Cam Cottrill and Hal Bower's new banked ZCPR40-ZSDOS2-B/P BIOS system also runs very well on both my 4 Mhz and 8 Mhz Ampro/MDISK floppy/RAM disk systems and I hope to have the B/P BIOS Ampro hard disk driver sorted out shortly.

Converting your old 4 Mhz Ampro system by speeding it up to 8 Mhz and adding 1 megabyte of RAM won't make it run neck-and-neck with an 18 Mhz YASBEC. I don't think even George Warner could do that - that wasn't a challenge, George, really! But I think you'll be impressed with how much smoother and more responsive your Ampro/MDISK has become. It's amazing how far you can take that old 4 Mhz Ampro Little Board floppy disk system if you don't mind handshaking with the cool end of the old soldering iron!

Terry Hazen has a background in analog electronic and mechanical engineering. He is currently a product design consultant, specializing in medical electronic systems. He encountered his first Z80 computer in 1982, installing ZCPR30 when it first came out and has been pursuing Z80 hardware and software projects ever since. His company, n/SYSTEMS, produces the MDISK add-on RAM disk for Ampro LB computers. MDISK also provides the Ampro with bank-switching capabilities for operating system expansion. Terry enjoys designing and building varied types of hardware and software projects, not all of them computer-related. His recent software projects include the HP and HPC RPN calculators, the ZP file/disk/memory record patcher, the REMIND appointment reminder utility as well as several new and upgraded Ampro-specific utilities: AMP8ZU13, a set of seven ZCPR3/NZCOM/8 Mhz versions of Ampro configuration/system utilities and LBCLKS12, clock utilities for reading and setting the BIOS39 clock, ZSDOS clock and SmartWatch. He may be reached on Ladera Z-node #2. His address is 21460 Bear Creek Road, Los Gatos, CA 95030.

RAM Disk Task	4 Mhz	8 Mhz
Floppy-Floppy copy (18 files):	51.30 sec	48.88 sec
RAM-RAM copy (18 files):	21.05 sec	8.81 sec
WordStar (^QC,^QR - 78k file):	26.66 sec	10.20 sec
dBASEII (plot data on terminal):	24.18 sec	12.40 sec
FILT84 (174k file):	59.06 sec	28.17 sec
ZMAC (174k source file to REL):	122.92 sec	61.05 sec
Z80ASM (174k source file to REL):	29.45 sec	14.16 sec
Z80ASM (174k source file to COM):	26.56 sec	12.74 sec

Sending Articles to *TCJ*

Send your articles and letters of inquiry to:

The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

The editorial policy is to seek articles that can enhance and educate our readers. Letters of interest will be printed in our Reader to Reader section on a space and topic consideration. Material is typically printed "as is", however *TCJ* does reserve the right to reject or modify (by omitting) portions of letters or articles deemed unfit for publication.

Major letters and minor articles are accepted on floppy disk and will aid in getting your letter published "as is." *TCJ* does not return disks and materials, unless suitable and appropriate return mailers and postage is provided.

Floppy disk and wordprocessing formats supported by TCJ, are all the current 51/4 and 8 inch formats in general use. CP/M formats are supported by both Xerox conversion program and PCDOS to CP/M conversion program. PLEASE do not use embedded punctuation in the file names, as PCDOS will not accept them. Output is to PageMaker 4.0, which accepts many standard PC based word processing file formats.

For quick response to inquiries and submissions, GENIE services are recommened. Leave mail and attached files for B.KIBLER. The GENIE system is typically checked two to three time per week, except during production of the next issue.

Many changes and projects are in the works. I ask that you be patient while we continue to improve *TCJ*.
Bill Kibler

Introduction to Forth

By Frank Sergeant

Feature Article

Beginners

Getting Started

Introductory Pygmy Forth Tutorial for the Complete Beginner (or is it An Incomplete Tutorial for An Incomplete Beginner?)

"I want to learn Forth but just can't seem to put the pieces together to do anything." Bill Kibler tells me he has heard this complaint from a number of people, and he wants me to do something about it.

It just so happens that I am in the last stages of preparing Pygmy Forth version 1.4 for release. It will probably be available by the time you read this. As part of the project I have been writing an experimental Forth tutorial for beginners. It is not meant to be the last word, but if you suffer from the above complaint, you might look it over and let me know if it helps any. Many of the following concepts apply broadly to whatever Forth you might have handy (and even other languages) but the exercises are narrowly aimed at Pygmy, so feel free to get a copy (available from FIG and fine BBSs and shareware houses everywhere), and even to borrow a PC if necessary, in order to do the exercises.

Foot in the Door

Here's the tricky part: what do *you* need to know to learn Forth? If we cover it at too low a level you can say "Yeah, yeah, I already know all that, but I'm still stuck." If we cover it at too high a level then perhaps nothing will make any sense at. There is a *lot* of information available about Forth in books, magazines, diskettes, and bulletin board messages. Much of this material is available through the Forth Interest Group (FIG) at 510-89-FORTH. You should join, or at least get their list of publications, and read as much as possible. At the very least you should study the book *Starting Forth* by Leo Brodie and you should read the Pygmy Forth manual (the file PYGMY.TXT) and study its source code blocks and their shadows.

RSVP

If you are a beginner, or if you are "stuck" somewhere in your study of Forth, please study this tutorial and actually do the practice exercises, no matter how silly they sound. Then send me a report. I would like to know exactly what you are having trouble with, what makes sense and what doesn't. Tell me what you have tried and where it went wrong and where it went right. What do you think are the key points, which if explained better would make it all make sense? What are your major remaining questions or difficulties? I will attempt to respond in an upcoming article.

Disclaimer1:

This tutorial takes a single approach to introducing you to Forth. Right here, at the beginning, I wish to acknowledge that many other approaches could be taken. Having said that, I will omit the on-the-other-hands. Let's pretend for now there is only one right way to do it, and this is it.

Disclaimer2:

This is a tutorial on Forth, specifically Pygmy Forth for MS/PC-DOS computers. It does not include information about how to use DOS or how to turn on your computer or warn you that you need to type a carriage return at the end of a command or explain what an .EXE or .COM or .BAT file is or how you can find the executable files on a disk or how to browse through ASCII text files or what ASCII means. I apologize in advance if I inadvertently touch on any of those subjects.

The One Right Way

To master Forth you only need:

A. a few broad, simple concepts
B. a "cookbook" collection of examples
C. a lifetime's worth of evaluating what you are doing and why

A Few Broad, Simple Concepts

Forth is modular.

We do our work in little bitty pieces and pyramid them into simple, powerful, hierarchical structures. We should all adopt Rob Chapman's slogan "It's so simple it has to work." Our job is to *make* it that simple and *keep* it that simple.

Forth is interactive.

We build a little piece and test it immediately from the keyboard. The more we test early the less we are bitten later.

Forth is not a religion.

Unlike some languages I might mention, Forth is so simple we do not need to take it on faith. We can see it and test it, rather than having to "believe" and "hope" and "trust." We can inspect and modify any part of the system. Full source code is included, and is small enough, and modularly organized enough, to be manageable.

I want to go over this point again because it is so radical it might be missed: Pygmy Forth includes *all* of its own source *code*. You can actually understand it. You can study and modify the system. You can examine any part of the system you are curious about. This is virtually unheard of with any language except Forth.

Forth has an explicit data stack.

Take a stack of magazines and put them on the floor one at a time, one on top of the other. Which is the easiest to get to (the one on top)? Empty the stack and put one magazine down, saying "3" then put another magazine on top of it, saying "5." Ok, what's on the stack (3 and 5)? Which number is on top (5)? Now pretend you are the + (i.e. "plus" or addition) operator. It is your job to take two numbers off the stack, add them, and put the sum back onto the stack. Go ahead do it, take the 5 and 3 off the stack, add them to get 8, and put the 8 back on the stack. How many items are on the stack now (one) and what is it (8)? (Notice the "subliminal" hints in the previous questions, in case you are having trouble answering.)

Forth has active operators.

In the previous example, + *did* something; it wasn't a parsing symbol to tell something else to do something. Forth words are active! They go to the stack to get the materials (numbers) they need, they do their work, then they place their results on the stack.

Forth must be practiced.

Ok, you know enough now to start practicing with the computer. Go ahead, bring up Pygmy.

Put some numbers on the stack by typing **1 3 5 7 9** followed by the carriage return (which I am *never* going to mention again!). What is on the stack? What is on top? Type **.S** to display the contents of the stack and check to see if your answers were correct. Now type **.** and see what happens. Type **.S** and see if the stack is different. The dot removes the top item from the stack and prints it as a number. Play with putting numbers on the stack and removing them.
Forth uses postfix notation.

This should not surprise you since it is how + worked several paragraphs back. Postfix means you type the operator *after* typing its operands. Try out these examples (and use **.** or **.S** to see the results):

multiply 3 by 5 by typing 3 5 *

subtract 7 from 9 by typing 9 7 -

Note, the order of operands is exactly the same as you are used to in "infix", i.e. 3*5 or 9-7; the only difference is the operator goes after its operands instead of between them. Ok, continue.

(3+5)*(6+2) by typing 3 5 + 6 2 + *
3+5 * 6+2 by typing 3 5 6 * + 2 +

Postfix is simple, direct, doesn't require precedence rules, doesn't require parentheses. You'll get the hang of it in no time. Notice even in these complex examples that the order of the operands (the numbers) is the same in both the infix and the postfix versions. The last example above, due to the higher precedence of the infix multiply operator, is *not* the equivalent of 8*8, but of 3+30+2. Forth uses one or more blanks to separate words in the input stream.

A word is a group of non-blank characters. Generally speaking, the only things you feed Forth are words. Each word is one of three things:

1. A word already known to Forth (i.e. it is in Forth's dictionary), in which case Forth executes the word.
2. A word not in Forth's dictionary, but one that can be interpreted as a valid number (e.g. 75), in which case Forth does interpret it a number and pushes it to the stack.
3. Neither of the above, in which case Forth reports an error.

Please re-read that. Isn't that simple? Isn't that pretty? Like the shark is a feeding machine, Forth is an executing engine. It gobbles input, pushing numbers to the stack, executing words which exist in the dictionary, and choking on anything else.

It makes no difference whether you separate words with one space or 50, whether you put several words on one line or spread them across several lines.

You can add words to the dictionary. And, they are full citizens with equal standing to all the other words in the dictionary. To use this power responsibly, remember to add little bitty words and not great big unmanageable words. I do not mean its name should be short, but that its function should be simple, obvious, straightforward. We "divide and conquer" at the beginning rather than trying to debug a hopeless mess at the end. Do not make a word a jack of all trades. Make it a master of one.

This process of adding a word to the dictionary is called defining a word. Several different types of words can be defined, but for now we will study a single type. This one type can be used for everything you need at first. This type of definition is called a colon definition, because the colon starts the definition, e.g.

```
: SEVENTEEN    17 ;
```

Type that in and see what happens. Nothing? Well, not nothing exactly, as you did get the "ok" prompt to indicate Forth gobbled it up with no complaints. The new word **SEVENTEEN** is now in the dictionary. Type **SEVENTEEN** and see what happens (use **.** or **.S**). So, what else would you expect it to do? It puts the number 17 on the stack. The definition begins with the word **:** which is followed by a blank because the colon is not a *symbol* but an active word.

You are used to numbers preceding a word that uses them (so they will be waiting on the stack). On the other hand, strings often follow the word the uses them. In this case **:** expects to find a string in the input stream, and it uses this string as the name of the new word it creates. Then colon collects all the other words up to the **;** and stores them in the body of the new word. The semicolon is a special word in that it ends the definition. Try defining and executing the following until you get the hang of it:

```
: 3*  ( n - 3*n)  3  *      ;
17   3* .
2     3*  .  2  3*  3*  3*  .

: FEET  ( feet - inches)  12  *  ;
6 FEET  .
5 FEET  .

: STAR  ( -)  ." *"  ;
STAR STAR  STAR  STAR
: STARS  ( # - )   FOR  STAR
```

```
 NEXT ;
0 STARS
1 STARS
2 STARS
200 STARS

: DIGITS ( -)    0 10 FOR DUP .
1+ NEXT  DROP  ;
DIGITS
: DIGITS ( -)    CR CR CR DIGITS
CR CR CR  ;
```

Wow, have we got some explaining to do. Notice the comments in parentheses inside every definition such as **(n -)**. Notice how the left parenthesis is followed by a space. As you must be getting used to by now, the left parenthesis is a word, but one that is executed during the defining process. What does it do? It skips the input stream up through the ending parenthesis.

Why do we put in comments? Several reasons. First, it is our minimum specification of what the word is intended to do. It shows what should be on the stack before and after the word executes. We need to know what the word should do as we write it and as we test it. For example, **DIGITS** takes nothing from the stack and places nothing on the stack. We call this comment a "stack comment" or a "stack picture". Get in the habit of putting this in every word you define. Notice the use of Forth's print statement in **STAR**. The **."** (i.e. dot-quote) is a word, so it is followed by a space, then by whatever characters you want to display, then by an ending quote mark. **FOR ... NEXT** should be obvious. **DUP** copies the top of the stack, i.e. its stack comment would be (n - n n). **DROP** drops the top item from the stack, i.e. (n -). **CR** does a carriage return.

Next, you try defining a word that prints your name. Then use that word in the definition of another word. Here's an example:

```
        : ME ( -) ." FRANK"  ;
```

(be sure to test it before you continue!) then

```
 : WHO? ( -)   ME ." , THAT'S
WHO!"  CR  ;
```

Forth is case-sensitive.
At least Pygmy and many others are case-sensitive. To test it try typing the following four lines to see which of them Forth executes and which it chokes on.

```
        CR
        cr
        cR
        Cr
```

Review
Where are words looked up (the dictionary)? Where do numbers go (the stack)? How long should a definition be (not very)? When should you test (immediately upon defining each word)? How do you find out how a particular Forth word works (look at its source and experiment with it from the keyboard)? There you have it. Those are the concepts you need. Everything else is simply a matter of asking "How do I do such and such?" and finding a cookbook example to copy, or better, to study and possibly modify.

The Cookbook
Where do you find examples, and how do you answer questions about the Forth system? Type **WORDS** and see what happens. This displays the words in the dictionary. Did they scroll by too fast? Try it again and press any key to make it halt, and then press a key again to let it continue. When you see a word that interests you, such as **EXPECT** type

```
        VIEW EXPECT
```

Bang, you are popped into the editor at the definition of EXPECT. You can read its definition, especially its stack comment. You can then use PgUp and PgDn to browse through nearby blocks of source code. If you have the shadow blocks for Pygmy you can press Ctrl-A to alternate between the block EXPECT is defined on, and its shadow block that gives more information.

Speaking of the editor, type **.FILES** and see what happens. This shows the files that are currently available and the beginning and ending block numbers for each file and the DOS handle number. (If the handle number is shown as -1 then the file is not available.) Notice that PYGMY.SCR begins at block zero, so type

```
        0 EDIT
```

This file holds _all_ the source code for Pygmy. Browse through it with PgUp, PgDn, (and switch to and from the shadow file with Ctrl-A, if the shadow file is available). Do not worry too much about learning all the details of what you see. Right now you are just taking a tour to see what's where. Go ahead, browse awhile. Notice that the definitions of words usually have stack comments. Press Esc to get out of the editor.

Between **WORDS EDIT** and **VIEW** you should be able to locate the source code for any word in the dictionary. Now here's where the Cookbook comes in: get into the editor at the first block or so (e.g. 1 EDIT) and then use the F3 (I.E. the function key F3) and the F10 keys to setup a search string for the word of interest and then search across blocks for it. This way you can find examples of how that word has been used within other definitions.

You can learn a lot from doing that. Plus you will get more comfortable with Pygmy as a whole. However, until you are comfortable, until you are familiar with just how it works, you can refer to the following list of How-do-I-do-such-&-such questions to get going.

Q: I am tired of defining words and having them scroll off the screen where I can no longer see the stack comment you made me write. Ditto for retyping the definition every time I restart Pygmy from DOS.

A: That's not a question; that's a complaint. But, nevertheless, you are now ready to use the editor to save and change the source code you write. An entire file is setup just for your own code. It is named YOURFILE.SCR. You can see it listed when you type **.FILES**. Note that it starts at block 2000. Type **2000 EDIT** and browse through it with PgDn and PgUp. All the blocks are blank, probably, unless you have already put stuff in them. See the documentation file PYGMY.TXT for instructions on using the editor, or use the quick reference reminder list on the status line at the top of the display and experiment.

Type away to enter your source code.

Then to compile your source code, say from block 2002, get out of the editor by pressing Esc and type **2002 LOAD**. To get back into the editor, you can type **2002 EDIT** or you can just type **ED** to return to the last block edited. When you exit from Pygmy with BYE the words you've loaded will vanish from the dictionary, but their definitions will still be on block 2002. Next time you run Pygmy, you can easily reload those definitions without retyping them just by saying 2002 LOAD.

Q: I'm tired even of typing 2002 LOAD to reload my favorite little additions every time I run Pygmy. Isn't there a way to avoid that step?

A: At least you asked a question this time. Yes, there is a way. After you have loaded your favorite words, type

```
        SAVE  A5.COM
```

That creates an executable file similar to the one you started up Pygmy from, but named A5.COM. Of course you are free to invent other names if you don't like the sound of A5. But, the file extension should always be .COM. Next time you want to run Pygmy, run it by typing A5 from DOS. The dictionary will contain all your goodies.

Q: How do I create a file in Pygmy?

A: Why do you want to know? You already have a file for your source code blocks, i.e. YOURFILE.SCR. Just use it!

Q: But it's full. It only has eight blocks and I've used them all up.

A: Oh, well, that's simple. Get into the editor and move to the block where you want more room. Press the F9 key. The editor will ask you how many. Enter a number, such as 20. The editor will spread open the file at that point and insert 20 blank blocks right after the current block.

Q: But, what if I want to use a *different* file for some of my code?

A: Ok, ok. There are several ways to do it. Just get out to DOS and type

```
        COPY   YOURFILE.SCR
NEWFILE.SCR
```

then get back into Pygmy and type

```
        " NEWFILE.SCR" 4 OPEN
        SAVE A6.COM
```

or, from within Pygmy you can create a new file by loading the definition of NEWFILE from block 136 and using it to create the new file, e.g.

```
 "NEWFILE.SCR"  NEWFILE
```

Well, that give you the idea, I'm sure.

Q: Can Pygmy load from textfiles? I'm more comfortable with them and with my favorite editor.

A: What? Well, just practice with Pygmy's block editor until you become comfortable with *it*. But, the answer is yes, you can load textfiles with FLOAD or with INCLUDE. I'll leave it as an exercise for you to look up those words in PYGMY.SCR and/or to look up the discussion of textfile loading in the manual PYGMY.TXT. However, block files have a lot to offer over textfiles, such as their instant availability with the built-in editor, and their inherent modularity. You *are* writing very small definitions, aren't you?

Q: But I've seen some of *your* source code and sometimes a single word damn near overflows an entire block.

A: Don't do what I do; do what I say.

Conclusion

Do you feel oriented and comfortable yet (yes, I sure do, you've opened my eyes to the power and beauty of Forth!)?

Author's Bio

In addition to developing the Bare Bones EPROM Programmer Kit and his newest, prettiest version of Pygmy Forth, Frank has recently learned to use the pumping lemma to prove that certain languages are not regular. In spite of that he is accepting Forth questions for an upcoming article via GEnie as F.SERGEANT or via the postal service at 809 W. San Antonio Street, San Marcos, Texas 78666.

On Language Independence

Jay Sage's article talks about generating programs for use in languages other than english. There are several ways in which Forth can achieve that goal.

In Frank's article, he gives an excellant example of how Forth can handle language independence. The example of seventeen, wherethe name is entered and the actual numeric value is put on the stack, is exactly how independence is achieved. The difference for use under other languages, is using the diffinition as the foreign word, and the action would be the english equivalent. This means the entire dictionary could be redifined in another language. Each language desired, would have a language load module that redifines all words to the new equivalent word.

Printed statements would still be in english, unless all statements had previously been defered words. Defered words are stored as a pointer to the the statement, which can be later redefined to another process or definition. Currently most Forths use inline text messages. For language independence, the Forth would need modification of all text strings to a defered structure.

Although some parts of this explanation were more complex than Franks article, language independnce can be tested very easily. To make a Spanish version, the number four would be defined (in F83):

```
 : QUATRO  4 ;
```

and it is that simple. For text strings we defer the word then change the pointer:

```
DEFER HELLO
: ENGHELLO  ." HELLO " ;
: DUTHELLO  ." HALLO " ;
```

Now to display english hello we do:

```
 ' ENGHELLO IS  HELLO
```

or for our Dutch version:

```
 '  DUTHELLO IS HELLO
```

and wherever any reference is made to print the banner hello, it is redefined in the Dutch version and "HALLO" prints as desired. This is another reason why I feel that Forth is an universal operating system.

Bill Kibler.

Shell Sort in Forth

by Walter J. Rottenkolber

Special Feature

Intermediate

F83 Forth

I saved this article for after Frank gave us his great beginners review of Forth. Now here is a complete review of SORTING with some Forth how-to as well. Should help all you beginners get a better handle on using Forth. BDK

SHELL SORTS

The Shell sort dates back to 1959, a time in the early days of computers when the main random access memory began the climb to the unheard of heights of 16 kilobytes, and programmers could contemplate sorting large arrays in memory.

Starting with the Basics

I started with Sedgewick's book. His algorithm for the Shell sort is based on the Shuttle sort, a.k.a the Insertion sort. Trying to understand the mechanics of this sort by examining his pseudocode code is like decoding cuneiform. Much more enlightening is working through a physical example of the shuttle sort.

Imagine, say, a row of 10 shuffled playing cards before you (better yet, use the real thing). Place the largest card at the left end, and the smallest at the right. The intent is to sort the cards small to large beginning from the left. The sweep of the sort is from left to right.

To start, pick up card #2 and compare it to card #1. If card #1 is larger than #2, shift #1 to the right one space. Since there are no more cards to the left of the card in hand, put it down in the slot opened by the shifted card. Now go to the next card in the line (#3), and pick it up. Repeat the same comparing and shifting until either the card to the left is smaller, or the edge of the card set is reached. Then set the card down. As you proceed with the sort, your hand will move first right, then left, until the rightmost card is processed, and the sort stops. This back and forth motion of the hand resembles the movement of the shuttle in a loom, hence the name of the sort.

However, I discovered the most common variety of Shell sort described in programming books is one based on a cousin of the Shuttle sort, the Sift sort. The general movement of the hand is the same, but instead of picking up the card, you first compare it to the one to the left. If the card on the left is larger, you exchange the two cards. Then you shift one card to the left, and repeat the comparing and exchanging until either the card on the left is smaller, or the end of the array is reached. At that point, you stop and process the next card in the set.

If the Sift sort reminds you of a backwards running Bubble sort, you're right. The big difference is that when the comparison fails, I.E. the left card is smaller, not only does the exchange not take place, but the 'bubbling' stops also, and the routine goes to the next card. A Bubble sort would continue to the end of the array.

(To add confusion, the term Sift sort is also applied to the Shuttle sort and to a type of binary tree sort, but I intend for it to refer to the exchange variant of the insertion sort.)

Notice that while small cards quickly find their proper places in the sorted array, they do so only after a tedious comparison with cards on their left. This is especially true of small cards on the 'wrong' end of the array. And large cards move along only incrementally with the main sweep of the sort. If only some simple way could be found to take these outlying values and bring them closer to home before the final sort took place.

Enter Shell

In 1959, Donald Lewis Shell proposed such a means in his paper ''A Highspeed Sorting Procedure''. His idea was to introduce a Gap between the array elements to be compared. After sorting, the Gap is reduced, and the sort repeated. This continues until the Gap declines to one, and the sort becomes a simple insertion sort. These gaps logically divide the array into groups that are increasingly meshed as the sort proceeds. He chose the Sift sort (Figure 1) to illustrate his idea. The number of elements was divided by two for the initial gap, with repeated division for subsequent gaps.

Knuth called the Shell sort an 'Insertion sort by diminishing increment'. Determining the optimum gap sequence seems to have defied analysis. His studies did show that gaps of odd, preferably prime, number would sort better than the even numbered groups Shell originally used. He suggests calculating the gap sequence using a gapfactor of three. Segewick and Wirth agreed with Knuth. The sequence should avoid gaps that are multiples of one other. They also preferred the Shuttle sort as the basis of the Shell sort (Figure 2).

If in our example, you begin with a gap of seven, the initial comparison would be between card #8 and card #1, then cards #9 and #2, and so on to the end. This effectively sorts the ends of the array. Then a repeat of the sort with a gap of three would include the middle cards. Card #4 would be compared with #1 to start, next #5 with #2. But when you got to multiples of three, the comparison would hopscotch to the left. When you reach card #7, then comparison would be made with cards #4, and #1, unless the left card was smaller, whereupon you would stop and go to card #8. Lastly, a sort with a gap of one would find most cards within two or three places of their final position. This presort of the array is what eliminates the long incremental movement of elements that slows the plain shuttle sort.

I found two methods employed to generate the gap sequence. I named them calculated and division.

The calculated method uses the formula (in Forth):

1 BEGIN GAPFACTOR * 1+ DUP #ELEMENTS > UNTIL GAP !

where gapfactor is usually 2 or 3, and #elements the number of elements (not bytes) in the array. This generates a starting value larger than the number of array elements. This value is then divided by the gapfactor to obtain the initial gap:

GAP @ GAPFACTOR / GAP !

with subsequent gaps obtained by the same routine. The gap sequence from a gapfactor of 2 starts with:

1 3 7 15 31 63 127 255 511 1023 ....

and for a gapfactor of 3:

1 4 13 40 121 364 1093 .... .

The advantage of this method is that the gap sequence is constant for a given gapfactor, and that the sequence ends in one. The disadvantage is that the number of elements must be kept below the value where the starting value would overflow the integer type. For a 16-bit integer, this is 65534 for gapfactor of 2, and 29523 for a gapfactor of 3.

The division method obtains the initial gap by dividing the number of elements by the gapfactor, and the remaining sequence by further divisions. The advantage of this method is simplicity, and the ability to have a larger number of elements for a given gapfactor (e.g. 65535 elements with a gapfactor of three). The disadvantage is that the gap sequence now also varies with the number of elements, so you can no longer be certain that the sequence has no multiples, or that it will end in one. You may need to add a test (e.g. 1 MAX) to ensure that the sequence ends in one, or list the sequence to make adjustments. As a result, this method is recommended only where the number of elements is fixed and the gap sequence checked and found valid.

Making Shell Sort in Forth

One of the great strengths of Forth is that it makes it easy to explore variations in implementing an algorithm. One of the great problems with Forth is that it's sometimes hard to stop exploring.

Shell sorts #1 to #4 are variations on the algorithm in Sedgewick. This is based on the shuttle sort with a calculated gap using a gapfactor of three. This sort requires three nested loops. The outer loop takes care of the gap size calculation, setting up the middle loop, and checking for the end of sort. The middle loop handles the incremental sweep of the sort. The inner loop keeps very busy not only with comparisons and shifts, but with testing for the array end.

Shell#1 is a fairly straightforward translation of the Sedgewick's pseudocode. Variable SV holds the datum to be sorted, and SW its address, which is the open slot. Two tests are required to stay in the loop. First to check that after subtracting the gap, the address is still in the array, and second to compare values. If both are true, then the lower value is shifted to the slot, the index address decremented by gap, and the loop continued.

In Shell#2a, I used the stack instead of a variable to hold the location of the open slot, and to temporarily hold the first flag value.

I next experimented, in Shell#3, with the use of the DO +LOOP. This loop would do the backward indexing needed for comparisons. However, as the index is the address of the upper value, it will not test if the lower value gone off the array end. So a separate test is required. The datum to be sorted is on the stack. Shell#4 worked out the exit problem by using the address of the lower value as the index, so that it will do double duty as the gap step and end of array test. The stack holds both the datum and its address. The address is adjusted so that at exit the proper one is presented to the routine after +LOOP that puts the datum in the slot.

Shell#5a must be the most widely known of the Shell sorts, as I found variations of it BASIC, Pascal, and Forth (see refs. 1,3,5). This one is loosely adapted from a version by Mark Manning. It is based on the Sift sort with a calculated gap using a gapfactor of two, and is closest to Shell's original description. Because the exchange always places the data back in the array, the extra steps to keep track of the 'floating' datum are not necessary. Somehow, this does not simplify the code. Shell#5b is Shell#5a with a gapfactor of three.

Shell Sort Shuffle

To test the various Shell sorts, I used a simplified version of the test suite in "The Challenge of Sorts". This provides for a 1024 element integer array that can be filled with eight different data patterns (Figure 3). Since my 5 MHz. Kaypro II doesn't have a built in clock, the times (in seconds) are the best I can do by hand. All times should be considered relative, as a comparison between algorithms, rather than as an absolute indication of performance on your computer system. If you have afast computer with a clock, I'd recommend you adapt the full "Challenge" test as it not only automates the test process, but provides statistics on the comparisons and exchanges.

To test the sorts, first run SETUP to initialize the random number generator. Next run RAMP, SLOPE, WILD, SHUFFLE, BYTE, FLAT, CHECKER, or HUMP to fill the array with the data pattern you want to sort. By using the word at the bottom of source screen, e.g. s2a, the sort will begin. At the completion of the sort, a beep sounds and the array checked if it is properly sorted.
The time tests for these Shell sorts are summarized in Figure 4. The Shell sorts based on the Shuttle sort with a gapfactor of three proved faster than those based

on the Sift sort. The fastest version is Shell#2a.

The Sift based Shell sort requires an extra fetch and store within the inner loop. You pay for it with an 18% penalty in speed even when compared to a shuttle based sort using a similar loop structure. Putting the exchange code inline doesn't help. Shell#5b, with a gapfactor of three, shows slightly better times than Shell#5a. The greatest time loss is in the implementation of all those nested loops (there are three). The worst performance is by routines using the DO +LOOP as the inner loop. DO +LOOPS may look cleaner on paper than BEGIN UNTIL and BEGIN WHILE REPEAT loops, but they hide a great deal of time consuming code.

A great deal of energy has been expended to discover the most efficient gap sequence, but how to find the best one? I set up Shell#2b to experiment with calculated gaps derived from gapfactors varying from 1.5 to 5.5. Figure 5a gives the times and Figure 5b, the gap sequence.

The times for random data (Wild, Shuffle, Byte) chart a broad corrugated bottomed 'U', with the left side rising at a gapfactor of two at the low end, and 3.7-4.0 on the right.

A slight downblip in time at gapfactor 2.3, shows the sequence (17 7 3 1) that works best for random data. Upblips at 1.9, 2.7 and 2.9 show the deleterious effect of data multiples in the terminal sequence. For a gapfactor of 2.7, a group of gaps goes 27 9 3, and this tends to resort (or rather, not sort) the same data. This time rise proved misleading, as the times declined again and remained fairly level for unsorted data until 4.0 and above. That the gapfactor of 4.0 sorted faster than one of 2.0 was a surprising revelation.

The gapfactor time data also reveals why predicting the performance of the Shell sort is so difficult -- it depends in part on the data. Data that is flat or already sorted (Flat, Ramp) does increasingly better with larger gapfactors because the shorter gap sequences more rapidly enter the final sort phase. Pure random data (Wild, Shuffle, Byte) sorts most efficiently with the shorter gap sequences ending in 3 1, or 4 1. Partially sorted data (Checker, Slope, Hump) did well with a wide range of gapfactors of 2.1 to 3.7 (except 2.9), that generate sequences ending in 3 1, 4 1, or 5 1.

The times of the presort and the main sort phases act inversely to one other. At gapfactors of 2.1 or less, the gap sequence becomes too long and finegrained, so that excessive time is spent in the presort. At gapfactors of 4.5 and above, the opposite is true, and the main sort is overburdened. In between, the tests reveal a broad spectrum of gap sequences that perform well.

It's obvious that any gapfactor chosen will be a compromise candidate. The winner, according to the tests, is a gapfactor of three used to generate a calculated gap sequence. It lies in the center of the time base, performs well, and requires only simple integer calcu lations.

I ran time tests on a Sift sort version and obtained a similar pattern, though the times were longer.

Future Shell

Predicting sort times for large arrays is helpful in the early design phase of a program. I ran some tests for larger arrays (Figure 6) so you could get a feeling of time increases as arrays grow larger. In his article, Shell plotted time verses array element number on a log-log graph to obtain a straightline with all the data points dead on the line. I like that kind of graph. On mine, the data points tend to scatter like buckshot fired through a rusty barrel. I followed his lead and used the following equation to calculate the slope of the line for the Wild pattern:

$$\log(t2/t1)/\log(n2/n1) = 1.23 \quad \text{(approx.)}$$

where t1 & n1 are the times and number of the smaller array, and t2 & n2 are for a larger array.

To estimate time for a still larger array, pick the time and number (t1 & n1) of a known array and use the equation:

$$t2 = t1(10\^{}(\log(n2/n1)(1.23)))$$

to get the time (t2) of the array with element number (n2).

I used this equation (based on a Wild pattern slope) to calculate times (in parenthesis) for an array of 5800 elements. The times are less accurate for other data patterns when compared to the empirical test times.

According to the formula, an array of 25,000 elements, if you could stuff it into RAM, would take 13 minutes to sort on my system. A fair time, but considerably better than an efficient Bubble sort which would take 21 minutes to sort just 1024 elements.

Conclusion

This article contains everything you ever wanted to know about the Shell sort (and more). I've also described several Shell sorts in Forth, some never seen before, and put them through their paces for your benefit and pleasure.

References

1. Dwyer, Thomas & Critchfield, Margot,'' Basic and the Personal Computer'', Addison-Wesley Pub. (1978) p. 219-221.

2. FIG Staff,"The Challenge of Sorts", Forth Dimension, vol. 11, no. 3 (Sept/ Oct 1989) p. 24-29.

3. Gilbert, Harry M. & Larkey, Arthur I.,'' Practical Pascal'', Southwestern Pub. (1984) p. 300-303.

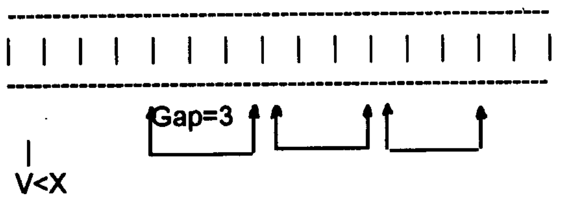
4. Knuth, Donald E.,'' The Art of Computer Programming, Sorting and Searching'', vol. 3, Addison-Wesley Pub. (1975) p. 84-95.

5. Manning, Mark I.,'' The Forth Sort'', Dr. Dobb's Journal, vol. 8, issue 9 (Sept. 1983) p. 103-108.

6. Sedgewick, Robert,'' Algorithms'', Addison-Wesley Pub. (1983) p. 97-99.

7. Shell, Donald. L.,'' A Highspeed Sorting Procedure'', Comm. ACM, 12, No.3 (1969) p. 30-32.

Figures and Listing continues on next page.



Exchange values V>X
Figure 1

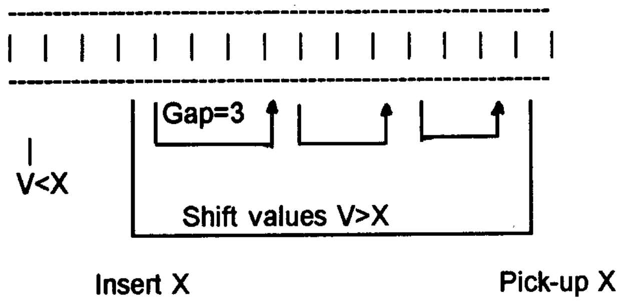


Figure 2

Test Sort Patterns

Ramp	-- ascending values, already sorted.
Slope	-- descending values.
Wild	-- random signed values.
Shuffle	-- a Ramp randomly reordered (no duplicates)
Byte	-- random eight bit values
Flat	-- a single random value.
Checker	-- two random values placed alternatly on odd/even addresses.
Hump	-- Gaussian distribution of random values.

Figure 3

Shell Sort Time Test (in Seconds)

		1	2a	3	4	5a	5b
Ramp	--	7	6	8	7.7	8	6
Slope	--	12	10.5	12.3	12	16	13
Wild	--	17.3	15.2	19	17.3	20.7	22
Shuffle	--	17.4	15.4	18	17	21.7	21
Byte	--	17	14	18	17	20	20.8
Flat	--	7	6	7.4	8	8.6	5.8
Checker	--	8	7	9	9	9	8
Hump	--	16	14	16	15	18	18

Figure 4

S#2b Variable Gapfactor Time Test (in Seconds)

		1.5	1.9	2.0	2.1	2.3	2.5	2.7	2.9	3.0	3.1	3.5	4.0	4.5	5.0	5.5
Ramp	--	14.5	10.3	9	7.8	7.8	7	6.8	6.7	6	6	5.5	5	4.8	4.3	3.8
Slope	--	16.5	13	13	12	11.3	11	10.5	10	11	10	10.2	13	10	9.8	10
Wild	--	19	17	16.2	15.2	14.9	15.2	16.3	18	15.3	14.8	15.5	16	16	17.8	20
Shuffle	--	20	17.6	17	15.2	15	15.2	16	19	15	15	16	17	16.4	17.2	28
Byte	--	19	16.8	16.8	14.4	13.8	14	15	16.2	14.5	14.8	16.2	15	15.8	16.8	17
Flat	--	14.5	10.3	9	8.3	8	7	7	6.4	6	6	5	4.8	4.1	4.1	4
Checker	--	14	11	9	9	8.2	7.3	7.3	6.4	7	6.3	8	5.2	5	6	142
Hump	--	18	15.3	15	14	13.8	13.8	14.5	15.4	14	13	12.8	13.8	14	15.3	22

Figure 5a

Calculated Gap Sequences for 1024 Items

1.5 =	709 472 314 209 139 92 61 40 26 17 11 4 2 1
1.9 =	778 409 215 113 59 31 16 8 4 1
2.0 =	1023 511 255 127 63 31 15 7 3 1
2.1 =	633 301 143 68 32 15 7 3 1
2.3 =	493 214 93 40 17 7 3 1
2.5 =	833 333 133 53 21 8 3 1
2.7 =	497 184 68 25 9 3 1
2.9 =	668 230 79 27 9 3 1
3.0 =	364 121 40 13 4 1
3.1 =	397 128 41 13 4 1
3.5 =	652 186 53 15 4 1
4.0 =	341 85 21 5 1
4.5 =	469 104 23 5 1
5.0 =	781 156 31 6 1
5.5 =	188 34 6 1

Figure 5b

Shell#2a Times for Larger Arrays (in Seconds)

# Items	--	1024	2200	3600	(5800)	5800
Ramp	--	6	15	26	(53)	49
Slope	--	10.5	26	42	(85)	73
Wild	--	15.2	40	72	(132)	130
Shuffle	--	15.4	41	73	(135)	122
Byte	--	14	37	67	(122)	114
Flat	--	6	15	26	(49)	45
Checker	--	7	15	28	(49)	60
Hump	--	14	33	58	(109)	103

(nn) = calculated value.

Figure 6

```
Screen 1
\ Shellsort Load Screen        WJR12FEB92
2 13 THRU
\S        Shell Sort in Forth
          Copyright Feb. 12, 1992
          Walter J. Rottenkolber

Screen 2
\  Data Array and Utilities      WJR12FEB92
:  CELLS  ( a -- a' )  2* ;
: 2CELLS  ( a -- a' )  2* 2* ;
VARIABLE ITEMS   1024 ITEMS !
CREATE  DATA  ( -- a )  ITEMS @  CELLS ALLOT ;
: S@  ( index -- n )  CELLS DATA + @ ;
: S!  ( n index -- )  CELLS DATA + ! ;
: EXCHANGE ( idx1 idx2 -- ) 2DUP S@ SWAP S@ ROT S! SWAP S! ;

Screen 3
\ Sort Variables, Subroutines, & Test    WJR12FEB92
 VARIABLE  GAP   VARIABLE  SV   VARIABLE  SW
: SETGAP  ( -- )   1 BEGIN 3 * 1+ DUP ITEMS @ > UNTIL GAP ! ;
: DECGAP  ( -- )   GAP @ 3 / GAP ! ;
 : TEST-DATA  ( -- )  \ Checks if data is sorted.
   DATA @  ITEMS @  1 DO  DATA I CELLS + @ SWAP OVER >
   ABORT" Data has not been sorted."
```

```
  LOOP DROP ;

Screen 4
\ Random Number Generator      WJR13JAN92
VARIABLE  SEED
: SETUP  ( – )  1234 SEED ! ;
: RANDOM  ( – n )
  SEED @  31421 * 6927 + DUP SEED ! ;
: CHOOSE  ( limit – 0..limit-1 )
  RANDOM  UM*  SWAP DROP ;
:.GAUSS  ( n – u )
  RANDOM  0    RANDOM  0  D+  RANDOM  0  D+
  RANDOM  0  D+  RANDOM  0  D+  RANDOM  0  D+
  6 UM/MOD  SWAP DROP UM*  SWAP  DROP ;

Screen 5
\ Random Data Patterns         WJR12FEB92
: RAMP   ( – )  ITEMS @  0 DO  I  I  S!  LOOP ;
: SLOPE  ( – )  ITEMS @  0 DO ITEMS @  1- I - I  S!  LOOP ;
: WILD   ( – )  ITEMS @  0 DO  RANDOM  I  S!  LOOP ;
: SHUFFLE  ( – )
   RAMP ITEMS @ 0 DO ITEMS @  CHOOSE  I  EXCHANGE LOOP ;
: BYTE   ( – )  ITEMS @  0 DO  256 CHOOSE  I  S!  LOOP ;
: FLAT   ( – )  RANDOM ITEMS @  0 DO DUP  I  S!  LOOP DROP ;
: CHECKER  ( – )  RANDOM  RANDOM
   ITEMS @  0 DO DUP  I  S!  SWAP LOOP  2DROP ;
: HUMP   ( – )  ITEMS @  0 DO  256 GAUSS  I S!  LOOP ;

Screen 6
\ Shell#1                WJR12FEB92
: SHELL1  ( – )
   SETGAP  BEGIN   DECGAP
   ITEMS @  GAP @  DO
     I  DUP SW !  S@ SV !
     BEGIN
       SW @  GAP @  2DUP - 0< NOT  >R - S@  SV @  >  R>  AND
     WHILE
       SW @  DUP GAP @ -  DUP SW ! S@ SWAP S!
     REPEAT
       SV @  SW @  S!  LOOP
       GAP @ 2 < UNTIL ;
: s1   shell1 beep test-data ;

Screen 7
\Shell#2a  (Fastest)          WJR12FEB92
: SHELL2A  ( – )
   SETGAP  BEGIN   DECGAP
   ITEMS @  GAP @  DO
     I  DUP S@ SV !
     BEGIN
       DUP GAP @ - DUP -0< NOT >R - S@  SV @  >  AND
     WHILE
       DUP GAP @ -  TUCK S@ SWAP S!
     REPEAT
       SV @ SWAP S!  LOOP
       GAP @ 2 < UNTIL ;
   s2a   shell2a beep test-data ;

Screen 8
\Shell#3                 WJR12FEB92
: SHELL3  ( – )
    SETGAP  BEGIN   DECGAP
    ITEMS @  GAP @  DO  I S@
      0 I  DO
         I GAP @ -  2DUP S@ <  SWAP 0< NOT  AND
         IF  I GAP @ - S@  I S!
         ELSE I S!  LEAVE  THEN
      GAP @ NEGATE +LOOP
    LOOP
    GAP @  2 < UNTIL ;
: s3   shell3 beep test-data ;
```

```
Screen 9
\Shell#4               WJR12FEB92
: SHELL4  ( – )
    SETGAP  BEGIN   DECGAP
    ITEMS @  GAP @ - 0 DO  I GAP @ + S@
      0 0 I DO
         DROP DUP I S@ <
         IF  I S@  I GAP @ + S!
         ELSE I GAP @ + LEAVE  THEN I
      GAP @ NEGATE +LOOP  S!
    LOOP
    GAP @  2 < UNTIL ;
: s4  ( – )  shell4  beep  test-data ;

Screen 10
\ Shell#5a  Sift Shell Sort   WJR12FEB92
: SETGAP2    1 BEGIN 2 * 1+ DUP ITEMS @  > UNTIL  GAP ! ;
: DECGAP2    GAP @ 2 /  GAP ! ;
: SHELL5A  ( – )
    SETGAP2  BEGIN  DECGAP2
    ITEMS @  GAP @ - 0 DO
      0 I DO
        I S@ I GAP @ + S@ >
        IF  I DUP GAP @ + EXCHANGE ELSE  LEAVE THEN
      GAP @ NEGATE +LOOP
     LOOP
     GAP @  2 < UNTIL ;
: s5a  ( – )  shell5a  beep  test-data ;

Screen 11
\Shell#5b  Sift Shell Sort (Gapfactor = 3) WJR12FEB92
: SHELL5B  ( – )
   SETGAP  BEGIN   DECGAP
   ITEMS @  GAP @ - 0 DO
     0 I DO
       I S@ I GAP @ + S@ >
       IF  I DUP GAP @ + EXCHANGE ELSE  LEAVE THEN
     GAP @ NEGATE +LOOP
     LOOP
   GAP @  2 < UNTIL ;
: s5b  ( – )  shell5b  beep  test-data ;

Screen 12
\ Routines for Variable Gap Tests   WJR12FEB92
VARIABLE  GAPFACTOR  \ Set Gapfactor in tenths, ie. 25 = 2.5
: SETGAP3  ( – )
   1 BEGIN GAPFACTOR @ 10 */ 1+ DUP ITEMS
                                   @  > UNTIL GAP ! ;
: DECGAP3  ( – )   GAP @ 10 GAPFACTOR @ */ 1 MAX GAP ! ;
: GGS  ( n – )  \  Generates gap sequence list 1.5 - 5.5
    CR  56 15 DO
    I GAPFACTOR ! SETGAP3 I 3 .R ." : " GAP @ U. ."  = "
    BEGIN  DECGAP3 GAP @ DUP U.  2 <  UNTIL
    CR  LOOP ;

Screen 13
\ Shell#2b – Variable Gap Test   WJR12FEB92
: SHELL2B  ( – )
    SETGAP3  BEGIN  DECGAP3
    ITEMS @  GAP @  DO
      I  DUP S@ SV !
      BEGIN
        DUP GAP @ - DUP 0< NOT  SWAP
        S@  SV @  >  AND
      WHILE
        DUP GAP @ -  TUCK S@ SWAP S!
      REPEAT
    SV @  SWAP S!  LOOP
    GAP @  2 < UNTIL ;
: s2b  shell2b  beep  test-data ;
```

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Issues 1 to 19 are currently OUT of print. To assist those who want a full collection of TCJ issues we are preparing photo-copied sets. The sets will be issue 1 to 9 and 10 to 19. Each set will be bound with a plastic protective cover.

The expected pricing will be in the $24 to $29 range (foriegn price will be $10 more). Expect TWO to THREE weeks for delivery.

Issue Number 20:

- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K

Issue Number 21:

- Extending Turbo Pascal: Customize with Procedures & Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition & 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 an Index
- The Ampro Little Board Column

Issue Number 24:

- Selecting & Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assemble 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 Real Time Clock for the Ampro Z-80 Little Board

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 & 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 in 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 Little Board
- 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 Little Board, Part 1
- 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 Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, 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 & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying 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 OS 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: 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 & How to Write Self Relocating 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 & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:

- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- 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.
- The Computer Corner

Issue Number 35:

- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner 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.
- The Computer Corner

Issue Number 36:

- Information Engineering: Introduction.
- Modula-2: A list of reference books.
- Temperature Measurement & Control: Agricultural computer application.
- ZCPR3 Corner: Z-Nodes, Z-Plan, Amstrand computer, and ZFILE.
- Real Computing: NS32032 hardware for experimenter, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.
- The Computer Corner.

Issue Number 37:

- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.
- The Computer Corner.

Issue Number 38:

- C Math: Handling Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- The Z-System Corner: Shells and ZEX, new Z-Node Central, system security under Z-Systems.
- Information Engineering: The portable Information Age.
- Computer Aided Publishing: Introduction to publishing and Desk Top Publishing.
- Shells: ZEX and hard disk backups.
- Real Computing: The National Semiconductor NS320XX.
- ZSDOS: Anatomy of an Operating System, Part 2.

Issue Number 39:

- Programming for Performance: Assembly Language techniques.
- Computer Aided Publishing: The Hewlett Packard LaserJet.
- The Z-System Corner: System enhancements with NZCOM.
- Generating LaserJet Fonts: A review of Digi-Fonts.
- Advanced CP/M: Making old programs Z-System aware.
- C Pointers, Arrays & Structures Made Easier: Part 3: Structures.
- Shells: Using ARUNZ alias with ZCAL.
- Real Computing: The National Semiconductor NS320XX.
- The Computer Corner.

Issue Number 40:

- Programming the LaserJet: Using the escape codes.
- Beginning Forth Column: Introduction.
- Advanced Forth Column: Variant Records and Modules.
- LINKPRL: Generating the bit maps for PRL files from a REL file.
- WordTech's dBXL: Writing your own custom designed business program.
- Advanced CP/M: ZEX 5.0×The machine and the language.
- Programming for Performance: Assembly language techniques.
- Programming Input/Output With C: Keyboard and screen functions.
- The Z-System Corner: Remote access systems and BDS C.
- Real Computing: The NS320XX
- The Computer Corner.

Issue Number 41:

- Forth Column: ADTs, Object Oriented Concepts.
- Improving the Ampro LB: Overcoming the 88Mb hard drive limit.
- How to add Data Structures in Forth
- Advanced CP/M: CP/M is hacker's haven, and Z-System Command Scheduler.
- The Z-System Corner: Extended Multiple Command Line, and aliases.
- Programming disk and printer functions with C.
- LINKPRL: Making RSXes easy.
- SCOPY: Copying a series of unrelated files.
- The Computer Corner.

Issue Number 42:

- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Screen Character Attributes.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
- Real Computing: The NS 32000.
- The Computer Corner

Issue Number 43:

- Standardize Your Floppy Disk Drives.
- . A New History Shell for ZSystem.
- Heath's HDOS, Then and Now.
- The ZSystem Corner: Software update service, and customizing NZCOM.
- Graphics Programming With C: Graphics routines for the IBM PC, and the Turbo C graphics library.
- Lazy Evaluation: End the evaluation as soon as the result is known.
- S-100: There's still life in the old bus.
- Advanced CP/M: Passing parameters, and complex error recovery.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 44:

Animation with Turbo C Part 1: The Basic Tools.
- Multitasking in Forth: New Micros F68FC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
- DosDisk: MS-DOS disk format emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Real Computing: The NS32000.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.

+ The Computer Corner

Issue Number 45:

- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- The Z-System Corner: Using scripts with MEX.
- The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
- Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jetfind.
- Animation with Turbo C: Part 2, screen interactions.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 46:

- Build a Long Distance Printer Driver.

Using the 8031's built-in UART for serial communications.
- Foundational Modules in Modula 2.
- The Z-System Corner: Patching The Word Plus spell checker, and the ZMATE macro text editor.
- Animation with Turbo C: Text in the graphics mode.
- Z80 Communications Gateway: Prototyping, Counter/Timers, and using the Z80 CTC.

Issue Number 47:

- Controlling Stepper Motors with the 68HC11F
- Z-System Corner: ZMATE Macro Language
- Using 8031 Interrupts
- T-1: What it is & Why You Need to Know
- ZCPR3 & Modula, Too
- Tips on Using LCDs: Interfacing to the 68HC705
- Real Computing: Debugging, NS32 Multi-tasking & Distributed Systems
- Long Distance Printer Driver: correction
- ROBO-SOG 90
- The Computer Corner

Issue Number 48:

- Fast Math Using Logarithms
- Forth and Forth Assembler
- Modula-2 and the TCAP
- Adding a Bernoulli Drive to a CP/M Computer (Building a SCSI Interface)
- Review of BDS "Z"
- PMATE/ZMATE Macros, Pt. 1
- Real Computing
- Z-System Corner: Patching MEX-Plus and TheWord, Using ZEX
- Z-Best Software
- The Computer Corner

Issue Number 49:

- Computer Network Power Protection
- Floppy Disk Alignment w/RTXEB, Pt. 1
- Motor Control with the F68HC11
- Controlling Home Heating & Lighting, Pt. 1
- Getting Started in Assembly Language
- LAN Basics
- PMATE/ZMATE Macros, Pt. 2
- Real Computing
- Z-System Corner
- Z-Best Software
- The Computer Corner

Issue Number 50:

- Offload a System CPU with the Z181
- Floppy Disk Alignment w/RTXEB, Pt. 2
- Motor Control with the F68HC11
- Modula-2 and the Command Line
- Controlling Home Heating & Lighting, Pt. 2
- Getting Started in Assembly Language Pt 2
- Local Area Networks
- Using the ZCPR3 IOP
- PMATE/ZMATE Macros, Pt. 3
- Z-System Corner, PCED
- Z-Best Software
- Real Computing, 32FX16, Caches
- The Computer Corner

Issue Number 51:

- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks--Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference as a Technique for Intelligent Real-Time Embedded Control
- Real Computing, the 32CG160, Swordfish, DOS Command Processor
- PMATE/ZMATE Macros
- Z-System Corner, The Trenton Festival
- Z-Best Software, the Z3HELP System
- The Computer Corner

Issue Number 52:

- YASBEC, The Hardware
- An Arbitrary Waveform Generator, Pt. 1
- B.Y.O. Assembler...in Forth
- Getting Started in Assembly Language, Pt. 3
- The NZCOM IOP
- Servos and the F68HC11
- Z-System Corner, Programming for Compatibility
- Z-Best Software
- Real Computing, X10 Revisited
- PMATE/ZMATE Macros
- Controlling Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance Single-Board Computer
- The Computer Corner

Issue Number 53:

- The CPU280
- Local Area Networks
- Am Arbitrary Waveform Generator
- Real Computing
- Zed Fest '91
- Z-System Corner
- Getting Started in Assembly Language
- The NZCOM IOP
- Z-BEST Software
- The Computer Corner

Issue Number 54:

- Z-System Corner
- B.Y.O. Assembler
- Local Area Networks
- Advanced CP/M
- ZCPR on a 16-Bit Intel Platform
- Real Computing
- Interrupts and the Z80
- 8 MHZ on a Ampro
- Hardware Heavenn
- What Zilog never told you about the Super8
- An Arbitary Waveform Generator
- The Development of TDOS
- The Computer Corner

Issue Number 55:

- Fuzzilogy 101
- The Cyclic Redundancy Check in Forth
- The Internetwork Protocol (IP)
- Z-System Corner
- Hardware Heaven
- Real Computing
- Remapping Disk Drives through the Virtual BIOS
- The Bumbling Mathmatician
- YASMEM
- Z-BEST Software
- The Computer Corner

Issue Number 56:

- TCJ - The Next Ten Years
- Input Expansion for 8031
- Connecting IDE Drives to 8-Bit Systems
- Real Computing
- 8 Queens in Forth
- Z-System Corner
- Kaypro-84 Direct File Transfers
- Analog Signal Generation
- The Computer Corner

	U.S.	Foreign (Surface)	Foreign (Airmail)	Total
Subscriptions				
1year (6 issues)	$18.00	$24.00	$38.00	________
2 years (12 issues)	$32.00	$44.00	$72.00	________
Back Issues				
#20 thru #43	$3.50 ea.		$5.00 ea.	________
6 or more	$3.00 ea.		$4.50 ea.	________
#44 and up	$4.50 ea.		$6.00 ea.	________
6 or more	$4.00 ea.		$5.50 ea.	________

Back Issues Ordered:

____

Subscription Total ________

Back Issues Total ________

California state Residents add 7.25% Sales TAX ________

Total Enclosed ________

Name: ________________________

Address: ________________________

____

____

My Interests: ________________________

____

Payment is accepted by check or money order. Checks must be in US funds, drawn on a US bank. Personal checks within the US are welcome.

TCJ The Computer Journal

P.O. Box 535, Lincoln, CA 95648-0535
Phone (916) 645-1670

SPECIAL REPORT

All PC Users

CP/M, ZCPR & PCDOS

Z AT Last!

By Lee Bradley

Z AT Last! Z AT Last! Thank Cran And Others! Z AT Last! Z-System On AT-Class Computers Under MYZ80

On the 1st day of the *8th* month, 1992, Simeon Cran brought forth MYZ80100.ZIP, a 58k package which lets AT-class machines run Z80-class software. Not only does Cran's emulator run CP/M programs, but it also runs Z-System programs!

TCJ readers may already be aware of other CP/M emulators (22NICE, Z80MU5, ZSIM12). Each of these emulators has its advantages and disadvantages. With the exception of ZSIM12, none of these emulators runs Z-System. ZSIM12 needs to use a 5 1/4" floppy drive at start up (a distinct disadvantage if your drive just died (a common event around here lately) or you don't have a 5 1/4" drive. Like MYZ80, ZSIM12 uses a real CP/M 2.2 compatible BDOS. The *only* thing that's emulated is the Z80 processor itself, not CP/M.

Although ZSIM12 has the advantage of being able to read and write files on CP/M floppies (CPM86 and Osborne worked for me but I had trouble trying to read others) and will run on pre-AT-class (ie. XT) computers (MYZ80 only runs on post XT-class IBM type computers), MYZ80's features (speed (due to the higher Mhz rating of 80x86-based computers), hard disk file support (3 8-meg (!) CP/M "partitions" plus a RAM disk) and the growing support from Cran and the Z-System community) make it the emulator of choice.

As CP/M machines die and IBM machines get produced, this development comes as exciting news to those who have remained active in the 8-bit community. Few AT-class machine owners are aware of the advancements that have taken place in 8-bit operating systems over the last ten years. Now that these advancements are available to them, I think a series of short articles which talk about how to get the software and how to use it has a place. This introductory piece will hopefully generate more interest in future articles. (I've encouraged Simeon to write something and hope he takes me up on it).

What You Need And How To Get It

As of this writing (September 1992) the first thing you need is the file MYZ80xxx.ZIP. (The first release had xxx = 100 but last I looked Simeon was talking about version 1.02). You may send for it directly by writing the author. MYZ80 is user-supported software. Author Simeon Cran will send you the latest version of his emulator and listen to your questions when you send him the registration fee ($23.50 US). The address is:

MYZ80 Registration
Software by Simeon
2 Maytone Avenue
Killara N.S.W. AUSTRALIA 2071

MYZ80xxx.ZIP is available for download from electronic bulletin boards. I run such a board in Connecticut, Z-Node #12, (203) 665-1100. If you don't care to or are not equipped to download, a 720k 3.5" disk with MYZ80xxx.ZIP *plus* 20 or so of the latest versions of key Z-System utilities may be obtained by sending a $10 check to:

Small Computer Support
24 East Cedar Street
Newington, CT 06111

I strongly encourage you to register your copy of MYZ80 with the author.

The second thing you need is the NZCOM package, an automatic, dynamic, universal Z-System, a CP/M 2.2 compatible operating system. To order it, send a check for $49 plus $3 S&H to:

Sage Microsystems East
1435 Centre Street
Newton Centre, MA 02159-2469

In the few weeks that MYZ80 has been out, bug reports have been sent to Cran and they are being addressed. Howard Goldstein discovered two bugs, one of which was serious. The EXPORT command (used to transfer files from the A.DSK, the simulated A: drive, to the DOS environment) failed for files larger than 16k. DOSDIR (used to show the directory of files in the DOS environment) did not show read-only files. Jay Sage's Z-Node in the Boston area recently received a file named MYUTES01.ZIP with repaired EXPORT.COM, DOSDIR.COM and IMPORT.COM plus a README file.

Tom Mannion has reported some problems running WordStar 4 on large files and several people have been struggling trying to get Z3PLUS going (the Z-System for CP/M 3.0 machines, which Cran is running without a hitch). So there are still some unresolved problems. On the plus side, Steven Hirsch has managed to get a ZSDOS clock driver to work under MYZ80 and Tom Mannion has released an IMP that works under MYZ80.

I strongly encourage you to take advantage of both Cran's new product (MYZ80) and the Z-System it lets you use on AT-class (80x86) computers. It is so exciting to me that CP/M, where "it" all started, has come full circle and is now usable in a much more powerful version (Z-System) on a much bigger and faster machine. Preliminary runs of benchmark programs show that programs run from slightly slower to much faster than they run on a standard CP/M machine running at 4 Mhz (depending on what AT-class computer you use). There is a wealth of software available for Z80-based machines and with MYZ80, new applications will increase.

Bio:

Lee Bradley is Sysop of Z-Node #12 in Connecticut (203) 665-1100. He published the Computer And Humor 'Zine "Eight Bits And Change" for two years and will send you all 12 back issues if you send him $40. His address is 24 East Cedar Street, Newington, CT 06111

Computer Corner

By Bill Kibler

Regular Feature

Editorial Comment

Universal OS

Z180/PC

Herbert Johnson, who is now our resident S-100 support person, suggested building a Z180 based system for the PC bus interface. At first the idea just did not ring any bells in the old gray matter (brain for those who do not watch Detective Perot.) I finally came to my senses and decided it was a very good idea. Here are some of the ideas and reasons behind the project.

The magazine needs a product which it can provide for users that allows them to learn new ideas and concepts. Collectors of older systems (my CLASSIC computers) can use their old machines, but what about new people looking for a cheap and simple way to get started. Yes, clones are cheap, but their design leaves little for experimenters and beginners. Clone type I/O cards, especially the swap meet type are cheap and plentiful. The older designed I/O cards are for the most part simple and straight forward. Newer clone designs are seldom simple and would not be practical for hardware learning tools.

When looking at software, we see that CP/M and it's variations are simple, straight forward, well documented, and in some cases the entire source code is available. People wanting to roll their own system can do so using any Z80 based computer. With ZCPR it is possible to have many new tools, as well as programs that allow real work on the system, not just learning projects.

The design that has been proposed is a Z180, some RAM and ROM, and maybe serial I/O for a debug terminal. That is all that would be on the card. It would be a half size XT bus format card. It would talk to the bus as if it were a regular 8088 CPU. That would allow use of all the XT style I/O interface cards. Cost is projected at under $100 for bare board, ROM, and any PALS or special devices. Full running and tested boards would cost $200 or slightly more.

What Herbert wants is your feedback. At this point we have had some positive and negative feedback. The question is, does this product sound better than the YASBEC? Does this product have solutions for needs you can't get elsewhere? Would you buy one? Could your business use one? Would they work in your embedded control system?

Drop me or Herbert a note and let us know what you think. I am on GENIE as B.KIBLER and Herberts address is listed in the DR S-100 column.

I am interested in Windows NT due to the closeness of it to an universal operating system.

Universal OS

For the old time readers, you know I have been talking about a Universal Operating system for some time. For our beginners, the concept is rather simple. Operating systems are defined (or lets say I define) as the control program that allows the hardware to talk to the rest of the world. They typically do this through the file system or I/O structure. Commands are entered at the keyboard to cause the system to use a program in the disk system.

An important function of any operating system is hiding the hardware from the running program. Windows NT does that and much more. System calls represent requests by the running program, that the operating system then translates into actual hardware calls. NT is built with the idea that it will reside on many hardware platforms.

Unlike my universal OS, Windows NT is written all in ''C'' code. Moving from hardware platforms will require complete recompilation of the code. With that fault out of the way, how is this different than other OS. They are doing the internals pretty much as I had suggested some years ago (multiple layers of translation and redirection.)

In Windows NT it becomes real appearant that knowledge and experience about how to do operating systems has reached a peak. By that I mean, the knowledge is rather universal. If you want to make an OS that will work across many platforms, there is only one or two ways you can do it. Thus, Microsoft has designed the NT to use pretty much the current thinking in OS design. What is that design? Modular and kernel based is the rule.

In Windows NT, the idea is separation from hardware, and yet provide many features of several operating systems (compatiblity). To do that you need a kernel design that high level requests talk to, and it interpets into hardware request that it gives to the hardware I/O modules. More recently the ability to change I/O interfaces at will has meant structuring that side of the kernel interface much like the high level has always been structured.

At this point an overview description might help put some sanity to the othewise insane discussion so far. Windows NT is suppose to be compatible with old and new versions of MSDOS. So if we start from a running program that thinks it is on a DOS machine, what happens. Suppose we output a character to the screen. In PC clone machines, usually the program would write directly to the screen memory for speed reasons. NO longer possible.

In NT, the DOS operating system is actually an emulator and so the first step will be to convert the DOS calls (the request to send the character to the screen) into instructions that this operating system understands. From there it then gets tranlated into how the DOS system is being handled. Say our DOS is actually a small window of many windows. The intermediate step then attaches information to the request that tells the system which window it is in. Should we be multitasking at the time, the time slice and priority of the task gets added. In short the system has to take a simple request and fit it into the complex system structure.

At some point in the process, the interface has done all the attaching and it now must work it's way to the hardware screen. After interfacing, the structure is given to the kernel, which inspects various layers of information and determines which screen process gets the data and if one of the many other I/O process (could go to file spool if process is not currently in a window display) might also get a copy of the data.

The screen process could look at the structure and assign a font to the character. The window the display is in may then get checked and our charter gets placed in the bit mapped location associated with that window. So what would have been a simple direct write or data movement now has become multiple layers of tranlation and conversion. I think that Microsoft has no alternative if they want an universal OS that can support previous systems. Add to that the fact that so many of these systems have completely different ways of handlng data and calls. Remember that Windows NT is suppose to support PC/XT versions of MSDOS, Windows programs, OS2, and UNIX as well.

My universal system is conceptually the same, however I use Forth's dictionary structure and platform independence to turn a very conplex concept into a simple reality.

FORTH the universal OS?

Now Windows NT is designed for the latest and fastest computers. As you saw how each layer must translate and interpret commands, it is very appearant that only big and fast computers can do that in reasonable amounts of time. I want an universal OS that can run on 8031s and up. Not possible you say, read on.

My concept of the layered approach uses Forth's dictionary as the key. Forth kernels look up the next word or operation to be performed in a dictionary of words. Actually the dictionary is a list addresses that contain the actual code that performs the task (or another list of lists.) Where my kernel would differ is in words and or operations that are too complex or involved for the 8031 to do, the kernel simply takes the word and stack frame or information and passes it to another processor.

My concept is really an extension to the way Forth has always done things. For embedded free standing tasks, an operating system is not and never has been needed. Most modern applications however, tie many embedded controllers together with some server or master computer. That master system usually has horse power to spare. Forth in the past would not handle words or requests not in the dictionary. What would happen now is the small kernels would assume any request not in the dictionary would be handled by the server or master system.

Conceptually it is very simple and rather easy to impliment. Lets take EFORTH as a basis for a conceptual system. We put a 6805 version of EFORTH (only takes about 31 unique words to create it) and put it into an intelligent door controller. Our door controller has a keypad and a two line LCD display. Entry is by entering a code squence that is sent to the master for verification. When things are ok, the display acknowledges that fact and our 6805 unlocks the door.

The code is rather simple, read keypad, packet data, send to host, and display or perform host requested commands. Where the OS concept comes into play is system maintenance. Suppose we have some mechanical problems with the door release mechanism. The maintenance person does not want to enter and reenter the password dozens of times to check out the device. Instead, simply enter a special code sequence and password. The host then puts the system into local control mode or the local OS takes over.

At this point the maintenance person can use regular Forth words to turn bits on and off. Other options are communications with host, such as asking the host to send the command as well. Operation as a mini-terminal is possible if operators or other technicans are at the host or other door sites. Suppose some pin numbers are needed and stored on the servers file system. Normally our tech would have to go to the host and print them out. Now they simply do a normal open for read request (like a DOS TYPE and MORE command) and then use keypad keys to page through the file. Our OS of course has no idea of these commands, but instead simplly passes them up the chain and sends the answers back for display.

Of course you can see that the real power is in the host system. If done as I see it, other advantages to even the host are possible. Multi-processor operations are now possible by simply having some dictionary words send their request to other processors and not the host.

Well as you can see, my idea conceptually is similar to Windows NT, but mainly scaled down for real time work in a real world environment. Now my operating system could be done by anyone with a few bucks and some nights learning the insides of Forth.

Well it looks like my other comments will have to wait till next issue. Till next time....Bill.

Market Place

TCJ The Computer Journal

TCJ MARKET PLACE

Advertising for small business

First Insertion:	$50
Reinsertion:	$35

Rates include typesetting. Payment must accompany order. VISA, MasterCard, Discover, Diner's Club, Carte Blanche, JCB, EuroCard accepted. Checks, money orders must be US funds. Resetting of ad consitutes a new advertisement at first time insertion rates. Mail ad or contact ,

The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

SUPPORT OUR ADVERTISERS TELL THEM "I SAW IT IN TCJ"