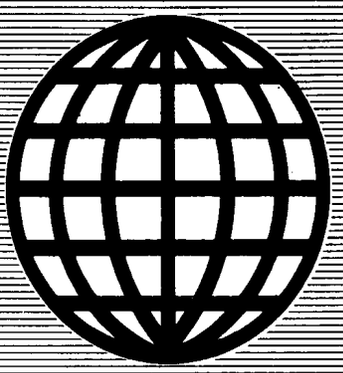


The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 62 July/August 1993 US$4.00

SCSI EPROM Programmer

Z-System Corner

DR S-100

Real Computing

Center Fold

Reminiscing and Musings

Modem Scripts

Moving Forth Part III

The Computer Corner

ISSN # 0748-9331

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Bill D. Kibler

Technical Consultant
Chris McEwen

Contributing Editors
Herb Johnson
Charles Stafford
Brad Rodriquez
Matt Mercaldo
Tim McDonough
Frank Sergeant
JW Weaver
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.

Subscription rates within the US: $24 one year (6 issues), $44 two years (12 issues). 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

TCJ The Computer Journal

Issue Number 62 July/August 1993

EDITOR'S COMMENTS

Welcome to issue #62 and what a surprise! Many surprises await you inside this issue, besides coming to you on time for a change. Yup, we did it, caught up at last. When issue #61 was done, I had enough to do #62 right away, and did. Trying a new printer, and oh yes, taking 2 months off from work helped some as well.

I have published our schedule in the past (and will again, space permitting) of when *TCJ* goes to press. As a reminder to all (especially my writers) it goes something like this. For a Sept/Oct issue, I want the material in my hands by mid August, that is two weeks before the first day of the listed month. I put the magazine together on the first day of September (or the named month of publication) and have it at the printer by the tenth. I generally will get it back by the 20th, with the post office getting it by the 25th. That means you should get a September/October issue before October 1. That also means the information in *TCJ* should not be more than 30 days old (45 max)!

Inside Surprises

This issue is special in other ways, with three feature articles and more 10th year comments. We start out with Rick Rodman talking about using EMAIL and a glossary of internet terms. Herb Johnson is back from moving to fill us in on S-100 buses and buses in general.

Summer is a time for vacations, and my catch up issue caught several of our regulars gone from home. Chuck Stafford is preparing an article on customizing CP/M and JW Weaver plans to catch up next issue on user groups. I had so many special items this time, I put CP/M notes inside my Computer Corner. Next time I promise to have a full discussion of CP/M for you beginning hackers.

Talk about vacationing but still supporting Z-Systems, is none other than our world trotting expert Jay Sage. Live from Israel via internet is Jay's column on what is for sale at Sage Microsystems East. Jay fills us in on price changes and just what you get for your money. It is just enough information to make you more curious, at least that is what happened to me. I plan on ordering several programs as soon as Jay gets back (before you get this issue), now that I know what they can do for me. Actually Jay has covered all these items in his past columns, but it is not often I can get him to talk about them all in one issue. Good going Jay, and hope you enjoyed your well earned time off.

Special Features

The surprises just keep coming, with a special editorial by Frank Sergeant. Due to my problems with subscriptions, Frank did not get his issues on time, and ended up setting on two articles. Well here they are, a special 10th Year Anniversary review in which he talks about his trials and such in producing Pygmy Forth. There is some good words in there about doing shareware or public domain development (also check out the Parker letter in Reader to Reader). Frank then muses and talks about everything except the kitchen sink (only because they haven't put computers in them yet, or have they?)

Our hardware and SCSI people will be happy with Terry Hazen's SCSI EPROM programmer. This issue brings you the hardware design, while next issue will have the software discussion. Terry has done some great work here, by keeping it simple and straight forward, most of our readers should be able to build it. Hopefully next issue will fill in any gaps in your understanding as we cover the software.

Our ever moving forward Brad Rodriguez is back with installment 3 of his Moving Forth series. He takes this issue to cover words that usually only the more advanced Forth person might use. For you beginners, his discussion should make you into intermediate programmers in one lesson (notice I didn't say "easy"). Brad also found some minor flaws in his past work which he corrects this time (also shows that you readers have been reading his stuff!)

Almost last is a promised discussion by Walter Rottenkolber on scripts in Forth for modems. Had planned on running this last issue, but ran out of space. With a few people on vacation, Walter gets to fill in and that he does. His fillers are "SFT's" or Stupid Forth Tricks (his choice of words). These were intended to be page fillers, but I was getting such a collection that I decided to lump them all in this issue (as I know Walter will send more).

Now for the last word, my Computer Corner looks at moving CP/M, LANS, and what language is appropriate if there is such a thing.

Business Note

This month I have started using post cards for renewal notices. John Hall of FIG indicated it took three notices to get people to pay. Well I will be sending you a pre-notice, a this is it, and you've lost it notices. Hopefully only one will be enough to get you to renew. Please check your mailing label each issue, as I will be changing it's appearance and as always it will have your expiration issue boldly displayed. I will be updating and changing databases which will require a few minor changes. Please check your labels and let me know if something is not correct.

Happy Reading, Bill Kibler.

READER to READER

Letters to the Editor
All Readers
MINI Articles

Hi Bill -
Here's that "Reader to Reader" letter I promised from issue #60.

I'm eagerly awaiting issue #61, to get a look at Walter Rottenkolber's modem script language in Forth. I have wanted such a thing many times, but without the inspiration (or gumption) to sit down and do it. Walter sounds like he has Just The Thing.

And I, for one, am very interested in Art Carlson's proposed articles on measurement, control, and particularly motion control. In my "old" *TCJ*s Art's articles are among the most useful...I'm still learning how to interface steppers and servos!

To give some perspective: I recently returned to university to fill in some deficiencies in my 1970's computer science and electrical engineering education. There were three hot topics in which I felt lost: artificial intelligence, digital signal processing, and robotics. All of the stuff written about these is either completely moronic (e.g., how to choose a commercial AI product), or up in the stratosphere with a huge amount of knowledge (and jargon) assumed. Well, I've tackled AI, and I understand the principles of DSP (although I've not yet done a DSP project). But university robotics courses seem to be either about a) machine vision, or b) how to program commercially made industrial robots. Phooey! I want to learn how to build my *own* robotics!

I definitely like the direction you are taking *TCJ* -- tinkering and "classic" computer support -- and intend to write some more hardware articles, soonest. I approve of your decision to make the articles more accessible to newcomers, and I'm working hard to change my style.

In the meantime, I want to encourage other *TCJ* readers to begin writing about what they're doing. Often, a project you think trivial is of vital interest to someone else! (Like Walter's modem script article is to me!) And even if you have only a partial understanding of your subject, you can write an article which is helpful to a beginner. Sometimes your article is *more* helpful, simply because it's not written from the Ph.D. level.

Finally, I'm really enjoying "Dr. S-100," "Mr. Kaypro," and the new "Support Groups for the Classics." I have several S-100 systems I'm hoping to piece together, and I've just acquired a few Kaypros. When I get one running -- Real Soon Now, with *TCJ*'s help -- I expect to become a regular reader of "Z-Systems Corner." (P.S. I still relish "The Computer Corner" and "Real Computing," too.)

Regards,
Brad Rodriguez

Thanks Brad for your letter and hard work! Actually your the only one I need to not send me so much. Your great articles are really helping, but could you put them in a bit more "BITE" size format. I keep putting things off, just to get all of your stuff in. Well one item missing from #61 was the modem scripts (in this issue) and I guess I could blame it on you (your big articles) but actually it just slipped away. I am trying a new printer who promises to print larger issues for same price, so only weight will now limit the number of pages I can use (need to stay under 6 ounces else cost to mail doubles!). Thanks for all the good work, Bill.

Dear Mr. Kibler
Please excuse this rather lengthy letter. I am filled with enthusiasm and questions. I received my first issue of *TCJ* by answering your ad in *Nuts & Volts* for a free trial copy. I received your 10th Year Anniversary edition to look over. I couldn't put it down. There were articles on CP/M, Forth, S-100, software and hardware how-to's with explanations of the why (just items I've been looking for - all in one place). Hey, this kind of reminds me of the old Byte magazines I use to get back in the 70's and early 80's (how I miss them). Just the type of publication I have been looking for. Anyway, your 10th Yr. Aniv. edition couldn't have arrived at a better time. Misosys, Inc. was having a close-out sale of the TRS-80 Model 1,3,4 items and dropping support for a lot of this stuff. I got a copy of their C-complier as well as HartForth(79-Standard by A. M. Graham). Also picked up their SCSI interface card for Seagate MFM Hard Drives. Now what does this have to do with you and *TCJ*. Well, with Misosys slacking their support on these machines, a few people who still use TRS-80 Model 1,3,4 computers (Z80 cpu) are trying to develop a SCSI-2 interface (in particular for a CD-ROM).

I was looking thru your back issues listing to see what type of articles *TCJ* had been doing and found several SCSI related items. Say these/look like they may be helpful and cut down development expense (time and money). Well shortly after this, Stan Slater of *Computer News 80* had posted some info about Z280 board on the TRS MOD134 Fidonet Echo and listed *TCJ*, Jay Sage and some oth-

ers. Boy, sounds like better subscribe. So, I did. Now issue #61 has just been devoured and marked up with notes. Talking about timing. Your "Beginning CP/M" in #61, p18 has been very helpful and informative. Just where I needed to start.

Thank you for *TCJ*!
Now for the questions (related to the TRS-80 (Z80) model 4 computers).

Q1 - How can I obtain only the article reprints on the SCSI items you had in prior issues (detail list enclosed).

Q2 - I have both CP/M 2.2 and 3.0 for my computer. What is Z3PLUS, NZCOM, etc... and can I use them with the above computer.

Q3 - Do you support the TRS-80 versions of CP/M in particular CP/M Plus (3.0) by Digital Research licensed to Radio Shack and CP/M 2.2 by Montezuma Micro.

Q4 - Can I use MP/M on a single Z80 cpu and if so, where do I find it and the source code.

Q5 - In issue #60 p13, you mention that Alpha Systems (Joe Wright) has/had the right to market the 8-bit version of Turbo Pascal. What is his/their address? I have T.Pas v2.0A w/missing docs and would like to upgrade it if possible as well as to see if there is a version that will run on LS-DOS 6.3.1. by Misosys (Roy Soltoff).

Q6 - In issue #60 p23 ("Real Computing" by Rick Rodman), Minix is mentioned. Has this been ported to Z80 platforms\ or is this only for a 16-bit cpu on up. If this has been ported to a Z80 where do I Find it and if not, is there another Unix like OS for Z80 cpus.

Q7 - In issue #61 the "Center Fold" shows a Xerox 820. How does this relate to the TRS-80 Model 1,3,4 computers. I have a manual for a clock speed-up board from Holmes Engineering that list both. Also what is the "Big Board" and "*Micro Cornucopia*".
Thank you for your assistance in these matters.
Sincerely, Mike Michaels, Canton, IL.

Well Mike, thanks for the great review and letter. I am glad we were able to answer so many of your questions with just two issues. For the rest of those questions:

Q1 - Currently I do not have reprint series. I plan to start doing so in the future, but for now you must just buy the back issues. I am starting to make bound versions available (which will save some money) but you might just want to take advantage of the 15% discount and order all back issues up to your starting issue. If you just want those covering the SCSI topic, I can give you a 10% large quantity discount (pre-paid or credit cards orders).

Q2 - By getting those back issues, you will be able to read all of Jay Sages Z-System Corner where he has explained all about ZCPR for the past 8 years. Basically it is an enhanced and super CP/M replacement and should work where ever CP/M 2.2 has been running.

Q3 - We support any old classic system. Do we have special articles on TRS-80s? Not yet, but maybe you could enlighten use as it seems you have quite a collection of them. I do remember that the TRS-80 required a special version of CP/M, unless you had an adapter board that moved their ROM from low memory. Most CP/M's have interrupt vector tables starting at bottom of memory, while TRS-80's put their ROM there. Taking that in to mind, our articles and programs should work as given, and especially the Forth ones which we try to make platform independent.

Q4 - MP/M is the CP/M version of a multi-user system for single Z80s. I am not sure about new sources for it and even more unsure about a version for any of the TRS-80 models. I will have to defer to our readers for the latest on this issue.

Q5 - I believe Jay Sage was selling Turbo Pascal and may still have some. Best to call him or send him a message (see his ad on the front cover).

Q6 - Minix is available through book stores or by calling Prentice-Hall. It is available for many 16 bit systems. In "Real Computing" (issue #5)8 is information about UZI a Unix like OS for Z80 systems. You can download this from the CP/M section of GEnie as I did. A few of our readers have mentioned they might rewrite UZI for the newer Z180/280 systems with multi banks of memory. Until then the Unix for Z80 is not much to offer.

Q7 - There were several computer systems that started the microcomputer revolution. Altair and IMSAI were leaders in the S-100 or BUS based systems. Big Boards were single all inclusive (no-expansion provided) micro-systems. Each had their own group of enthusiast and magazines supporting them. Micro Cornucopia supported the Big Boards just after the computer became available (you might check a back issue of Byte where Dave Thompson had a short article on putting the Big Board Kit together and said he was startling a newsletter for them that became the Micro Cornucopia magazine, no longer printing, but we are planning on getting the rights to reprint them, soon!). Later the design became the basis for Kaypros and Xerox computers. The clock speed up (thanks for the copies of the information) would work on any Z80 system that used a similar clock circuit and had a socketed Z80 chip. What Holmes sold was a plug in board that held the old Z80 and controlled the signals from the mother board. You might checkout Chuck Staffords Kaypro support column as some of his information might cross over to TRS-80s.

Hopefully Mike, I have answered all your questions, and if not try writing to some of our writers directly. I can not stress strongly enough that you should consider buying all the back issues. The index just can't cover all the little other articles that might apply to your projects. The regular columns, like my Computer Corner, often have detailed explanations and projects that would be just what your needing.
Thanks Again, Bill Kibler.

Dear *TCJ*
I'd like to subscribe to *TCJ*. I've enclosed a money order for $24 for one

year. I guess the main reason I'm subscribing is due to Brad Rodriguez's articles. I've written to him several times in the past 5 months and he's been quite helpful in my getting a Forth running on a Color Computer 3.

Most of your articles are somewhat specific, but there seems to be enough generic material to be useful. I'm not really all that hardware oriented and I'm just now learning Forth (I'm planning on writing a chess program), but *TCJ* seems like it's worth $24.

I like the center folds. I don't really have a use for them, but I like history. At least with *TCJ* I don't have to hide that I own and use a Color Computer 3 with 512K and OS9 level 2. (I've had a CoCo since Dec 1982.) Compared to some in the TCJ, I'm using a modern computer!

Sincerely, Carey Bloodworth, Van Buren, AR.

Glad to see your appreciating TCJ! We don't have very many OS9 articles yet, but if I can get some of YOU readers to send us an article or two maybe others will become interested and vocal. I will be printing some Center Folds of GIMIX or 6809 systems later, which should be of interest to you. Actually I sometimes think CoCo's and such are more modern than PC clones, at least OS9 is closer to a real operating system than PCDOS! Welcome to TCJ! BDK.

Dear *TCJ*;
I'm in the process of reading my trail issue of your magazine, #60. I really do enjoy this type of reading. I wish to express my thanks to PesudoCorp for the trail issue offer included with their brochure.

I'm not a professional programmer nor an electronic engineer. I just consider myself as a computer hobbyist, and as such I try to learn as much as I can about all the different types of microprocessors. I do program my own systems some, I have two IBM PC/AT 386 clones and an old Z100 from Zenith Data Systems.

I had tried in the past to obtain information on the Zilog Z8000 & Z80000 but had no luck. I wish I could find a SBC based on the Zilog processors that will fit in my ISA bus computers, so far I've had no luck in this endeavor.

As I mentioned before, I am just a computer hobbyist but I'm afraid that I have taken it to the extreme. I have spent about $3,000.00 on various software tools and books which include MASM 6.0, Matrix Layout 2.0, MSC 5.1 and Mix Software's POWER C. Last but the most costly item in itself, is my own OEM License for General Software's Embedded DOS Source Adaptation Kit. I think that at the time I had more cash than brains, anyway I have used the Source Adaptation Kit to enhance my knowledge of Operating Systems (MS-DOS type) for the Intel 80x86 processors and to help teach myself how to program. I actually have my own Operating System which resembles MS-DOS 3.31 but based on Embedded DOS. It will not run MS WINDOWS or some protected mode programs yet but for real mode it works great. My OS will run some of MS-DOS 5.0 device drivers and utilities when I have it return its version number as 5.00 in the proper registers.

If any of your staff or readers/subscribers are interested in my Operating system which I call MCOS or Micro Computer Operating System for lack of something better, please feel free to call or write to me at my address above. Please no collect calls, I have just been laid off my regular job as an Electrical Technician for a contractor under contract for our local Naval Base, so now I've got to pinch my pennies.

One more thought, I don't recommend computer hobbyist to go to the extreme that I have. If they cannot get a return on their investment, then just take it easy on the pocket book and think about if you really need to purchase all of that stuff BEFORE it is purchased!! My return for my investment is the knowledge I've gained and satisfaction of having my own OS, even if I never market my MCOS to the public. My cost for royalties to are small and based on each copy produced which has been none so far, other than for myself to test. If any small companies need an Operating System for their 8088-80x86, V20-V50 type projects, check out General Software. If you think the GS price is too high (I do) then check with me. I must remind you that my programming skills aren't as good as the good folks at GS.

Thank You, Danny M. Parker, Route 1 Box 548-R, Toomsuba, MS 39364, (601) 632-1720.

Well Danny, it seems you fell pray to the PC mania. That is what I dislike about the whole PC-clone market and especially the DOS world. If you want to do anything, you had better have deep pockets. We at TCJ keep saying that if you want to learn, DON'T BUY PC, but buy "CLASSIC". You could have done all the same things for about $300, not the $3000 you spent.

I am sure a few of our readers may be interested in your OS, but unless it is simple or easy to change like Forth, I don't expect you will have much luck. Big companies who could afford your stuff, would want you to provide lots of expensive support. I really can not express how important it is for our readers to remember how easy it is to get carried away with a project and over engineer or spend on it (in your case). I am afraid what you learned was how easy your money can go away and not how computer systems worked.

Hopefully we can show you how to keep tinkering for only $24.00 a year. And yes the Z80 is a Zilog part and maybe you need to check out some back issues that cover Z180 and Z280 projects. The Z8000 and above are used in telephone switches by the thousands. With that type of sales who cares about personal computers using Z8000 CPU's. We were trying to develop a Z180 ISA board, but didn't get any reader interest. Check out MYZ80, a Z80/CP/M emulator for the PC (reviewed in #57), or check at swap meets for older Z80 plug in boards. I have one without software, but hope to find something for it soon (a Microlog Z80B).

Thanks for the words of wisdom, Bill Kibler.

Bill,
The 10th Year Anniversary issue was great! I am impressed with the changes you have made to *The Computer Journal.* Chuck Stafford's "Mr. Kaypro" column and Herb Johnson's "Dr. S-100" column are great additions to the publication.

At home I use four computers on a regular basis. I have a Kaypro 10, a Zenith Z-100, a Z-150, and a 486 IBM AT clone. I would like to see continued support for the Kaypro 10, and perhaps more for the Z100, which I also consider to be a classic computer.

As a reader, I prefer long articles to be broken up over several issues. I also have an idea for another article that I could write. While *The Computer Journal* has focused more on hardware lately, I also would like to write an article comparing high-level computer languages that are available on classic systems. I could take a particular program and write it in BASIC, FORTRAN, COBOL (yes I did write COBOL) and PASCAL. After briefly comparing the coding features of each, I could report on how well they perform when run on the same computer against the same data. We could call the article "High-Level Language Run Off." Maybe someone could follow up with a comparison of the same application written in assembly, Forth, and C?

Let me know if you think this idea is worth pursuing and if *TCJ* readers would be interested. Thanks again for the hard work you're put into *The Computer Journal*, it shows!

Sincerely, Steve Westlund. Belleville, IL.

Your article sounds great Steve, only I have gotten tired of speed based reviews. What our readers want to know is how to use their favorite language with code based on some other language. I would appreciate it if you focused on reliability, interfacing with editors, number of steps to get output, debugging tools or lack there of, and yes what type of output is produced in relation to others (bad code runs slow, but does it run at all?). How about a series of small articles, the first laying out the project and what you want the software to do. A lot of people are upgrading to laser printers, so how about a page formatting (like two pages on one) or taking some dot matrix printer's ESC codes and converting them to the laser printers different ESC codes. Use Pascal or a Pascal like pseudo code and then in each article convert that to a different language, with debugging and compiling tips thrown in. Take the last article to review and compare all the results. Interested? I am. Thanks Bill Kibler.

Gentlemen,
Enclosed is my money for.....

I also have some questions:
Are you aware of a source for information such as the original Digital Research Alteration Guide or other which shows how to tailor MOVCPM to a specific CP/M system?

I have an orphan Royal aplha Tronic CP/M 2.2 system with source listings for BIOS and MONITOR program segments, but the MOVCPM program that came with it doesn't work. (At least I can't make it work, the first statements just start reading at 5Dh, the program name, and result in an "Invalid Memory Size" error).

Are there back issues other than those above(than I am buying) which address modifying the CP/M disk system?

I would like to modify the BIOS to be able to use IBM compatible floppy disk formats.

Thank You, James M. Harper, Bellevue, WA.
P.S. I just read the letters (Reader to Reader) where you mentioned you are interested in collecting old computer source code. If you are interested in the Royal alpha Tronic, I'd be glad to send listings-I don't have the true source on disk. JMH.

Thanks for the order James. I have talked with Chuck and he is working on several articles that show and explain how to move and change CP/M. It has been several years since I put together CP/M from scratch, so I am not sure what you are doing wrong. I will research this and put together a CP/M article in the next issue to back up what Chuck says with more details and other platform information. I have several guides, but alas they are not much help if you don't already know it all. Hold on as we are planning several articles on changing both CP/M and BIOS code, like adding IDE drives. Check out Jay Sages column this issue, he mentions a program for reading PCDOS disk formats. Thanks again. BDK.

Dear B.K.,
Enclosed is my check for .... Thanks.

Bye the bye, don't let those who advocate all manner of highest tech doohickeys such as PALS, etc. convince you to change your mind on the design principles that you have set upon for the magazine. At the very least, demand the discrete logic equivalent accompany any PAL type circuitry. Likewise, hang in there on platform independent projects. I for one, have stopped fooling with strictly BUS oriented projects as I now have too many disparate systems to want such troubles. I may at any given time, want to use a peripheral with anything from an 8 bit SBC to a DUAL processor system to an IBM clone & some are portable & some not. Also a number of my systems are no longer available and I don't want to mess with them at the board level. I also use Forth due to it's portability.

Yours, Duane L. Ruck, Lima, OH.

I am sticking by my position Duane. It took a lot of talking, but I think everyone sees my point of view NOW. I am still waiting for a PAL article worth printing! I am working on Language independence now, then operating systems after that. A long hard battle, but I think I am winning converts, especially after MSDOS 6.0 came out and provided nothing NEW for your money. Learning is a hard and slow process. Stay independent! Bill.

Mail to:
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535, U.S.A.

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

JPEG & MAIL

PC-532 news, mail servers, et cetera

JPEG performance

Before I get into JPEG itself, let me explain how I had to get there.

Moving files to the PC-532 is a little strange. I don't have Ethernet, and, to my knowledge, dosdir, dosread and doswrite don't work with the PC-532 Minix. So, what I do instead is use putdisk to write a file to a 1.44 floppy, then put the floppy into the SCSI floppy on the PC-532 and use ncat to get the file. For example, for UNSSHAR.C, the steps were:

On the PC:

putdisk a8 unsshar.c

On the PC-532: ncat /dev/fd0 2460 >unsshar.c

This works much better with larger files. For example, an "sshar" format file containing all of the JPEG source, plus the uuencoded images, was 1,022,070 bytes.

Putdisk writes a file to a floppy starting at head 0, track 0, sector 1. No file system is used. Note in the example above that /dev/fd0 is used to directly access the floppy disk block by block. This works for hard disks too. Will it work for CD-ROM? It certainly may. (Need a SCSI host adapter for your S-100 or homebrew machine? Check out issue #48, where Wayne Sung published the whole thing.)

Anyhow, I haven't gotten the JPEG software working under Minix yet. I've been having trouble with Minix's make program and the C preprocessor. More bulletins as they come in.

PC-532 news

Yet another Minix user who's been trying to port Unix software is Randy Hyde. He's been following a twofold track of adding missing functions to the library and recoding some of them in assembler, for massive speed improvements. He hopes to release his improved library shortly.

In other PC-532 news, it appears that the ETH-532 Ethernet boards are becoming closer to reality. It isn't clear whether the 32GX32 chip used in this design is still available; most of the NS32 chips have, alas, been deleted from National Semiconductor parts catalogs. It was the best of all microprocessors; too bad marketing ("Intel Inside") recapitalizes philately.

Novell and Microsoft both take the "Not Invented Here" approach to standards, so Novell has formulated ODI and Microsoft has formulated NDIS. Most network cards today come with drivers for both, since they don't interwork well. I prefer NDIS, mainly because Novell still doesn't have a good Netbios. At any rate, in the PC-532 world we hope to avoid these pomposities and provide TCP/IP and real, true Netbios, both working under Minix. This may, depending on circumstances, open the possibility of supporting X Window in the near future.

Mail servers

Usenet (news and mail) users are the poor stepchildren of the Internet. As one of these, I haven't been able to use ftp (file transfer protocol, part of TCP/IP) to get any of the free software archived on various sites throughout the net. Lately, too, it seems like all of the free software is being put on ftp servers. Nobody posts (as news) much of anything anymore. My guess is that this is a result of the continuing concern over net bandwidth. Unfortunately, it has resulted in class divisions among the Internet users.

In some cases, it's possible to get software by using a "mail server". These are very cumbersome and difficult to use. Often by the time you figure out where something is and how to get it, it's been moved. It can take weeks.

Basically, how it works is like this. You find a server where the software you want is suspected of being, for example, the "hobbes" or "ftpos2" archive. Next, find out if that machine has a mail server itself. If not, you might be able to access it through an "ftp server" such as "decwrl". The server has an e-mail address. Each one uses completely different, but consistently arcane, commands, so you first need to send a mail message with the subject "help", and the first line of the message, too, saying "help". Then wait for a reply.

It's possible that "help" will not be the correct command for getting help. If not, the system will send a mail reply telling you what the correct help command is. It won't, of course, send the help, so you have to send the new message. Then wait for a reply.

Since the ftp software is divided into various directories, you need to know what directory it's in as well as the exact filename. First, get the list of the directories. The command for doing this will vary from server to server. Send the

message. Wait for a reply.

Now, looking at this list (assuming you didn't receive an error), pick out the directory or directories you think might have the software you want. Construct a message containing the command to list the directory and send it. Wait for a reply.

If you've found the file you want, now you can send a message containing the send command for it. Wait for a reply. If things go well, you'll have the file.

The ftp server is more complicated. You send a message to the ftp server containing commands which are somewhat like the commands for a mail server - specifying the directory, list commands, and so on. For example, the following is a message I sent to "ftpmail@decwrl.dec.com":

```
connect ftp-os2.nmsu.edu
chdir /pub/uploads
binary
uuencode
get pmjpg095.zip
quit
```

You'll note I already knew the host (system), directory and file.

Some of you will be saying, "What a ridiculous process! It must never work!" But it does - I have actually been able to get *two files* from a mail server. I've never been able to get anything through an ftp server, though.

Others might ask, "OK, smart guy, how would you improve it?" That's a fair question. I would make a single-message query like, "If you have a file matching this name or wildcard in any directory, send it." The sending system should know whether the file is binary or not. And if you don't think it can be done, I will be happy to write the program.

Despite whatever faults it may have, the Internet is a very good thing - a public electronic marketplace - and any such system or network will simply reflect the foibles and conceits of the people who construct, manage and use it.

Will the Real programming platform please stand up?

Unix users have rightly criticized PC platforms as being "stone-age" platforms for programming. One example is wildcards in filenames. Under Unix, if you have a command line with a wildcard in it - for example, "ls *.c" - the operating system expands the wildcard for you. You, as an application programmer, don't have to worry about that.

CP/M didn't do that, so neither do MS-DOS nor OS/2, its intellectual derivatives. Instead, each application program that needs to do this must include code for expanding wildcards. Most compilers under CP/M, like BDS C, included a "wildexp" routine for expanding the wildcards so you at least didn't have to *write* the code. However, compilers under DOS and OS/2 don't even include that small courtesy!

It's really amazing that Microsoft DOS is really up to version 6. Here's a list of glaring problems that we're still dealing with, *ten years* after the introduction to the product: You can't backspace past the end of a line. The path is limited to 128 characters. The type-ahead is limited to ten characters. F3 doesn't work all the time. XCOPY has been buggy for three major DOS versions - it still flakes out on empty directories and mishandles a single-directory source. COPY is still almost devoid of modern features - VMS's COPY command could be studied if Microsoft is running out of ideas. And people have been asking for *years* for file sizes to be displayed with commas. For example, my C: drive shows "250068992 bytes free". Quick - is that 25 megabytes or 250 megabytes?

Microsoft says DOS 7 will be 32-bit, multitasking, etc. Seeing the glacial pace at which even the slightest improvements don't get made, I'm disinclined to believe them - especially since I remember them saying the same things about DOS 4.

Such is the tunnel vision of the PC industry that pigs in a poke like Windows NT are praised sight unseen - while really powerful packages like BSD-386 and Linux are never even noticed. Oh well, it's hard to have fun when everyone's watching, anyway.

From last time

I was discussing file archivers. Since that time, I've modified my readtar program to uudecode and decompress as described. I also have a version of LU which runs on PCs. If these would be helpful to you (all in source form of course), drop me a note by any acceptable technology.

Next time

Next time I hope to have those JPEG benchmarks, and maybe some Group 3 and Group 4 ones as well. Walnut Creek CD-ROM has produced a CD of Linux, and I plan to give my initial review of that. And how about accessing a CD-ROM from Minix?

Where to call or write

BBS: +1 703 330 9049 (eves; fax during the day)
E-mail: rickr@virtech.vti.com

To assist those in understanding what Rick has said, I have reprinted this glossary. This glossary came from GEnie Unix section as ZEN.TXT. The entire topic is from a book:

Zen and the Art of the Internet
A Beginner's Guide to the Internet
First Edition
January 1992

by Brendan P. Kehoe

Should you want more information, I suggest you either download this file, or buy the book in your local bookstore. BDK.

Glossary

This glossary is only a tiny subset of all of the various terms and other things that people regularly use on The Net. For a more complete (and very entertaining) reference, it's suggested you get

a copy of The New Hacker's Dictionary, which is based on a VERY large text file called the Jargon File. Edited by Eric Raymond (eric@snark.thyrsus.com), it is available from the MIT Press, Cambridge, Massachusetts, 02142; its ISBN number is 0-262-68069-6. Also see RFC-1208, A Glossary of Networking Terms.

:-)
This odd symbol is one of the ways a person can portray "mood" in the very flat medium of computers-by using "smilies."This is 'metacommunication', and there are literally hundreds of them, from the obvious to the obscure. This particular example expresses "happiness." Don't see it? Tilt your head to the left 90 degrees. Smilies are also used to denote sarcasm.

Network addresses are usually of two types: the physical or hardware address of a network interface card; for ethernet this 48-bit address might be 0260.8C00.7666. The hardware address is used to forward packets within a physical network. Fortunately, network users do not have to be concerned about hardware addresses since they are automatically handled by the networking software.

The logical or Internet address is used to facilitate moving data between physical networks. The 32-bit Internet address is made up of a network number, a subnetwork number, and a host number. Each host computer on the Internet, has a unique address. For example, all Internet addresses at Colorado State have a network number of 129.82, a subnet number in the range of 1-254, and a host number in the range of 1-254. All Internet hosts have a numeric address and an English-style name. For example, the Internet address for UCC's CYBER 840 is 129.82.103.96; its Internet name is csugreen.UCC.ColoState.EDU.

address resolution
Conversion of an Internet address to the corresponding physical address. On an ethernet, resolution requires broadcasting on the local area network.

administrivia
Administrative tasks, most often related to the maintenance of mailing lists, digests, news gateways, etc.

anonymous FTP
Also known as "anon FTP"; a service provided to make files available to the general Internet community--Anonymous FTP.

ANSI
The American National Standards Institute disseminates basic standards like ASCII, and acts as the United States' delegate to the ISO. Standards can be ordered from ANSI by writing to the ANSI Sales Department, 1430 Broadway, New York, NY 10018, or by telephoning (212) 354-3300.

archie
A service which provides lookups for packages in a database of the offerings of countless of anonymous FTP sites. archie for a full description.

archive server
An email-based file transfer facility offered by some systems.

ARPA (Advanced Research Projects Agency)
Former name of DARPA, the government agency that funded ARPAnet and later the DARPA Internet.

ARPAnet
A pioneering long haul network funded by ARPA. It served as the basis for early networking research as well as a central backbone during the development of the Internet. The ARPAnet consisted of individual packet switching computers interconnected by leased lines. The ARPAnet no longer exists as a singular entity.

asynchronous
Transmission by individual bytes, not related to specific timing on the transmitting end.

backbone
A high-speed connection within a network that connects shorter, usually slower circuits. Also used in reference to a system that acts as a "hub" for activity (although those are becoming much less prevalent now than they were ten years ago).

bandwidth
The capacity of a medium to transmit a signal. More informally, the mythical "size" of The Net, and its ability to carry the files and messages of those that use it. Some view certain kinds of traffic (FTPing hundreds of graphics images, for example) as a "waste of bandwidth" and look down upon them.

bounce
The return of a piece of mail because of an error in its delivery.

btw
An abbreviation for "by the way."

client
The user of a network service; also used to describe a computer that relies upon another for some or all of its resources.

datagram
The basic unit of information passed across the Internet. It contains a source and destination address along with data. Large messages are broken down into a sequence of IP datagrams.

disassembling
Converting a binary program into human-readable machine language code.

DNS (Domain Name System)
The method used to convert Internet names to their corresponding Internet numbers.

domain
A part of the naming hierarchy. Syntactically, a domain name consists of a sequence of names or other words separated by dots.

dotted quad
A set of four numbers connected with periods that make up an Internet address; for example, 147.31.254.130.

email
The vernacular abbreviation for electronic mail.

email address
The UUCP or domain-based address that

a user is referred to with. For example, the author's address is brendan@cs.widener.edu.

ethernet
A 10-million bit per second networking scheme originally developed by Xerox Corporation. Ethernet is widely used for LANs because it can network a wide variety of computers, it is not proprietary, and components are widely available from many commercial sources.

FDDI (Fiber Distributed Data Interface)
An emerging standard for network technology based on fiber optics that has been established by ANSI. FDDI specifies a 100-million bit per second data rate. The access control mechanism uses token ring technology.

FQDN (Fully Qualified Domain Name)
The FQDN is the full site name of a system, rather than just its hostname. For example, the system lisa at Widener University has a FQDN of lisa.cs.widener.edu.

FTP (File Transfer Protocol)
The Internet standard high-level protocol for transferring files from one computer to another.

FYI
An abbreviation for the phrase "for your information." There is also a series of RFCs put out by the Network Information Center called FYIs; they address common questions of new users and many other useful things. RFCs for instructions on retrieving FYIs.

gateway
A special-purpose dedicated computer that attaches to two or more networks and routes packets from one network to the other. In particular, an Internet gateway routes IP datagrams among the networks it connects. Gateways route packets to other gateways until they can be delivered to the final destination directly across one physical network.

header
The portion of a packet, preceding the actual data, containing source and destination addresses and error-checking fields. Also part of a message or news article.

hostname
The name given to a machine. (See also FQDN.)

IMHO (In My Humble Opinion)
This usually accompanies a statement that may bring about personal offense or strong disagreement.

Internet
A concatenation of many individual TCP/IP campus, state, regional, and national networks (such as NSFnet, ARPAnet, and Milnet) into one single logical network all sharing a common addressing scheme.

Internet number
The dotted-quad address used to specify a certain system. The Internet number for the site cs.widener.edu is 147.31.254.130. A resolver is used to translate between hostnames and Internet addresses.

interoperate
The ability of multi-vendor computers to work together using a common set of protocols. With interoperability, PCs, Macs, Suns, Dec VAXen, CDC Cybers, etc, all work together allowing one host computer to communicate with and take advantage of the resources of another.

ISO (International Organization for Standardization)
Coordinator of the main networking standards that are put into use today.

kernel
The level of an operating system or networking system that contains the system-level commands or all of the functions hidden from the user. In a Unix system, the kernel is a program that contains the device drivers, the memory management routines, the scheduler, and system calls. This program is always running while the system is operating.

LAN (Local Area Network)
Any physical network technology that operates at high speed over short distances (up to a few thousand meters).

mail gateway
A machine that connects to two or more electronic mail systems (especially dissimilar mail systems on two different networks) and transfers mail messages among them.

mailing list
A possibly moderated discussion group, distributed via email from a central computer maintaining the list of people involved in the discussion.

mail path
A series of machine names used to direct electronic mail from one user to another.

medium
The material used to support the transmission of data. This can be copper wire, coaxial cable, optical fiber, or electromagnetic wave (as in microwave).

multiplex
The division of a single transmission medium into multiple logical channels supporting many simultaneous sessions. For example, one network may have simultaneous FTP, telnet, rlogin, and SMTP connections, all going at the same time.

network
A group of machines connected together so they can transmit information to one another. There are two kinds of networks: local networks and remote networks.

NFS (Network File System)
A method developed by Sun Microsystems to allow computers to share files across a network in a way that makes them appear as if they're "local" to the system.

NIC
The Network Information Center.

node
A computer that is attached to a network; also called a host.

NSFnet
The national backbone network, funded by the National Science Foundation and operated by the Merit Corporation, used to interconnect regional (mid-level) net-

works such as WestNet to one another.

packet
The unit of data sent across a packet switching network. The term is used loosely. While some Internet literature uses it to refer specifically to data sent across a physical network, other literature views the Internet as a packet switching network and describes IP datagrams as packets.

polling
Connecting to another system to check for things like mail or news.

postmaster
The person responsible for taking care of mail problems, answering queries about users, and other related work at a site.

protocols
A formal description of message formats and the rules two computers must follow to exchange those messages. Protocols can describe low-level details of machine-to-machine interfaces (e.g., the order in which bits and bytes are sent across a wire) or high-level exchanges between allocation programs (e.g., the way in which two programs transfer a file across the Internet).

recursion
The facility of a programming language to be able to call functions from within themselves.

resolve
Translate an Internet name into its equivalent IP address or other DNS information.

RFD (Request For Discussion)
Usually a two- to three-week period in which the particulars of newsgroup creation are battled out.

route
The path that network traffic takes from its source to its destination.

router
A dedicated computer (or other device) that sends packets from one place to another, paying attention to the current state of the network.

RTFM (Read The Fantastic Manual).
This anacronym is often used when someone asks a simple or common question. The word 'Fantastic' is usually replaced with one much more vulgar.

SMTP (Simple Mail Transfer Protocol)
The Internet standard protocol for transferring electronic mail messages from one computer to another. SMTP specifies how two mail systems interact and the format of control messages they exchange to transfer mail.

server
A computer that shares its resources, such as printers and files, with other computers on the network. An example of this is a Network File System (NFS) server which shares its disk space with other computers.

signal-to-noise ratio
When used in reference to Usenet activity, signal-to-noise ratio describes the relation between amount of actual information in a discussion, compared to their quantity. More often than not, there's substantial activity in a newsgroup, but a very small number of those articles actually contain anything useful.

signature
The small, usually four-line message at the bottom of a piece of email or a Usenet article. In Unix, it's added by creating a file a no-no.

summarize
To encapsulate a number of responses into one coherent, usable message. Often done on controlled mailing lists or active newsgroups, to help reduce bandwidth.

synchronous
Data communications in which transmissions are sent at a fixed rate,
with the sending and receiving devices synchronized.

TCP/IP (Transmission Control Protocol/Internet Protocol)
A set of protocols, resulting from ARPA efforts, used by the Internet to support services such as remote login (telnet), file transfer (FTP) and mail (SMTP).

telnet
The Internet standard protocol for remote terminal connection service. Telnet allows a user at one site to interact with a remote timesharing system at another site as if the user's terminal were connected directly to the remote computer.

terminal server
A small, specialized, networked computer that connects many terminals to a LAN through one network connection. Any user on the network can then connect to various network hosts.

TeX
A free typesetting system by Donald Knuth.

twisted pair
Cable made up of a pair of insulated copper wires wrapped around each other to cancel the effects of electrical noise.

UUCP (Unix to Unix Copy Program)
A store-and-forward system, primarily for Unix systems but currently supported on other platforms (e.g. VMS and personal computers).

WAN (Wide-Area Network)
A network spanning hundreds or thousands of miles.

workstation
A networked personal computing device with more power than a standard IBM PC or Macintosh. Typically, a workstation has an operating system such as unix that is capable of running several tasks at the same time. It has several megabytes of memory and a large, high-resolution display. Examples are Sun workstations and Digital DECstations.

worm
A computer program which replicates itself. The Internet worm
(The Internet Worm) was perhaps the most famous; it
successfully (and accidentally) duplicated itself on systems across
the Internet.

wrt
With respect to.

Regular Feature

ZCPR Support

Live from Israel

The Z-System Corner

By Jay Sage

This column is coming to you from Israel by the miracle of electronic communication. Bill Kibler, it seems, is catching up on the publication schedule for *TCJ*. This is good news for readers but not for me; I have come to rely on the slippage. Just before I left for Israel, I learned from Bill that issue 62 was essentially ready to go to press, but I had been counting on writing my column after I got back. You can easily understand that things get pretty hectic when planning an international trip, especially when it involves making long-distance arrangements for a Bat Mitzva celebration.

It looked as though I would miss this issue, but, when Bill heard that I would have electronic mail contact while I was away, he encouraged me to write something short and send it along as an email message. In particular, he suggested that in view of many recent changes I review the current Sage Microsystems East (SME) product line.

A New Sales Approach

At the Z-Fest we held over the weekend of the Trenton Computer Festival I announced major price reductions. Changes in the economy in general and in government funding for scientific research in particular have tremendously increased the demands of my job at MIT. Critical circuit testing during the weeks of my vacation requires daily contact with my colleagues. That's why I arranged for an email account here in Israel. With such work pressures I could no longer find the time to produce custom orders in custom disk formats. So I decided to make two changes.

On the one hand, I decided to limit the disk formats supported to just two: Kaypro DSDD and IBM 360K. The former was the most popular true CP/M format, and many people can deal with the PC format in one way or another. In the worst case, there are several people (David McGlone and Elliam Associates, to name just two) who will convert disks for a modest fee. On the other hand, the prices have been reduced to the lowest level at which I can justify the time spent to handle orders at all.

This was, in part, in response to a sudden and dramatic decrease in sales starting at the beginning of this year. It looks as though we may finally be coming to the beginning of the end of 8-bit computing as an area that can support active new developments and commercial products. So one could regard the current pricing as a kind of close-out sale. At the new prices, you can buy an item just for the fun of playing with, even if you have no real, long-term use for the program. Think of it as you would going out to the movies or to dinner!

Operating System Extensions

The flagship products of the SME line are NZCOM and Z3PLUS. These are versions of the Z-System that install themselves automatically on top of the existing CP/M operating system. NZCOM is for computers currently running CP/M version 2.2, while Z3PLUS is for computers running CP/M version 3, also known as CP/M-Plus. Each of these products is now only $20, down from the previous $49 and original $70.

There is no room here to describe what Z-System is in any detail; that has been the subject of most of my columns for the last six years. Suffice it to say that Z-System is a highly advanced operating system that, while almost totally compatiple with CP/M, has many features more commonly found in mainframe and minicomputer operating systems. Some of its features are unique and better than anything found in any other operating system I know of. The basic goal of Z-System is to give users more freedom: freedom to add new capabilities, to automate and simplify tasks, and to perform tasks in alternative ways that suit individual tastes.

Central to the power of the NZCOM implementation of Z-System is the ZCPR version 3.4 command processor. This is the part of the operating system that provides the direct interface between the computer operator and the computer (the other parts of the operating system serve programmers). The source code for ZCPR34 is not needed, as NZCOM already comes with several versions in binary form that that NZCOM uses. Some people, however, like to make custom versions or modifications, others just like to have source code on principle, and still others want it so they can learn how the command processor works. For these people, the source code is available as a separate product. Those who have already purchased NZCOM from SME can get it for $10; for others the cost is $15.

There is also a replacement for the BDOS (Basic Disk Operating System) part of the operating system. I call this product ZDOS (Z-System DOS). It actually includes two slightly different DOS replacement modules, ZSDOS and ZDDOS. You can use whichever one you like and can even switch between them.

Both support file time-and-date stamping. ZDDOS does this using entirely internal code; ZSDOS contains the basic datestamping code but requires a small external module with the clock interface code. Both include the ability to locate and use files that are not in the directory area specified. The ZCPR command processor does this for locating command (COM) files, but this cannot help when progams use auxiliary files (e.g., WordStar with its OVR file or spell checking programs with their dictionaries). ZSDOS uses the space freed up by moving the clock code out to an external module to implement more varied and extended methods of file searching.

Those are only the two most dramatic features of ZDOS. Here is a list of other features that apply to one or both versions: automatic disk logging when diskettes are changed (no more control-C required); fast logging of fixed (hard) disks (done only once); improved error handling with plain-English error messages and reporting of the name of the file involved, if any; support for the archive bit for tracking modified files; enhanced write protection; wheel security protection; and larger files (32 MB) and disks (2 GB). ZDOS is an excellent piece of work that I highly recommend. Its price used to be $75; now it is only $30.

As a brief aside, I would like to mention that the authors of ZDOS have come out with a follow-on product that supports banked memory. It is called B/P-BIOS (Banked/Portable BIOS) and ZSDOS2. SME will not be carrying it; if you are interested, please contact Hal Bower, the principal author, directly at 7914 Redglobe Court, Severn, MD 21144, 410-551-5922.

The most advanced and spectacular operating system extension offered is BackGrounder-ii, or BGii for short. BGii adds multitasking capability to a CP/M-2.2 or NZCOM system. It allows several programs to be run independently. Two of these programs can be any CP/M tasks (for example, a text editor and an assembler or compiler); a third program can be chosen from more than a dozen "background" commands internal to BGii. The tasks cannot all run actively at the same time. Only one performs active computations; the others are suspended in their current state, ready to be restored to active status at the user's command. BGii is now only $20.

One note of caution. BGii uses a "swap" file on disk to save the complete machine state for the main task that has been swapped out. The time required to switch tasks is determined by the time that about 100K of data can be exchanged between this file and memory. A RAM disk provides superb performance (my SB180 switches tasks in about one second). A hard disk is generally adequate, since BGii is designed to optimize access to the swap file. BGii is not recommended for computers that have only floppy disk drives, though it can be used provided a system diskette with the swap file can be kept in drive A at all times. BGii's internal commands require swapping only 4K of data, and this can be accomplished quite quickly even with a floppy disk. A full task swap may take several tens of seconds, but this is still faster than terminating one task and starting up another every time one wants to alternate between programs.

The final operating system extension is DosDisk, which implements a virtual MS-DOS file system, allowing one to use MS-DOS 360K diskettes directly on a CP/M computer. DosDisk, unlike any other CP/M interface to MS-DOS-format diskettes, has full support for DOS subdirectories. It is also unique among foreign-format support programs -- under either CP/M or MS-DOS -- in that it maintains time and date stamps when used together with ZDOS. DosDisk allows you to make full use of a DOS DSDD diskette to carry data back and forth between, say, a DOS computer at work and a CP/M computer at home. DosDisk is now only $15.

DosDisk, like BGii, requires a caution. Because it depends on facilities in the host computer's BIOS (Basic Input/Output System) code, it cannot be used on all computers. Here are the computers for which custom versions are available: Kaypros equipped with TurboROM or KayPlus ROMs, with CP/M or QP/M; Xerox 820-I equipped with a Plus-2 ROM and QP/M; Ampro Little Board; SB180 and SB180FX with XBIOS; Morrow MD3 and MD11; Oneac On!; and Commodore C129 with a 1571 drive. There is a kit version for brave souls who are ready to write their own interface drivers (and possibly modify their BIOS).

Other Products

Well, this "short" column is already not so short, so I will have to mention the remaining products with even less complete descriptions. The ZMATE text editor has been described in a number of my recent columns. The new version has been completed, and the price has been reduced to $20. Gene Pizzetta's revision of the manual still needs a little editing. I hope it will be available in a few months.

Another spectacular product is DSD, the Dynamic Screen Debugger. This is a full-screen debugger and Z80 simulator. It is a fabulous tool for debugging programs under development or figuring out how programs work. Price is now $25.

Al Hawley announced at the Trenton Z-Fest that he would reduce the price for his excellent assembler/linker package, ZMAC. It used to be $70 with a printed manual and $50 with the manual as a disk file. Now it is sold only with the printed manual and for $45.

The BDS C compiler package, with both Z-System and standard versions, is now only $30. JetFind, a text-search utility with support for grep (generalized regular expression parser) text specification and support for crunched files and files in libraries, has been reduced to $10. XBIOS, a banked BIOS for SB180 and SB180FX computers, is still available (though no longer supported) at $30. My ZCPR33 User Guide is still in print; the price has been dropped to $10. It still has useful information on the design philosophy behind ZCPR 3.3 and 3.4.

A number of other items are still available at their old prices. These include

Continued on page 17

Regular Feature

Intermediate

Letters and S-100 BUS

Dr. S-100

By Herb R. Johnson

This month, we have a brief **tutorial on buses** for the hardware novice. From the mailbag: more on hard disk controllers, and stories of S-100 systems from around the world! But first, some personal events....

Moving, Mail and Messages

My wife and I bought a house recently and moved in at the end of May. Of course, we are still unpacking at the end of June. In fact, I pulled my back last weekend whilst scrambling through my S-100 system collection in the garage. My pain is your gain, since I now have some time to write my column! You might also take advantage of my grief at moving hundreds of pounds of systems by relieving me of a few of them!

Please note that I am still using the commercial PO box, namely **CN 5256 #105, Princeton NJ 08543,** but my new phone number is **(609) 771-1503**. You can also contact me over the **FidoNet CPMTECH** conference as "Herb Johnson."

Tutorial Topic: What is a Bus?

Regular readers of this column are familiar with the S-100 bus, either as past or as present owners of systems that use this 100-pin interconnection "standard." But a lot of readers may own systems that do not offer a "bus" to plug new devices into, or may not know what a bus provides to its users.

I particularly want to provide those readers who are unfamiliar with computer bus architecture some idea of what is going on when we talk about buses using terminology such as "timing problems," "what is its address," "wait states," and so on. Like most jargon, it makes sense once you see the principles.

For an extensive review of the S-100 bus, I refer readers to my articles in *The Z-Letter* issues #11 and #13, "What is the S-100 bus to me?" which describes the S-100 bus in more detail. Most buses are based on a particular processor's timing signals. A convenient reference for typical processor timing signals is any Z-80 hardware manual, or a manual for almost any processor.

Why have a bus?

From the beginning, users of computer systems wanted to plug devices into their computers that were not anticipated by the computer's designers. There are two ways to provide connections between a device and a computer. The designer can provide a standard single-device interface, such as a serial or parallel interface, which are common standards across several computer manufacturers but are limited in speed or response. Or, they can provide a general interface or "bus" that is faster, with multiple channels or "addresses" to selectively access each device, but with a common data path and control signals for all devices to monitor.

Many designers have chosen to provide buses to facilitate their product's expansion. By contrast, the designers of the early Macintosh computers wanted to keep users OUT of their box, so Apple chose NOT to use a bus! The only other reason not to provide a bus is to save money (a bus requires extra chips, and connectors, and circuit board space), especially on specialized systems which are typically "upgraded" by complete replacement.

Almost all computer buses feature signals for address, data and control. Address lines are a set of wires with signals that at a certain time contain a unique pattern which is recognized by a specific device. **Data lines** carry another set of signals that contain a pattern of information to be sent to or received from a specific device, which is referred to or "addressed" by the address lines. **Control lines** carry yet another set of signals that provide timing information about the presence of address and data signals, so that a device knows "when" to look at the address and data lines and "what" to do with them at that time.

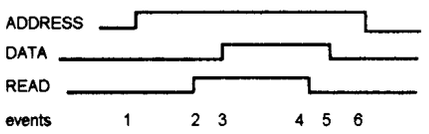
Reads and writes

If your CPU or processor intends to "read" data from a device, it must first set up the address lines with the address, then tell the "addressed" device to put its data on the data lines to be "read." The processor must read the data lines while the device holds the data lines active, then tell the device to release the data lines. Finally the processor releases the address lines. The control lines are used by the processor to communicate these events to the device.

Again, the address lines are the first and last to be active, with the data lines active in between and the control lines marking the times at which these events occur. We show these events with "timing charts," which represent some or all of each group of signals as a bar graph, with the x-axis (left to right) representing **time** and the y-axis (bottom to top)

representing that a timing event is **active** (top) or **inactive** (bottom).

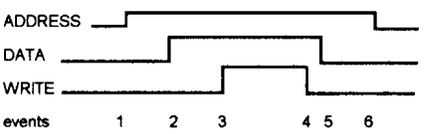
The following is a timing chart for a generic read operation on a bus:



The sequence of events is very important! First, the address lines must have the correct address information on them (event 1) and the corresponding signals must take time to adjust or "settle," as the voltage levels on the lines cannot change immediately; plus the circuits of the device must process these signals. By event 2, the device has had time to become ready and the processor can send out the READ signal. The device can detect the **leading** (start), **rising edge** of the signal and read the address lines to know that IT is the selected device and it is selected to be read. By event 3, the device has processed the READ signals and put its data out on the data lines. The processor waits until event 4 to read the data lines by using the **falling, trailing** (or end) **edge** of the READ signal. The device holds the data lines **active** (holds the correct data) until event 5, giving the processor just enough time to complete the read. In addition, the processor holds the address lines active a little longer until event 6.

(For the S-100 bus, there are several signals to mark these events. **MREQ**, a memory request control signal, occurs from before event 2 to after event 5 to define the entire memory cycle. **PSYNC** is a timing signal that is active briefly to mark event 2, the start of the read command. **PDBIN** is the timing signal corresponding to the "read" signal described. The status signal **SMEMR** for "memory read" is not described, but is timed similar to **MREQ** and indicates the processor status. The S-100 input data lines are **DI0** through **DI7**, and the address lines are **A0** through **A15**.)

The process of writing data is similar, but now it is the processor that puts data on the DATA lines. Consequently, the data lines are active early, before the WRITE signal, so that the device has time to read the data:



Typically the processor puts both address and data on the bus at event 1 and data at event 2, makes the WRITE signal active at event 3, holds address and data "steady" for the write at event 4, and releases both data and address lines at events 5 and 6 respectively.

(Again for the S-100 bus, there are several signals that mark these events. The **MREQ** control signal occurs from before event 2 to after event 5 as before. **PSYNC** is also active briefly to mark event 2. **PWR** or **MWRITE** is the timing signal corresponding to the "write" signal described. The status signal **SWO** is longer in time than **PWR**, occurring from event 2 to event 5. The S-100 output data lines are **DO0** through **DO7**; the address lines are **A0** through **A15**.)

A brief on other bus signals

How does the bus keep these "events" in sequence? First, the processor provides most of the control signals that represent these events. Secondly, there is also a "universal timing source" or **CLOCK** signal on the bus that all devices can use to reference the timing of events.

Not all reads and writes are equal. Some devices are strictly **memory** to hold programs, some are **I/O** (input and output) devices. In addition, some events require delays or **wait states** from the processor, so the read and write signals are "stretched" or extended longer in time. Also, some processors provide informational signals or **status** signals to describe the processor's actions. Not all buses have the same set of signals. For example, buses favored by designers of computers using the Motorola 68000 series processor have no "I/O" status or control lines, as all devices are addressed as locations in memory.

A bus also provides a convenient source of **power,** so there are several lines devoted to DC power. An interesting note about the S-100 bus is that these DC power lines are **unregulated** and may drift around a volt or two. Voltage regulators on the card turn the +7 to +9 volts (nominally +8 volts) on the bus into +5 volts for logic power; the same occurs for plus and minus 18 volts, to +/- 12 volts.

Finally, for special events that occur at the device side of the bus instead of the processor side, there are lines for **interrupt** signals which require the processor to perform functions controlled by software "interrupt handlers."

Particulars for the S-100 and IEEE-696 bus

The original S-100 bus, shown in past issues of *TCJ* in its centerfold of the IMSAI CPU and front panel, is distinguished by **8 data read** and **8 data write** lines; **16 address lines** (2**16 or 64K of address space); several **control** lines for read, write, interrupt acknowledge; and several **status lines** for I/O, stack, memory, wait states, and so on.

On the more advanced IEEE-696 bus, the data lines DO0-DO7 and DI0-DI7 can be temporarily combined into one 16-bit bidirectional bus, if the addressed device can use them. With the control line **SIXTEEN**, the device can tell the processor that a read or write can be performed on all sixteen data lines. In addition, the IEEE-696 expands the number of address lines to 24.

Letters

I've accumulated many letters over the last months, so I'll try to catch up here. Please contact these people if you can help them out, or to at least offer them moral support. I'd appreciate a copy of the correspondence for future reference.

Rolf K. Taylor has a few Osborne I parts and a Dynabyte S-100 system with

some Morrow/Thinker Toys and Dynabyte cards on an "as-is" basis.

Robin R. Whitten of Santa Ana CA is looking for some 8" IMDOS disks for his IMSAI. I think I have a few..

Michael Griffin of Tillsonburg, Ontario in Canada has a Compupro 8/16 with a harddrive but without docs, and is looking for help and information.

Chris Christensen of San Bruno, CA has a Compupro CP/M-80 system as part of his "large collection." He wants to implement keyboard buffering in his BIOS:

"There is a file included with Compupro's CP/M version 'N' called HMX210.com that, I believe is supposed to do this, but when I run it I get the error message 'mismatch between base of CBIOS code area and assembled load module base address'. I'm not an expert at modifying the operating system so I'm not sure what this means or how to go about solving the problem. I'm also wondering what was the last version of CP/M-80 released by Compupro...I have heard of versions up to 'T'. There is no one at CompuPro any more that has any information about CP/M-80.

"Also, to leave the 8-bit world for a moment, I recall that Dynacomp once sold a version of UNIX for the CompuPro with the CPU 68K board. I heard that there were efforts to make this available again. Do you know anything about it? I would like to hear from other people who still use CompuPros, especially anyone who has upgraded some of the older single-user systems."

Todd Silk of Hayden ID, a regular correspondent, is in the process of restoring an Altair he just purchased: "The Altair showed up but it was repainted dark brown. Do you have a picture or drawing of what it is supposed to look like? It's a turnkey model, with what looks like IBM blue and tan underneath the paint. The front must have been replaced. The cards in it include a FDC-1 and a 64K memory card by Teletek, and it seems to be a much better machine than the SD Systems based system that I have. "I really enjoy your articles, I hope you can keep it up!"

Roger L. Foco of Hightstown NJ, was kind enough to send me some Forth book's along with a request to spread the word. He has an IMS S-100 system (Z-80) for sale, including 8-inch drives, docs and CP/M software. Call him or me for a list and details.

Thanks to all who write or call me! Between my move, my (unggh!) back and my day job, it's tough to write back and respond promptly, but be patient, I'll get to ya!

Hard Drive Interests

A few columns back I asked if anyone was interested in a new hard disk controller for the S-100 bus. Among a few positive responses, Paul Herman of the *Z-100 Lifeline* reminded us that he sells a SCSI controller for that (IEEE-696) system for $210, plus $60 for the software. Here is some follow-up correspondence:

Alberto Girlando, a professor at a university in **Parma, Italy**, is looking for boards and disk drives for his NorthStar Horizon system. He'd like an IDE controller too, but he hasn't "the slightest idea of what a right price might be!"

Rick Rodman of Manassas VA (of TCJ's Real Computing) writes:

"A SCSI host adapter for $210 is way too high! It can be done with a single NCR [the manufacturer's name - Herb] chip and some miscellaneous decoding. In fact, it may even be possible to connect the SCSI bus to an 8255 PIO [Intel parallel I/O chip] on some boards. I tried to do this about ten years ago when it was SASI and ran into some timing problems, but SCSI has looser time constraints than SASI.

"I have some ACT hard disk controllers, and I think some of the host adapters [which interface the controllers to the host computer]. These controllers are all LSTTL [low power fast TTL], very big and power-hungry, but they are documented and easy to connect. I used to do software consulting connecting these to various machines where we didn't have BIOS source, such as the Osborne, Zorba and Otrona. They have 8 read-write registers that require 3 address lines to access, so if I could get 12 or so bidirectional TTL bits, I was there. The software used a tiny BIOS called OBIOS, which loaded itself below the original BIOS."

Rich would be interested in selling and helping others with these controllers, and he also has a number of S-100 boards for sale. "I've currently got three S-100 systems running. First, my BBS which is an all-IMS [Industrial Micro Systems] with a 21MB SMD hard drive I turn on sometimes and dual floppies, running CP/M, CP/M-Plus, or MP/M as I desire. Second, another IMS system in an Integrand box. Third, a Compupro CPU-32016 with 1MB of RAM (tada!) using a Konan David Jr. [hard drive controller?] and a Cromemco 16FDC, running Bare Metal. I think I bought the 16FDC from you a few years back...

"Oh, another thought: Ten megabyte 8" Bernoulli are showing up in big stacks at hamfests at giveaway prices. I have a PC interface board for one of these, and it doesn't have any LSI [big chips] on it at all - just 21 LSTTL chips, two resistor packs and a DIP switch. Maybe folks could use those for their hard drives? Documentation would be a problem."

I seem to recall an article on interfacing 10MB Bernoulli drives that was not too difficult to implement: maybe it was in *TCJ* or *Micro Cornucopia*. Anybody know? By the way, Rick, you are the first person I know with a working and complete 32016 system! For the rest of us, this refers to a National Semiconductor 32-bit processor that was a hot topic a few years ago. In fact, I cannot think of any major processor that was not available for the S-100 bus!

References

Copies of *The Z-Letter* S-100 articles in issues 11 and 13 can be obtained from **Lambda Software Publishing,** 149 West Hillard Lane, Eugene OR 97404-3057. Call (503) 688-3563 for details.

The Z-Letter supports all Z-80 compatible systems and is a great source for current (!) developments in Z-80 based systems and software.

FidoNet is an international network of bulletin board systems (BBS) that carry many message areas or "echos" that you can participate in, including the CPMTECH echo. Contact a local BBS to get a list of systems or to ask the system operator (SYSOP).

Rolf K. Taylor, RFD#1 Keeler Land, N. Salem NY 10560-9705 (914) 669-5421.

Prof. Alberto Girlando, 1st Chimica Fisica, Universita Degli Studi Di Parma, 43100 PARMA, Italy.

Robin R. Whitten, 909 E Camile St, Santa Ana CA 92701 (714) 550-1005.

Michael Griffin, Apt 207, 182 Lisgar Ave, Tillsonburg Ontario Canada N4G 4L2.

Charles Christensen, 2780 Concord Way, San Bruno, CA 94066.

Rick Rodman, 8329 Ivy Glen Court, Manassas VA 22110.

Todd Silk, 10721 Oak, Hayden ID 83835.

Roger L. Foco Sr, CCP, 202 Maxwell Ave., HIghtstown NJ 08520 (609) 448-6826.

Z-System Continued

the SLR assembly-language tools, the MEX telecommunication programs that I covered extensively in TCJ columns some time ago, and several Z-System IOPs (Input/Output Packages).

Closing Comments

In closing, I would like to put in a plug for 4DOS. This is a superb command processor replacement for MS-DOS computers. If, like many 8-bit hobbyists, you also use MS-DOS computers, then you should replace Microsoft's highly mediocre COMMAND.COM with 4DOS.COM. If you are used to Z-System on your 8-bit computer, you will finally feel that the DOS straight-jacket has been removed. 4DOS provides what DOS should have offered from the beginning. SME sells 4DOS at the slightly discounted price of $65.

For the next issue of TCJ I hope to present the material I originally planned for this issue on how to implement highly sophisticated automation using Z-System or 4DOS and ZMATE or PMATE. This is a technique I developed last summer to allow my 486 computer at work to carry out a complex series of electronic circuit simulations while I was away on vacation for a month. The work would even resume automatically after a power failure!

CLASSIFIED, FOR SALE and WANTED

Amstrad (c) PCW SIG: $9 for 6 bimonthly newsletters dealing with the most popular CP/M machines still in production. Learn where to buy 3" discs, how to add 3.5 and 5.25 drives and where to buy the 8 MHz Sprinter board with room for 4 Meg of RAM. Make checks out to Al Warsh, 2751 Reche Cyn Rd #93, Colton, CA 92324.

For Sale: **GIMIX** 6809 SS-50 floppy disk controllers. Have six to sell at $25 each plus shipping ($5). Bill at TCJ (800) 424-8825.

FOR SALE: ***Collector's Guide to Personal Computers and Pocket Calculators***. Prices and illustrations included. 336 pages. $14.95 plus $2.00 shipping. Fred Harfield, Box 52466, New Orleans, LA 70152. Digitial Cottage BBS (504) 897-6614, help, support, sales, of old systems.

NEEDED: **CRT for HP87**, mine's gone out. Jim Zikes, 808 1/2 Maint St., Quincy, IL 62301.

WANTED: DOCS/SOFTWARE for **MICROLOG Z80B XT** Board, have just the board. Bill at *TCJ*, 800-424-8825.

Willing to Help ZX81 users find items. Plenty for sale in England. Contact T. James, 12 Deacons Court Copmanthorpe, York Y02-3TR England. ZX-81's go for about $30/50US.

NOW AVAILABLE: **96TPI drives**, TEAC, Use in KAYPROS, $12.00 + Shipping, Mr. Kaypro, Chuck Stafford at (916) 483-0312 (eves).

Coming in September, CPM CDROM Available from Walnut Creek CDROM 1-800-786-9907.

The Computer Journal classified section is for items FOR SALE. The price is based on Nuts & Volts rates. If you currently have a Nuts & Volts ad just send us a copy of the invoice and we will print the ad for the same price.

Classified ads are on a pre-paid basis only. The rate is $.30 per word for subscribers, and $.60 per word for others. There is a minimum $4.50 charge per insertion.

Support wanted is a free service to our readers who need to find old or missing documentation or software. Please limit your requests to one type of system. Call *TCJ* at (800) 424-8825 or drop a card to *TCJ*, P.O. Box 535, Lincoln, CA 95648.

Guest Feature

All Readers

10th Year Special

REMINISCING and MUSINGS

By Frank Sergeant

ONE FORTH FITS ALL? or PLEASING ALL THE PEOPLE ALL THE TIME

Here is a bit of historical reminiscing which missed the deadline for the anniversary issue of *TCJ* (#60).

It all begin in 1989, I suppose. I was shipping v1.1 of Pygmy Forth for the IBM PC by August, 1989. Did I ever release a v1.0? I cannot remember. V1.1 was followed quickly by v1.2 in January, 1990. Then v1.3 appeared in September/October, 1990. About two years later (October, 1992) I released v1.4 with many improvements. It's been a long, hard four years.

I've had a lot of fun developing Pygmy Forth and offering it as a shareware system with no required minimum payment. The shareware disk includes the executable file, the full source code, and the manual. Thus it was a fully open system, not crippled in any way. At first, to encourage shareware "registration" I offered a Bonus Disk and a printed Glossary for $25. The disk contained shadow blocks commenting, block by block, on the source code. It also contained double (32-bit) and quad (64-bit) math routines, interrupt-driven serial routines, etc. Supplying the printed Glossary was the most tedious part of processing a bonus order. So, finally, as of version 1.4 I stopped offering the Glossary and offered the Bonus Disk alone for $15. The Glossary is not really needed. V1.4 has VIEW which pops you into the editor at the source code of the desired word, and ctrl-A switches between the source code block and the corresponding shadow block. The Bonus Disk still contains the math and serial port routines as before, but now contains copies of various Forth articles I have written. It even contains a 68HC11 assembler, written in Forth, that can run on a 68HC11 or on a PC. I am much happier to copy a disk and mail it inside a folded sheet of cardboard for $15 than to print a Glossary and ship it and the disk for $25. But, I have appreciated every one of those Bonus Disk requests. I no longer send the shareware disk for $5. If you want it from me, you must order the bonus disk for $15 (which includes the latest version of Pygmy). Alternatively, the shareware version is available from GEnie, various shareware houses and BBSs, and via ftp. The Forth Interest Group makes it available for $20. At first I was concerned that this wasn't quite right: $15 to me for the bonus disk plus the shareware version, versus $20 to F.I.G. for just the shareware version. On the other hand, I see their point; I don't want to bother with mailing a disk for $5 or so.

Let's face it, Forth has a small (but enthusiastic) following. Within Forth, Pygmy has an even smaller (but enthusiastic) following. Monetarily speaking, I should have been developing a C library or digging ditches. This project was not a good choice from the stand point of making money. I suppose I have approximately broken even, as long as we do not count my time. I made it a point to send each new version to every person who had previously ordered either Pygmy or the Bonus Disk from me. For example, if someone sent me $5 for a copy of version 1.1, I not only sent him a disk with 1.1, but 3 more disks over the years as versions 1.2, 1.3, and 1.4 became ready. If the post office returned it with an address correction, well, I mailed it once more. I have also spent a fair amount time answering questions, making suggestions, tracking down hardware incompatibilities, and responding to suggestions. My goal has been to give superb service, and I think I've succeeded in this.

Has it been worth it? Yes, I think so -- not in money but in other ways. I've made some good friends, most of whom I've never seen. I enjoyed exchanging ideas about what a Forth should be. I've enjoyed the _satisfaction_ of feeling I have produced something that at least some people appreciate. I've heard from people from all over the world. It is such a pleasure to receive a comment such as "My favorite feature of PYGMY 1.3 is its simplicity. In most cases, even - I - can understand the inner workings." or "I just received your Pygmy FORTH v1.3 from the FORTH Interest Group and am quite delighted. I have been developing high performance instruments using the RTX-2000 ... Thus your FORTH felt comfortable immediately." or from Japan, "Pygmy is fine.. it makes me happy to touch." Many tell me how they've successfully customized it and send me examples of their code and suggestions for further improvements.

From time to time bulletin board postings (the Forth Roundtable on GEnie, etc.) have discussed whether public domain or shareware Forths have hurt "Forth." First of all, even though I am a Forth enthusiasist, I do not believe Forth has any feelings or is capable of being hurt in any way. I do not feel much compulsion to "Save Forth." I don't intend to present the arguments impartially here (you can read them yourself on GEnie). Rather, I intend to point out what I see as an interesting contradiction. It appears that it is argued both that shareware/PD Forths are so _bad_ they

hurt Forth and they are so _good_ they hurt Forth. (To be fair, I don't suppose the same person offers both arguments.) You see, if some poor perspective Forth user is exposed to a bad Forth implementation, he may give up Forth forever, attributing the trouble to the language instead of the implementation. On the other hand, if good shareware/PD Forths are available cheap, then he has no need to spend big bucks buying a system from a vendor. Thus the vendor fails to receive the money which would lead to greater advertising of Forth and the development of better Forth systems. So, what do you think, is Pygmy Forth hurting Forth? I certainly hope not. If so, that's too bad, because I'm not going to withdraw it.

I've made mistakes. Version 1.4 allows the stacks to be placed either in the main segment or in a separate segment. The latter allows very large stacks. But, large stacks are usually not needed in Forth. I shipped v1.4 with the stacks in a separate segment. This ordinarily doesn't hurt anything, and it is easy enough for the user to change, but I should have shipped it with stacks in the main segment.

Worse than that, I modified the video routines to use BIOS calls instead of direct memory writes. My goal was to eliminate problems for weird systems (AT&T 6300 for example). Work on previous versions was done on my trusty old XT. Unfortunately I used a '386 system while working on v1.4. I was delighted with how fast the BIOS routines were and figured any tiny slowdown (although unnoticeable to me) was worth the increased compatibility. Too late I finally ran v1.4 on an XT and was disappointed with the video speed. This version is still very usable though. And, the I/O routines are vectored, so it is easy to replace them with faster routines. So, a project for the near future is a version of EMIT to use direct writes. Soon, soon. Then you can choose BIOS for the greatest compatibility or direct writes for the greatest speed.

At first I had no particular constraints on how I created Pygmy. I could do it my way. Then a user base gradually came into existence. I do not want to disappoint them. They argue for a change and I feel compelled to listen. Often they are right and I _still_ refuse to do it their way. This convinces me I am not completely rational, but then again, who is? There is more to developing a Forth system than logic. There is also beauty. There is also fatigue.

Is Pygmy perfect and why do I keep harping on this? Because I would like it to be. But I do not think it needs to be. It may be that it is excellent for certain purposes and unsuitable for others. Such, I think, is the case with every language and every tool and every person. One of its strong points is its small size and understandability. The whole system can run from a single floppy. This goes against the modern trend of requiringmulti-megabytes of RAM and harddisk space. With it you can produce turnkey applications under your complete control, which you can understand. I think it takes a lot less faith to use a system you can master, and this is good.

At first I resisted adding textfiles for loading source code, but changed my mind for v1.4. An argument by Brad Rodriguez convinced me. What I heard was that it didn't matter whether _he_ wanted textfiles, his _clients_ wanted them! Oh, OK. That I can understand. So now you can load source code from either blocks or textfiles. Also, you can regen Pygmy to remove this feature if you don't want it. I'm glad I did this even though I still do all my own coding in blocks. But, now, I can write an application that FLOADs a configuration file the user creates with any old text editor -- no need to try to force him to use blocks and a block editor. So, I'm a convert to textfiles, sort of.

Some people suggest that Forthers spend too much time developing new Forth systems and not enough time actually doing something useful with them. Perhaps I am guilty of that. I do use it, though, for various utility applications for my own use, such as a PCB layout program and custom LaserJet font editor. Unfortunately these are "quick and dirty" and nowhere near being releasable products. Pygmy runs the PC portion of the EPROM programmer system I developed. Pygmy is at the heart of a bond and money market calculation program I wrote and maintain for a client. The shell is still in BASIC, a remnant from many years ago, but I am much happier working on the Forth part of the system. The Forth development goes much faster and is much more comfortable. I hope it will eventually replace the BASIC part completely.

MUSING ABOUT

I've had a number of things on my mind lately to talk to you about. I'll express some ideas about nearly everything from EPROM programmers to education, and hope to hear your ideas and suggestions.

Some Come Loudly

Yesterday I couldn't even spell Computer Science ...

I am tickled pink to announce that on May 15, 1993, Southwest Texas State University (SWT) conferred upon me the degree of Bachelor of Science in Computer Science, summa cum laude. I hope you will forgive me for saying it, but who else can I tell? It wasn't just summa cum laude, it was a 4.0 SWT GPA (grade point average) on 82 semester hours on a 4.0 scale. I am 46 years old, so my advisor says I've been on the "30-year plan." I started out in Accounting many years ago at the University of Texas, where, in my foolish youth, I often made far less than all As.

I have to have done this for fun, as I don't see how it is worth any money to me. I've worked in the computer field for many years now. Most of the time as a contract programmer, working on business applications in IBM (or Univac) mainframes in COBOL or assembly language. As the years have passed, more of my work has been done on micros (mostly IBM PCs) and I've used Forth whenever possible. I haven't worked at a "straight" job in many years and don't expect to start now. Starting back to school a few years ago has given me an interesting glimpse into the academic side of computing. First, I was surprised at how serious my fellow students take

their studies. I thought SWT was supposed to be a party school, but it sure isn't in the Computer Science (CS) Department. The professors are accessible and knowledgeable, but that doesn't mean they all teach well. The degree isn't called a BS for nothing. I suppose employers see the degree as proof there is no limit to the amount of BS you will put up with.

Still, I've had enough fun with it, and I enjoy the internet access enough, that I'm continuing on into graduate school at SWT. The CS department has asked me to teach four Intro to Computer Architecture labs and two C Programming labs next fall. There has to be some humor in me teaching C Programming labs! The Architecture labs bring me to my next subject.

Programmable Logic Devices (PLDs)

I agree with Bill that *TCJ* construction articles shouldn't rely on PLDs (PALs, GALs, etc.) exclusively. They should at least show the logic in standard gates, even if PLD details are also provided. The SWT lab uses regular TTL and LS TTL (e.g. 7400 or 74LS00 quad 2-input NAND gates). They stand up well even under fairly rough handling. I think I'll suggest students buy their own personal plastic breadboards and their own chips, do the assignments at home, and bring the results into the lab for discussion, examination, and troubleshooting. I think most of the time spent _wiring_ the chips together is wasted time as far as learning goes. The advantage of using SSI (small scale integration chips such as the AND, OR, and NAND gates) and MSI (counters, multiplexers) is you get to see their interconnections. This helps in understanding what is going on, compared to merely entering a Boolean equation into a PLD compiler. I think all the logic buried in the PLD chip is too invisible and looks too much like magic. With the separate chips you can put a logic probe on each gate's inputs and output and see what is going on. For my own use my choice is 74HCxx. These are CMOS chips with the same pin-outs as the corresponding TTL chips.

In spite of that, I am interested in using PLDs to some extent in the lab. If anyone has any opinions on how to go about this, especially the cheapest way to go about it, please contact me with your suggestions. I may build my own PLD programmer, similar to the Bare Bones EPROM programmer I have designed.

Bare Bones EPROM Programmer

Bones, as I call it, fits on a 3" x 3" PCB or can be wire-wrapped or built on a plastic breadboard. It programs 2716, 2764, 27128, and 27256 type EPROMs. The circuit requires two ICs: a 4049 hex inverter and a 68HC11 micro. It connects to a MS-DOS PC via a serial cable. Software running on the PC controls Bones, presenting a menu of options and allowing copying to and from an EPROM and the PC's disk, etc. I sell plans and software on an MS-DOS diskette for $15 (in the US) and sell parts (excepts sockets, etc.) for $25. I bring this up in order to discuss Bill's topics of platform independence and serial interfaces. Since the hardware part of Bones has a serial interface, in theory it could be controlled from any host computer with a serial port. However, a lot of the convenience and comfort of the complete system is the software which currently requires MS-DOS. On the one hand, I don't see this as a major limitation, because a PC of some sort is available to nearly everyone, even if you borrow the use of one at a local college whenever you need to burn an EPROM. Even the slowest PC running DOS 2.1 will work fine as a host for Bones. On the other hand, I have excluded people without access to a PC. I think in the future I'll pay more attention to allowing a wider range of hosts to be selected -- perhaps on the PLD programmer. In with the plans and schematics, I include the artwork for the PCB in case you want to make one rather than wire-wrapping, but I do not sell a PCB for Bones, which brings me to my next subject.

Printed Circuit Boards (PCBs)

I made a few PCBs for Bones, but find I dread doing this. There is no way I can charge a price that would make me smile at the thought of making the board. I have tried 2 or 3 boards with the iron-on method. None came out worth a damn. I keep reading about this technique working. Maybe I'm just too sloppy or stupid to make it work. It sounds like such a great idea, just laser print or photocopy your artwork on to a plastic sheet (or onto the DynaArt paper), then transfer the toner to the board with a household iron, pull off the plastic backing (or disolve the DynaArt paper backing), leaving a beautiful pattern of toner on the copper blank, then etch the board. Well, for me _some_ of the pattern transfers, but never does _all_ of the pattern transfer! Are any of you doing this successfully? If so, tell me how you do it.

The only way that has really worked for me, so far, is to produce the artwork on white paper, using a laserjet printer, take the paper to a print shop and have them make a film negative (solid black everywhere except clear where I want copper to remain). Then I do a contact print of the film negative onto a presensitized (dry film) copper blank from Kepro using a #2 Photo Flood bulb. The blank is then "developed" by soaking and rubbing in an akaline bath, and then etched as usual. My boards usually come out OK, but not always. After etching, I drill the holes with a Dremel drill in a little drill press. This is mentally tedious. Do I have the hole lined up or am I about to ruin the board? All of this is so much trouble that I can't see myself in the board-making business at the moment.

I have some other ideas, though. I've bought some positive resist-covered boards, from Circuit Specialists in Arizona, but I haven't tried them out yet. This eliminate the step of having a film negative made. Maybe I could draw the positive artwork directly onto a plastic sheet in a laser printer and then do a contact print. It might even be possible

-- I'd like to test it sometime -- to print a positive onto white paper and do a contact print of the paper directly onto the board. The black lines of the toner will block more light than the paper will block. I have done this fairly successfully onto lithographic film, so it might just work. If any of you have any experience or ideas related to this subject, please write.

The thing I want most, though, is an XYZ table. Or, at least an XY table. I'm willing to provide the force to move the drill up and down, I just don't want to have to position it. So, the XY table is for drilling the boards, which would eliminate a lot of the tedium of making a board. If I build an XYZ table, I would try to use it not only to drill the board, but to route the board. That is, put some sort of grinding head into the drill, and, under computer control, cut away copper where I don't want it. This would eliminate the need for the photosensitive boards.

I really want to try this, but so far I've been too lazy or busy to build anything. I have some stepper motors. I'm not worried about the programming of the computer or about the electronics, it's the mechanics of building the table that I feel incompetent to handle. Various places offer high-precision threaded rod and anti-backlash nuts. I wonder, though, how well the cheapest set up would work. Take regular 1/4 inch, 20 threads per inch, $1/foot threaded rod from the local hardware store and attach it to a 200 steps per turn stepper motor. The nuts to go on the rod would cost about a nickel each. 200 steps would move the nut 1/20 of an inch (i.e. 0.05 inch), so 100 steps would move it 0.025 inch and one step would move it 0.00025. This means 4 steps move the nut 1/1000 of an inch.

Wouldn't this resolution be sufficient for prototype PCBs? I think it's worth a try. If the system worked fairly well, but was a little too sloppy, then extra money could purchase the higher precision rod and nut. If the only problem is backlash, I see two possible solutions. One is to use something like the old-fashioned screen door springs to put just a little tension on the nut in one direction to take up any slack. The second solution is done in software. Don't move left, cutting or drilling, and then down and then work back to right cutting or drilling. Instead, work to the left cutting or drilling, then home it all the way to the right, move it down, and work leftward once more. In other words, always work out from home position, don't work back in.

I understand _Circuit Cellar Ink_ is planning to publish some sort of XYZ table from one of their design contests, but I don't know when it will appear. Maybe that will spur me on to try it. What do you all think?

User Interfaces

I loved in #60 how Rick Rodman showed what a monstrosity Unix is. Still, it and its spawn of the devil, C, have a big following. At the university, I've had a chance to get much more familiar with them. I've had to use C and C++ extensively at SWT since C is the de facto official language these days in academia. In an operating systems course we delved fairly deeply into the source code for Minix. I was one of the very few who successfully completed the assignment to rewrite the Minix floppy disk driver to use whole cylinder buffering. A glutton for punishment, I am enrolled this summer in a Unix programming class. I want to criticize C and Unix, so I feel I need to learn a little about them; seems only fair. Bill, if you want to criticize MS-DOS, and I do think you have many legitimate reasons for doing so, I think working with Minix for a while might put MS-DOS in a better light. I don't say there isn't some fun in hacking around with it. Plus, Minix serves a useful purpose in exposing people to many of the ideas in current use in operating systems. If you are a mechanic and your customers drive Chevy Geos, well, good or bad, you need to know something about Geos.

Unix does have some concepts similar to Forth. The collection of executable files and scripts available from the Unix command prompt can be likened to the collection of words available in a Forth dictionary. The idea of small tools, each of which does one simple thing, and which can be linked together easily to do complex jobs, sounds similar to the Forth idea of a collection of small, simple tools (words), each designed for a specific task. Both systems offer the user a command line interface. These similarities make me a little uncomfortable.

Tell 'Em I Didn't Send Ya'

I'm not happy with Hard Drives International and their parent company Insight. I feel they cheated me. I don't plan to ever do business with them again. They credited me with the wrong amount on a disk controller I returned. I polited explained their error in a letter and got the kind of reply that P. J. Plauger says he files in his "The Customer is Always Right" file. P. J. says his file isn't very thick because there is seldom more than one letter in it from any given company.

File Transfer Protocol

Or, ftp for short. Later I'll mention getting Ghostscript via ftp. Ftp is a means of getting shareware or public domain files and programs from sites on the internet. For example, my Pygmy Forth is available from a site in Portugal via ftp (and maybe other places as well, by now). If you do not have ftp access you should go to a local university and ask around the computer terminal room. Get someone to show you how to ftp files. It's "free."

GEnie

Speaking of getting files, BBSs and services such as GEnie are another source. GEnie is changing their rate structure to (in the US) a flat $10/month for up to 4 hours a month, plus $3/hour over 4 per month. I always worry when they say they are changing their prices to help me. In this case, though, it looks like I'll save some money.

Drawing Schematics and Ghostscript

I've tried several approaches. The fastest is to sit down with a white piece of paper and a plastic template and just draw it out by hand. I have not yet found

a better way. Yes, I've heard there are schematic drawing programs available. Maybe the expensive ones work well, but I can't afford them. The cheap ones, shareware, I've tried were horrible. I could never "get into them" and feel oriented and comfortable. It has gotten to the point where I don't want to spend the time learning a new one to find it doesn't suit me.

I've done some schematics on a laserjet printer directly from Forth. Usually I define some large character cells, for resistors, AND-gates, etc as a laserjet softfont, which is downloaded to the printer. Then I code Forth words to draw lines and symbols. This has worked fairly well, but the scaling of the final drawing is inflexible and the layout work is not interactive, so it requires printing many partial results and adjusting the drawing until it finally looks right.

Don Lancaster has been touting the PostScript language for PCBs and schematics (and everything else). He runs the PostScript Round Table (PSRT) on GEnie. When I located a free PostScript clone named Ghostscript, I decided to give it a try, since PostScript has many more graphics primitives built into it than I have yet written for my Forth, and since it allows easy scaling and rotating of drawings.

Source code for Ghostscript is available, but I just got an executable file for a '386 PC via ftp. That, plus some samples from the PSRT, plus a careful study of several books written by the company that created and markets PostScript (Adobe), and a lot of hard work gave me a pretty good grip on the language. It is very similar to Forth, but not really interactive, at least the way I was using it. I would edit my program, get out of the editor, bring up Ghostscript, have it execute my program (i.e. produce a page or two on the printer), get out of Ghostscript, bring up the editor, etc.

Unfortunately, this was not much better than the way I'd been doing it in Forth. The extra graphics primitives were nice, but the feedback was considerably slower than I was used to with Forth. The drawing was still not interactive. I managed to produce some pretty schematics, but the process was so tedious that I've gone back to hand drawing them. Ghostscript accepts Postscript commands and programs and produces output for a PC's VGA display or for a non-Postscript laser or dot-matrix printer. The programs you produce can then be taken to a friend's or school or copy shop with a real PostScript printer for your final copy. It would be much, much better to be developing programs in PostScript if you had a real PostScript printer attached to your computer. Ghostscript gives you a cheap taste of it. I still plan to use PostScript (Ghostscript) for such things as graduation announcements, Xmas cards, "What part of NO don't you understand?" posters, and anything that requires easy access to a variety of fonts.

Throw Away MS-DOS

I've had the urge to run Forth on the bare metal. The entire system would be available in source code form, so nothing would be hidden or mysterious. One way would be to use BIOS calls for the interface to the hardware (OK, this isn't quite bare metal) and would be very portable to PC clones and semi-clones, but the use of the BIOS means you don't have full control over the code of your system and it means you can't run at the fastest speed. The other way is to use the BIOS only to boot the system from disk and then do everything else directly, no BIOS, no MS-DOS, no nothing! (This is how Minix does it. One of the uses of Minix is as source for details on how to access the PC hardware directly.) Would an approach like this meet Bill's objections to PCs?

I don't think that could be my only system, though. I feel I have to have a system fairly similar to that of my clients and potential clients. I think this means at least a '386 PC with MS-DOS (or PC-DOS or DR-DOS) and maybe running WINDOWS. I keep saying I'd throw away MS-DOS if I could, but I don't yet see how to do it, except by separating hobby from business.

Conclusion

Please send your solutions to my problems and your comments and suggestions to me at 809 W. San Antonio St., San Marcos, TX 78666, or via email to fs07675@academia.swt.edu on internet or to F.SERGEANT on GEnie.

Possibly trademarked names:
MS-DOS, PC-DOS, IBM, DR-DOS, PostScript, WINDOWS, Chevy, Geos.

A list of CPM User groups :

FROG Computer Society
Jim McCollum
321 Executive Office Building
Rochester, NY 14616-1701
(716) 244-4038
FROG Pond BBS:
(716) 461-1924

Vancouver Island Computer Operators Club
Dee Schoolingin
942 Cloverdale Avenue
Victoria, BC V8X 2T6
(604) 388-5464 x 107

Vancouver Portable Computer Club
Jay H. Siegel
4251 Lancelot Drive
Richmond, BC V7C 4S4
(604) 271-1519

Osborne/Kaypro User Club of Toronto
Leslie F. Fontaine
4 Munhill Road
Weston, Ontario M9P 1P9
416-247-8503

To include your name in this list, please send information to:
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535
or
B.Kibler@GEnis.com
GEnie = B.Kibler
CompuServe = 71563,2243
or
1-800-424-8825

Modem Script and Forth Tricks

By Walter J. Rottenkolber

Guest Feature

Introductory Forth

Quick Words

As part of my ever ongoing modem program, I looked over a commercial product. One of the major features was an extensive script language. It soon became clear that forth provided most of the functions of a script language with few exceptions, in fact just three basic words.

First, words to output command strings to the modem -- .M'', MTYPE, and MCR. Second, words to output command strings to the remote system -- REPLY'', and REPLY.

Third, words to wait for a key string from the remote system before continuing with command sequence -- WAITFOR'', and WAITFOR. Use these words very much like .'' string'' and '' string'' TYPE. Screen 2 provides some examples of use.

To output control codes, a carat (^) placed before a letter in the string converts it into the equivalent control code, eg. ^M or ^m outputs a carriage return. Case does not matter. I left out error checking so you must select proper character. This is only implemented in .M'' and REPLY''.

Standalone control words, such as MCR, can also be defined. In WAITFOR'', the variable TIMER2VAL must be adjusted to provide a one second basic delay. A value on the stack determines the number of seconds remote input is scanned before timeout. A flag (true= match before timeout) left on the stack allows you to modify the direction of the program.

```
\                    WJR18FEB93
Script Words for Forth Modem Programs
Walter J. Rottenkolber
Feb. 18, 1993

\s
MEMIT = primative direct modem output word.
T-OUT = terminal output word.
T-IN  = terminal input word.

a carat (^) before a letter in string converts it to the equivalent control
character.

VARIABLE &M^CHR?  ASCII ^ CONSTANT ^CHR
: M^CHR  ( -- )
&M^CHR?  @ IF UPC ASCII @ - MEMIT &M^CHR? OFF
ELSE  DROP  &M^CHR?  ON THEN  ;
: MTYPE  ( addr$ len -- )
&M^CHR? OFF BOUNDS ?DO I C@ DUP ^CHR =
&M^CHR? @ OR IF M^CHR ELSE MEMIT THEN 20 MS LOOP ;
: (.M")  ( -- )
R> COUNT 2DUP + >R MTYPE ;
: .M"  ( -- )
COMPILE  (.M") ," ; IMMEDIATE
: MCR ( -- ) CRR MEMIT  ;

\S   -- examples of use --
: ATTN  ( -- )  .M"  +++^M"  1200 MS  ;
: TONE  ( -- )  .M" ATDT,"  ;
: PULSE ( -- )  .M" ATDP,"  ;
DEFER T/PDIAL  ' TONE IS T/PDIAL
: DIAL#  ( a l -- )  T/PDIAL MTYPE MCR ;
: ZDIAL# ( a l -- )  \Dials then quiets modem for voice.
T/PDIAL MTYPE .M" C0^M" ;
: MINIT  ( -- )  \ Init. string for Zoom modem.
.M" ATM1 S0=0 S7=60 X1^M" ;
: DIAL  ( a l -- )  2DUP TYPE DIAL#  ;
\ : nerdbbs ( -- ) CR ." nerdbbs = " " 123-4567" dial ;

VARIABLE &R^CHR?  ASCII ^ CONSTANT  ^CHR
: RPY^CHR  ( -- )
&R^CHR?  @ IF UPC ASCII @ - T-OUT &R^CHR? OFF
ELSE  DROP  &R^CHR?  ON THEN  ;
: REPLY  ( addr$ len -- )
&R^CHR? OFF BOUNDS ?DO I C@ DUP ^CHR =
&R^CHR? @ OR IF RPY^CHR ELSE T-OUT THEN LOOP  ;
: (REPLY")  ( -- )
R>  COUNT 2DUP + >R REPLY  ;
: REPLY"  ( -- )
COMPILE  (REPLY") ," ; IMMEDIATE

VARIABLE  &TIMER     VARIABLE  &MATCH?
VARIABLE  &TIMER2    VARIABLE  TIMER2VAL
1600 TIMER2VAL !  \ for 5 Mhz Kaypro II
: RESTIMER2  ( -- )  TIMER2VAL @ &TIMER2 ! ;
: CNTDN  -1 &TIMER +! RESTIMER2  ;
: TIMEOUT?  ( -- f )  \ T= TIMEOUT
-1  &TIMER2 +! &TIMER2 @
0=  IF CNTDN THEN &TIMER @ 0=  ;
: TIMEMKEY ( -- f| c f )  \ t= timeout
BEGIN TIMEOUT? IF TRUE EXIT THEN MKEY? UNTIL MKEY
FALSE ;

: W-MATCH  ( addr$ len -- addr$ len )
2DUP  BOUNDS  DO
TIMEMKEY ?LEAVE UPC I C@ UPC = DUP
&MATCH? ! 0= ?LEAVE LOOP ;
: WAITFOR  ( addr$ len n -- f )  \ t= match within timelimit
&TIMER ! RESTIMER2 BEGIN &MATCH? OFF W-MATCH
&MATCH? @ &TIMER @ 0= OR UNTIL
2DROP  &MATCH? @ &TIMER @ AND  ;
: (WAITFOR") ( -- f )  R> COUNT 2DUP + >R ROT WAITFOR  ;
: WAITFOR" ( n -- f )  \ t= match witin n= #sec timelimit
COMPILE (WAITFOR") ," ; IMMEDIATE
```

-- Sum of Digits Cubed --

This small program solves the problem: Find those numbers greater than one whose digits cubed add up to the original number. (Hint: There are only four, all under 1000.)

Lest you consider this problem important, the British mathematician G. H. Hardy termed it "insignificant" since the solution contributes nothing to the advancement of mathematics.

Reference: Thomas P. Dance:"The Fortran Cookbook", 2nd Edition, TAB Books, 1984, p. 41.

```
\ Screen 32
: 5DROP  ( n n n n n -- ) 2DROP 2DROP DROP ;
: PRINT# ( n n n n n -- ) 5 0 DO  6 .R  LOOP CR ;
: CUBE ( n -- n3 ) DUP DUP * * ;
: HEDER ( -- )
CR ."       Sum    Cube of Digits" CR
."  Int. Cube  Hun  Ten  One" CR ;
: SUMCUBDIG ( n -- n n n n )
100 /MOD CUBE >R 10 /MOD
CUBE >R  CUBE R> R> 3DUP + + ;
: ?PRINT# ( n n n n n -- )
2DUP = IF PRINT# ELSE 5DROP THEN ;
: SUMCUBE ( -- )
HEDER  1000 1 DO I SUMCUBDIG
I ?PRINT#  LOOP ;
```

-- The Greatest Common Denominator --

This one screen program finds the Greatest Common Denominator, and so should be a boon to those students struggling with fractions. After a modulo operation on the original numbers (adjusted so the larger is the numerator), the modulo is repeated on the previous denominator and the current remainder until a remainder of zero is reached. The previous remainder is the answer. On randomly selected numbers, this generally is one, but the program will search out any valid GCDs with less pain and suffering than you will endure.

```
\ Screen 33
\ Greatest Common Denominator              WJR12APR93
\ Max values 65535

: U/MOD  ( u u -- ur uq ) 0 SWAP UM/MOD ;
: UMOD  ( u u -- ur )  U/MOD DROP ;
: GCD  ( u u -- u )
2DUP < IF SWAP THEN
BEGIN  TUCK UMOD DUP
WHILE REPEAT DROP ;
: .GCD  ( u u -- )
GCD ." = " U. ;
```

-- Perfect Numbers --

Divisors of a natural number are those integers that divide evenly into the number. Proper divisors are all those divisors except the number itself. When the proper divisors are summed, numbers can be divided into three classes -- deficient, perfect, and abundant -- depending on whether the sum is less than, equal to, or greater than the number. For example: 8 is deficient because its proper divisors, 1, 2, 4 = 7, which is less than 8.

Ancient numerology considered these distinctions important, and Perfect numbers were regarded as especially pleasing. When you examine the numbers some rules emerge. Prime numbers are always deficient because the proper divisor is always 1. Also deficient are powers of prime numbers, and proper divisors of a perfect or deficient number. On the other hand, proper multiples of a perfect or abundant number are abundant.

Lest you believe Numerology passe', consider all the searches for the number 666 in the name or title of your favorite enemy, so that you can apply the 'Mark of the Beast' to him. (It originally referred to Nero Caesar, if you've wondered).

The program takes numbers from 1 to 499, and determines the class to which it belongs. ?WAIT pauses the calculations every 20 lines so you can examine the number list and pick out the Perfect Numbers. A keypress continues the process.

If you plan to search more and larger numbers, it's best to rewrite the program to pickout only the perfect numbers or to save the results to a file. The calculations become much slower after 500, and stopping every 20 lines can be tedious.

Reference: Thomas P. Dance:"The Fortran Cookbook", 2nd Edition, TAB Books, 1984, p. 28.

```
\ Screen 34
\ Perfect Numbers                          WJR20APR93
VARIABLE  N#   VARIABLE  N/2   VARIABLE  SUM
: NXT# ( n -- ) N# ! 0 SUM ! ;
: FINDIVSUM ( -- )
0 BEGIN  1+  N# @  OVER
MOD 0= IF DUP SUM +! THEN
DUP  N/2 @  >= UNTIL  DROP ;
: WATIS# ( -- )
SUM @  N# @  2DUP
< IF NIP  . ." = Deficient" EXIT THEN
TUCK > IF . ." = Abundant"  EXIT THEN
. ." = Perfect *****" ;
: ?WAIT  ( i -- )  20 MOD 0= IF BEGIN KEY UNTIL THEN ;
: PERFNO  ( -- )
500 1 DO CR I DUP NXT# 2/ N/2 !
FINDIVSUM WATIS# I ?WAIT LOOP CR ;
```

-- THE END --

TCJ Center Fold

Special Feature

All Users

XEROX 820

8.1 General

The 820 family is a table top microcomputer composed of the following assemblies:

820 IP Processor

1. D.C. Power Supply
2. Processor (CPU) PWA.
3. CRT Assembly
4. Keyboard Assembly

820 IP – SA400 (5.25" Single Sided Floppy Drive)
820 IP – SA800 (8" Single Sided Floppy Drive)
820 IP – SA450 (5.25" Dual Sided Floppy Drive)
820 IP – SA850 (8" Dual Sided Floppy Drive)

820-II Processor

1. D.C. Power Supply
2. Processor (CPU) PWA
3. Floppy Disk Daughter PWA or,
   Fixed Disk Daughter PWA
4. CRT Assembly
5. Keyboard Assembly

820-II IP – SA400 (5.25" S.S. Dual Density Floppy Drive)
820-II IP – SA800 (8" S.S. Dual Density Floppy Drive)
820-II IP – SA450 (5.25" D.S. Dual Density Drive)
820-II IP – SA850 (8" D.S. Dual Density Floppy Drive)
820-II IP – SA1000 (10 MB Fixed Drive)

8.2 D. C. Power Supply

The D. C. Power Supply converts the AC supply input to three DC voltages required by the system. These voltages are +5, +12 and - 12VDC. Each voltage has short circuit protection by electronic current limiting. When any of the outputs are overloaded the entire Power Supply will shut down. The +5VDC is provided with overvoltage protection.

8.3 Processor (CPU) PWA

The processor (CPU) PWA provides the master control for the system. The Microprocessor is the central processing unit. It executes programs (software) that are stored in the 64K Ram and the 2K ROM (820), 6K ROM (820-II). The 820 IP incorporates a Z80 Microprocessor where as the 820-II IP incorporates a Z80A. Added features of the 820-II IP processor are:

A. 4 MHz Clock
B. 2-RS232 Ports (one dedicated to serial printer)
C. 820 System Bus Access
D. Audible Alarm
E. Video Highlighting
F. 6K ROM Expansion Capacity
G. 2-Fixed Disc Drive Options:
   SA606 and SA450
   SA1004 and SA850
H. Ethernet Connection (via 872/873 Comm Server)
I. 2-Buffered 8 Bit Parallel ports
J. Display Graphics

The CPU is supported by five intelligent periphial controllers. These devices handle the tasks of transferring the data to and from the periphial devices, thus relieving the burden on the CPU.

A. Disc Controller

This device (On the 820-II, it is located on the Daughter PWA for the floppy or the fixed) interprets commands from the CPU and generates appropriate control signals for the disc drives. It also interprets status signals from the disc drives and delivers them to the CPU upon request. The second function is to convert parallel data from the Data Buss to serial data suitable for recording on the disc and also the conversion from the serial data read from the disc to parallel data suitable to the CPU. The fixed drive assembly contains a 1403D Controller PWA that in effect tells the system what type of drives are being used (SA800, SA850, or SA1004).

B. CRT Controller

The devices that make up the CRT Controller provide interface for the display and CPU. The CRT Controller will convert data from the system data bus into Horizontal Sync, Vertical Sync and Video signals used by the display. The CRT Controller also handles the task of scrolling characters up the screen.

C. Timer Controller

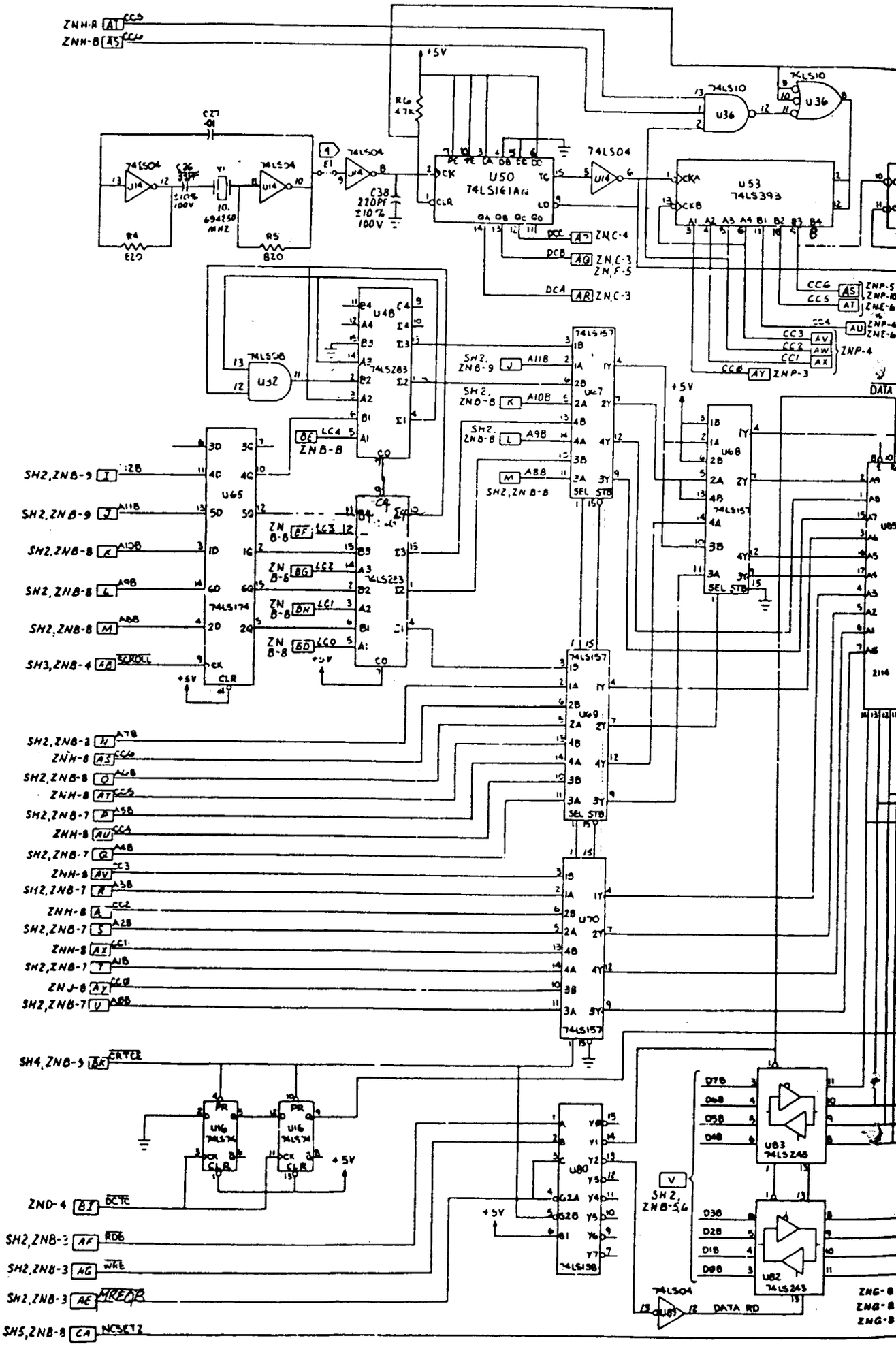
The timer controller's function is to signal the CPU when a pre-programmed amount of time has elapsed. One of the uses of this timer is the 30 second delay before turning off the 5.25" Disc Drives.

D. Serial Interface Controller

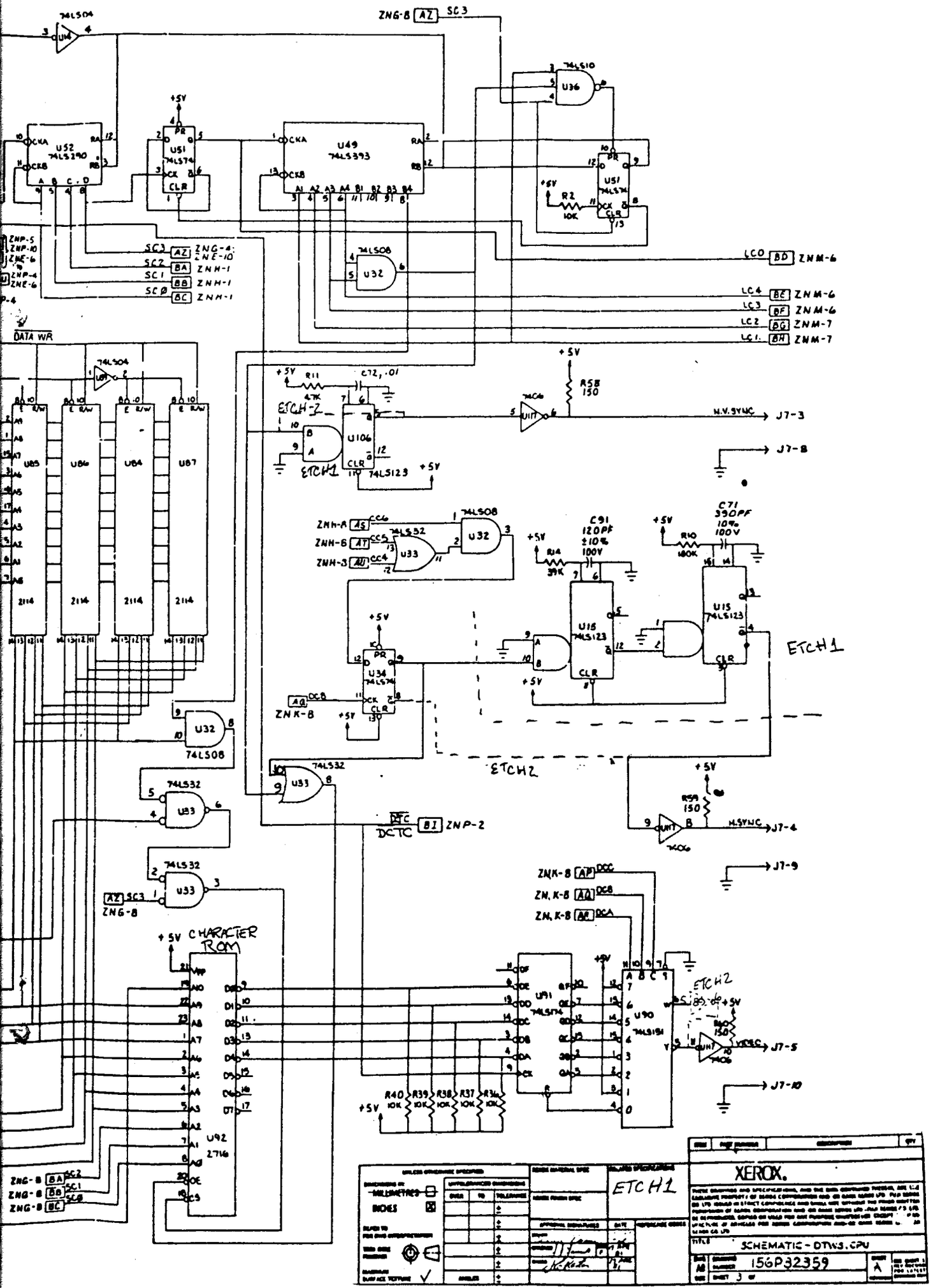
This device handles the conversion of the CPU's parallel data to serial data required for serial printers and data communications equipment (modems), also the conversion of serial data to parallel data suitable for the CPU. The controller also provides status information from the external serial device to the CPU. Modem control commands from the CPU are generated by this controller.

E. Parallel Interface Controller

This device is used as an interface between the CPU and the parallel keyboard. It also generates some control signals used as Disc Drive selects and memory bank selecting.



CRT DISPLAY GEN


74LS04
U52
74LS290
U51
74LS74
U49
74LS393
U36
74LS10
U32
74LS08
U33
74LS32
U34
74LS74
U15
74LS123
U106
74LS123
U91
74LS174
U90
74LS151
U92
2716
U85
U86
U84
U87
2114
DATA WR
ETCH-2
ETCH1
ETCH2
CHARACTER ROM
+5V
R11
4.7K
C72, .01
R58
150
R14
39K
C91
120PF
±10%
100V
C71
390PF
10%
100V
R10
180K
R59
150
R60
150
R40
R39
R38
R37
R36
10K
H.SYNC
J7-3
J7-8
J7-4
J7-9
J7-5
J7-10
ZNG-8
SC3
SC2
SC1
SC0
ZNH-1
ZNH-6
ZNM-6
ZNM-7
ZNP-2
ZNK-8
LC0
LC4
LC3
LC2
LC1
XEROX.
SCHEMATIC - DTWS.CPU
156P32359

8.4 DISC DRIVES (5.25")

The left and right disc drives are identical except for the placement of jumpers/resistor networks on the disc drive PWA's. Each of the Floppy Disc Drives contains the following.

1. DC Drive Motor
2. DC Head Stepper Motor
3. Read/Write Head
4. Head Load Solenoid And Load Pad.
5. Track - Detector Switch
6. Index Led/Detector
7. Write Protect Switch
8. Control PCB
9. Drive Indicator LED

DC Power is constantly supplied through the disc interface harness from the power supply in the processor. The DC drive motor is turned on when the appropriate control signal is active from the processor PWA. The disc drives receive control signals through the disc signal harness from the Floppy Disc Controller on the processor (CPU) PWA. These control signals select the appropriate disc drive, control the head stepper motor, the head load solenoid and select read or write modes.
The disc drives send the following status information through the disc signal harness to the Floppy Disc Controller on the Processor (CPU) PWA:

1. Ready (Floppy disc loaded and at speed)
2. Index (Index hole sensed)
3. Track 00 (Read/Write Head positioned on Track 0)
4. Write protect (Write protected disc loaded in the drive)

The function of the Disc Drives is to magnetically record (write) data on a floppy disc, and to play back (read) information that had previously been stored on a floppy disc.

8.5 DISC DRIVES (8")

The left and right Disc Drives are identical except for the placement of jumpers on the disc drive PWA.

Each of the Floppy Disc Drives contains the following:

1. AC Drive Motor
2. DC Head Stepper Motor
3. Read/Write Head
4. Head Load Solenoid and Load Pad
5. Track 00 LED/Detector
6. Index LED/Detector
7. Write Protect LED/Detector
8. Control PWA
9. Drive Indicator LED

AC power is constantly supplied through the Disc AC power cord to the drive motors from the AC Power Distribution Panel when the power on switch is on. The disc rotational speed is 360 rpm. The drive pulleys and belts are different sizes for the USO/XC systems (60Hz) and the RX systems (50 Hz) in order to obtain the 360 rpm speed.

The Internal Supply supplies DC power (+5 VDC, -5 VDC, +24 VDC and GND) through the Disc DC Harness . The DC power is used for the logic circuits and driver/receiver circuits on the PWA's. The Disc Drives receive control signals through the Disc Signal Harness from the Floppy Disc Controller on the Processor CPU PWA. These control signals select the appropriate Disc Drive, control the Head Stepper Motor, the Head Load Solenoid, and select Read or Write modes.

The Disc Drives send the following status information through the Disc Signal Harness to the Floppy Disc Controller.

1. Ready (Floppy Disc loaded and at speed)
2. Index (Index hole sensed)
3. Track 00 (Read/Write Head positioned on Track 0)
4. Write Protect (Write protected disc loaded in the drive)

The function of the Disc Drives is to magnetically record (write) data on a floppy disc, and to play back (read) information that had previously been stored on a floppy disc.

8.6 5.25" AND 8" DUAL SIDED

The SA450 and SA850 Disc Drives are also used on the 820 Family. The functions are the same as the SA400 and the SA800 with the exception of a additional signal "side select" thus allowing the Dual sided 5.25" drives to have 80 tracks and the dual sided 8" drives to have 154 tracks. On the 820-II Processor, we have double density capability. This is obtained by the use of MFM (modified frequency modulation) and M2FM (modified, modified frequency modulation) rather than FM, which is the standard method of encoding data on the diskette. This causes the write oscillator frequency to double. Data transfer rate is also doubled. Thus we now have dual sided, double density which is approximately four times the capacity of a single sided, single density.

8.7 CRT ASSEMBLY

The CRT Assembly contains a complete CRT monitor requiring only DC Power, horizontal and vertical Sync and video inputs.

The CRT has a 12 inch screen with a display capability of 24 lines of 80 characters per line. The Video rate is 15MHz.

The 820-II has Business Graphics made possible by a 4*4 Pixel Resolution. It has two sets Of 128 character sets (1 U.S. FONT, 1 GRAPHIC FONT), plus the capabilities for 2 additional sets. The 820-II also has Character Blinking and Highlighting. The RX units have a INTERNATIONAL FONT.

8.8 KEYBOARD ASSEMBLY

The Keyboard provides the keyswitches that upon activation generate the appropriate ASCII code to the parallel interface controller on the CPU PWA.

SCSI EPROM Programmer

By Terry Hazen

Special Feature

Intermediate Project

SCSI Interfacing

A Simple EPROM Programmer for the SCSI Bus

At last year's Trenton Z-Fest, I bought a set of YASBEC boards from Paul Chidley. After I got them assembled and running, the next step was to put in a 20mhz static ZS180 chip and run them at 18.4mhz. The static ZS180 has an additional control register that allows you to change the internal ZS180 clock divisor to make the ZS180 clock run at the crystal frequency instead of 1/2 the crystal frequency. You do that by sending 80h to control port 1Fh each time the Z180 is turned on. So far, so good. That's just a small program that can be run at startup time. There's just one small fly in the ointment. When you double the clock speed, you also double all the baud rates. The fastest a Z180 can drive a terminal at 9mhz is 19,200 baud and I normally use 38,400 baud. When I tried to boot at 9mhz, my boots started out with garbage on the screen until the little program that doubles the ZS180 clock speed was run and the console port baud rate doubled to 38,400 baud. The obvious answer was to add a few bytes of code to the boot EPROM to set the clock divisor before the monitor or BIOS cold boot routines were run. Since regular Z180 chips ignore port 1Fh, the new code would boot properly with either chip. Ok. Now all I needed was an EPROM programmer...

I thought I remembered seeing simple serial EPROM programmers advertised for under $100, but when I started looking, all the programmers I could find were either much too expensive or required a PC to run them. Since I don't have a PC, I decided to look into designing and building a simple and inexpensive EPROM programmer that would get all its smarts from software run on a host computer.

The first thing I had to decide was how to interface the programmer with the host computer. I have several Ampros as well as the YASBEC, so I didn't want the programmer to be computer-specific. Serial communication isn't difficult but it requires the use of computer-specific modem-type supplemental auxillary port routines. Parallel communication is better and easier but unlike the PC, neither the Ampro or YASBEC has a bidirectional parallel printer port. But the Ampro 1B, YASBEC, SB180 and most other "modern" 8-bit computers have a SCSI (Small Computer System Interface) interface available. Since the SCSI bus is an 8-bit bi-directional parallel bus, it met my requirements very nicely except for two things. I didn't know very much about it and at first glance it seemed very complicated.

A quick ZFIND search of my TCJ index file produced a number of references to the SCSI bus (see References.) Rick Lehrbaum wrote a very interesting and informative 5-part series on the SCSI bus which includes the design for a Z80 SCSI adapter board and its driver software. He also wrote an article on the use of SCSI for generalized I/O.

A similar search of my CCI index file turned up Jim MacArthur's Circuit Cellar Ink article, "Build a Simple SCSI-to-Anything Interface", which describes a simple SCSI interface design. These articles convinced me that a simple SCSI EPROM programmer board was feasible and would be a good way to get some hands-on experience with the SCSI bus.

Now for the EPROM part. A search of my article files produced several excellent EPROM programmer articles by Steve Ciarcia in BYTE. These articles cover EPROM basics, how they're programmed and how Steve chose to set his boards up to accept different EPROM types. They showed me what is required to program an EPROM.

In this first of two parts, I'll describe the SCSI EPROM programmer design. It uses Jim MacArthur's "Build a Simple SCSI-to-Anything Interface" simple discrete IC hardware design to create a "SOS" (Sort Of Scsi) interface. We pay for the hardware simplicity by requiring special SCSI drivers to communicate with the programmer board, but it turns out that they are also pretty simple. I'll talk about those software drivers in part 2. While my finished EPROM programming utility, available as EPROG10.LBR on your favorite Z-Nodes, is a ZCPR3 program that runs on both Z80 and 64180/Z180 computers, I'll try to describe the drivers in enough detail to allow a user to implement them in other languages.

Before starting the hardware design, I had to make a few other decisions. I wanted to keep the design simple and to use only easily available parts. To further simplify the design, I decided to accommodate only the newer 64k, 128k, and 256k CMOS EPROMs that use a 12.5v programming voltage. These devices share a mostly common pinout and programming sequence and are readily available by mail from sources such as JDR Microdevices and Jameco. Of course a builder can easily provide for other chip sizes and programming voltages if desired.

The SCSI Bus

The SCSI bus has 9 data lines, 9 control lines, one (often optional) termination power line and 31 ground lines. SCSI devices are usually connected with 50-conductor flat ribbon cables, which can be up to 18 feet long. All SCSI data and control lines are active low. When they're not being driven, they're pulled high by terminators in the SCSI devices at both ends of the bus cable. Each terminator consists of a 220 ohm resistor connected to +5v (or to the TERMPWR line) and a 330 ohm resistor connected to ground.

Most SCSI devices use a single dedicated SCSI interface chip like the NCR5380 to interface the SCSI bus to a host processor. Since the board will be a simple dumb low speed system with no on-board processor, a simpler and less expensive approach will do fine. Because our interface doesn't completely comform to the SCSI specifications, it needs to stay invisible to any other inhabitants of the bus whenever it doesn't have control of the bus.

The SCSI bus specification allows up to 8 devices, each assigned an ID from 0 to 7, on the bus. Devices are either initiators, usually the host computer, or targets, such as your SCSI hard disk. Although multiple initiators are allowed on the SCSI bus, for hardware simplicity we will assume that there is only one initiator, that it has a SCSI ID of 7, and that no bus arbitration is required.

The EPROM Programming Board Hardware

The programming board schematic is shown in Figure 1. The 50-conductor SCSI bus ribbon cable is connected to the programmer board through a 50pin header socket, J1.

Inverting buffers U1 and U2 are used as I/O data buffers to interface the SCSI bus to the EPROM. SCSI data bus receiver U1 is a 74LS240, which has input hystersis for good noise immunity. The SCSI bus driver U2 must be able to sink 48ma to drive the SCSI bus, so it must be an open collector or tri-state device with good sinking capabilities such as a 74AS760 or 74S240. Because I had one on hand, I used a 74S240.

When I did the programmer board layout, I provided mounting holes for the bus terminators, but I didn't install them. Since the board is very slow and in my application would be the only bus occupant, I thought that terminators at the board end wouldn't be necessary. This proved to be the case, which reduced the current sinking requirements for the bus drivers to 24ma. The board runs just fine as the sole bus target with one set of terminators at the computer end of the bus only. While it's certainly prudent to provide the specified termination, I guess I just like to live dangerously. If you do too, and if your programmer board will always be the only SCSI bus target on your system, you could use the 74LS540 or 74ACT540 for both U1 and U2 for easier board layout. Both have a straight-through pinout, which makes printed circuit board layouts much easier than the old style cris-cross pinout.

74LS151 data selector U3 detects the one-of-eight programmer board ID on the data lines during board selection. The initiator ID (7) line must remain deasserted to avoid problems with standard SCSI devices. The three data select inputs are jumpered to form the binary board ID. You can provide a set of ID jumpers as shown in the schematic or you can do as I did and simplify the board layout by fixing the board ID at 5 or some other arbitrary ID that you think will remain unused on your system.

Selecting the Programmer Board

The SCSI bus is free when BSY\ and SEL\ are deasserted (high.) When the initiator wants to select the programmer board, it asserts the SCSI data lines corresponding to the EPROM board ID, keeping its own ID line (D7\) deasserted, and then asserts SEL\ (low.) When data selector U3 sees this condition, it responds by setting the BUSY flip-flop formed by U7-U8 and asserting BSY\ through transistor Q1 to tell the initiator that the selection is complete. The initiator responds by deasserting SEL\ and removing the IDs from the data bus. The programmer board now has the bus.

Once the board has control of the bus, it retains control by keeping BSY\ asserted, and SEL\ and RST\ deasserted. As long as this condition is maintained, nothing we do will affect any of the other targets that might be on the bus.

When configured as an initiator, the 5380 can assert only ATN\ and ACK\. When configured as a target, it can assert MSG\, C/D\, I/O\ and REQ\. Since we need all the control lines we can get, we'll configure the 5380 as a target during the selection process.

Whenever the board is selected, power is applied to DPST DIP relay RY1 through transistor Q2. RY1 applies both Vcc and Vpp (the programming voltage) to the EPROM and also lights red led LED2. To prevent damage to the EPROM, it should not be removed from the ZIF socket or replaced while this LED is lit. As long as the board remains selected, both Vcc and Vpp are applied, allowing us to both read from and write to the EPROM.

Since the 5380 already uses the I/O\ line to control the direction of its own transceivers, we'll use it as our own read/write line to control our I/O data buffers. If we reserve REQ\ as a strobe line, that leaves MSG\ and C/D\ as control lines, which turns out to be just enough. 74LS138 1-of-8 decoder U4 decodes our three SCSI control lines. Figure 1 shows the decoder output line definitions. DPDT switch S1 reconfigures the EPROM pinout to use 64k/128k EPROMS or 256k EPROMs.

Addressing the EPROM

Since the EPROM programming process usually involves stepping sequentially through all the EPROM addresses, I decided to use a cascaded pair of CD4040 12-bit binary ripple counters, U5 and U6, to drive EPROM address lines A0-A14. This simple hardware arrangement means that the EPROM can't be addressed at random, but it greatly simplifies the software drivers as well as the hardware.

The EPROM address is initialized by asserting I/O\ and MSG\, deasserting C/D\ and strobing REQ\. This pulses U4 output line Y1 low and, through inverter U8, pulses the common counter reset line high to reset both counters.

The EPROM address can then be stepped as required by asserting I/O\ and C/D\, deasserting MSG\ and strobing REQ\. This pulses U4 output line Y2 low, clocking the counters to advance the current address by one byte.

Reading the EPROM

The normal state of the bus, bus free, is with all three control lines deasserted (high.) This is our read state. When the bus is free, bus driver U2 is enabled, putting the EPROM byte at the current address on the data bus, where the software can read it from the 5380.

Writing to the EPROM

We select the write mode by asserting I/O\ to enable bus receiver U1. The write mode is used whenever we need to write to the EPROM. For decoding convenience, the write mode is also used whenever we want to control the EPROM address. We can write a byte to the EPROM at the current address by asserting the data bus, putting the byte on the data bus, asserting I/O\ and thenstrobing REQ\ to apply a timed pulse to the EPROM PGM\ pin under software control.

Deselecting the Programmer Board

The programmer board is deselected by asserting all three control lines and strobing REQ\. The strobe, ORed with RST\ through U7, resets the BUSY flip-flop, removing the board from the bus and removing power from the EPROM.

Powering the Board

Since our programmer board will only require 5-6vdc and 12.5vdc, it can be easily powered by an inexpensive 12vdc 500ma wall-type plug-in unregulated power supply, such as the ones sold by Radio Shack, JDR, Jameco, etc. Provide a female PCB connector on the board to match the power supply output connector. At our much lower current levels, the supply's output voltage will be high enough that simple linear IC regulators can provide our operating voltages. I used TO-220 chips because I had them on hand. REG1, a LM340T-5, provides 5v to the interface chips. Depending on your power supply input voltage, you'll probably want to heat-sink this one.

I wanted to have some flexibility in providing EPROM voltages. REG3, an adjustable LM317T, supplies Vcc to the EPROM. The LM317 output voltage is controlled by a voltage divider. Program resistor Rp, connected between the output and adjustment terminals, is usually set to about 240 ohms. Output set resistor Ro, connected between the adjustment terminal and ground, is selected to control output voltage Vo. The following equations allow you to calculate either Ro or Vo:

$$Ro=Rp(Vo/1.25 - 1)$$
$$Vo=1.25(Ro/Rp + 1)$$

Switch S2 allows you to select an EPROM Vcc of 5v for normal use or 6v for fast programming. If you prefer, you can substitute a multi-turn pot or fix Vcc to one voltage. If the EPROMs you will be programming operate at up to 7v Vcc (including any overshoot,) as most newer ones do, a single fixed Vcc of 6v will allow both normal and fast programming. Consult your EPROM spec sheets to be sure.

REG2, a second LM317T, supplies Vpp. Although the design doesn't provide a Vpp adjustment, I have programmed 12.75v EPROMs using a Vpp of 12.5v with no problems. If you wish, you can substitute a multiturn pot for the fixed output set resistor. Capacitors C5 and C6 help reduce any overshoot spikes.

Constructing the Printed Circuit Board

Although wire-wrapping or direct point-to-point wirign are very suitable techniques, I chose to build my EPROM programmer on a 5"x7" double-sided epoxy glass board using the direct resist technique. Homemade double sided boards aren't hard to make but since you can't easily produce plated-through holes, your layout won't be as dense or efficient as it could be with commercially produced boards. Instead of plated-through holes, vias must be produced using small pieces of wire or existing component leads. To make the layout even more tricky, you have to keep in mind that you won't be able to get at the top pads of IC sockets to solder them, so you'll only be able to bring traces to ICs through the bottom pads. The same limitations apply to the SCSI socket and power connector socket.

Since I don't have a PC on which to run PADS, I did my board layout the old-fashioned way. I first laid out my board full size on tracing paper taped over a 0.100" grid on a homemade light table. I used one sheet of paper for the hole pattern. I taped another piece over it to lay out bottom side traces and later added another piece for component side traces. I provided space for a zero-insertion force (ZIF) socket for the EPROM and sockets for all ICs. Try to run all component-side traces in the long direction of the IC packages and all bottom-side traces across the IC packages.

The key to making your own double-sided boards is pattern alignment. Whether you use a program such as PADS to produce resist patterns or apply the resist directly to the board, you must have a way to align the patterns on opposite sides of the board. The easiest and most direct way to do this is to drill all the holes before you apply your resist patterns. I taped a piece of perf-board with a 0.100" grid of holes over my double-sided board to act as a drill jig and transferred the hole pattern to the jig board with a Sharpie marking pen. I drilled the holes with a surplus 0.042" diameter drill with a 0.125" shank chucked in a small drill press.

After all the holes were drilled, I carefully rubbed each surface of the board on a flat surface covered with #600 wet/dry paper to remove the drilling burrs. If you leave any burrs, you won't be able to get the resist pads sealed down over them properly. Finally, I cleaned the board with cleanser and a nylon scrub pad,

rinsed it with hot water and dried it. From now on, the board should be handled only by its edges to avoid leaving oily fingerprints on the board surfaces.

Now for the fun part. With simple boards, you can use a Sharpie marking pen to directly apply resist traces to the board, but you can't make very fine traces. You get much better results using Datak dry-transfer IC patterns and pads for the hole patterns and thin printed circuit layout tape for the traces. Datak supplies are available by mail from Jameco if you can't find them locally. All traces and patterns must be carefully burnished so etchant won't find its way under them. Dry transfer letters may also be applied if you wish to have lettering on the board.

I etched the board in a double layer of 1-quart plastic zip-lock sandwich bags containing Kepro ferric chloride etchant that I'd prewarmed in the microwave. Use enough etchant to generously cover the board. Swish the etchant around gently and constantly, keeping the board surfaces covered with it. Monitor the etching process visually, using a strong light if necessary. Don't let it etch too long or you'll start undercutting the traces and pads. The time required to completely etch the board will vary considerably with the amount of copper to be removed, the temperature of the etchant and the amount and strength of the etchant, which I reuse as many times as possible. When you're satisfied that the board is completely etched, remove it and thoroughly rinse it. Inspect it and continue the etching if it isn't completely done.

I took some laquer thinner outdoors and used a nylon scrub pad to remove the resist from the board. Then I cleaned the board with cleanser and rinsed and dried it. Now the board is ready to be inspected for incomplete etching and shorted traces. Don't feel bad if you find you have to make a number of small repairs. It's to be expected.

To finish the board, I mixed up a small quantity of Datak TINNIT tinning solution in another pair of zip-lock bags and tin plated the exposed copper. After fixing any problems caused by overetching or fine discontinuities in any of the traces, the board is ready to load.

Next time we'll look at the software required to test and control the finished SCSI EPROM programmer board.

References

1) Rick Lehrbaum, "The SCSI Interface - Introductory Column," TCJ#22

2) Rick Lehrbaum, "The SCSI Interface - Introduction to SCSI," TCJ#23

3) Rick Lehrbaum, "The SCSI Interface - The SCSI Command Protocol," TCJ#24

4) Rick Lehrbaum, "The SCSI Interface - Building a SCSI Adapter," TCJ#25

5) Rick Lehrbaum, "The SCSI Interface - Software for the SCSI Adapter," TCJ#26

6) Rick Lehrbaum, "SCSI for Generalized I/O," TCJ#31

7) Steve Ciarcia, "Adding SCSI to the SB180 Computer," BYTE, May, June 1986

8) Jim MacArthur, "Build a Simple SCSI-to-Anything Interface," Circuit Cellar Ink, Apr/May 90

9) NCR 5380 SCSI INTERFACE, Product Brief, NCR Microelectronics Division, Colorado Springs, CO

10) SCSI/PLUS Technical specification, Ampro Computers, Inc, 1985

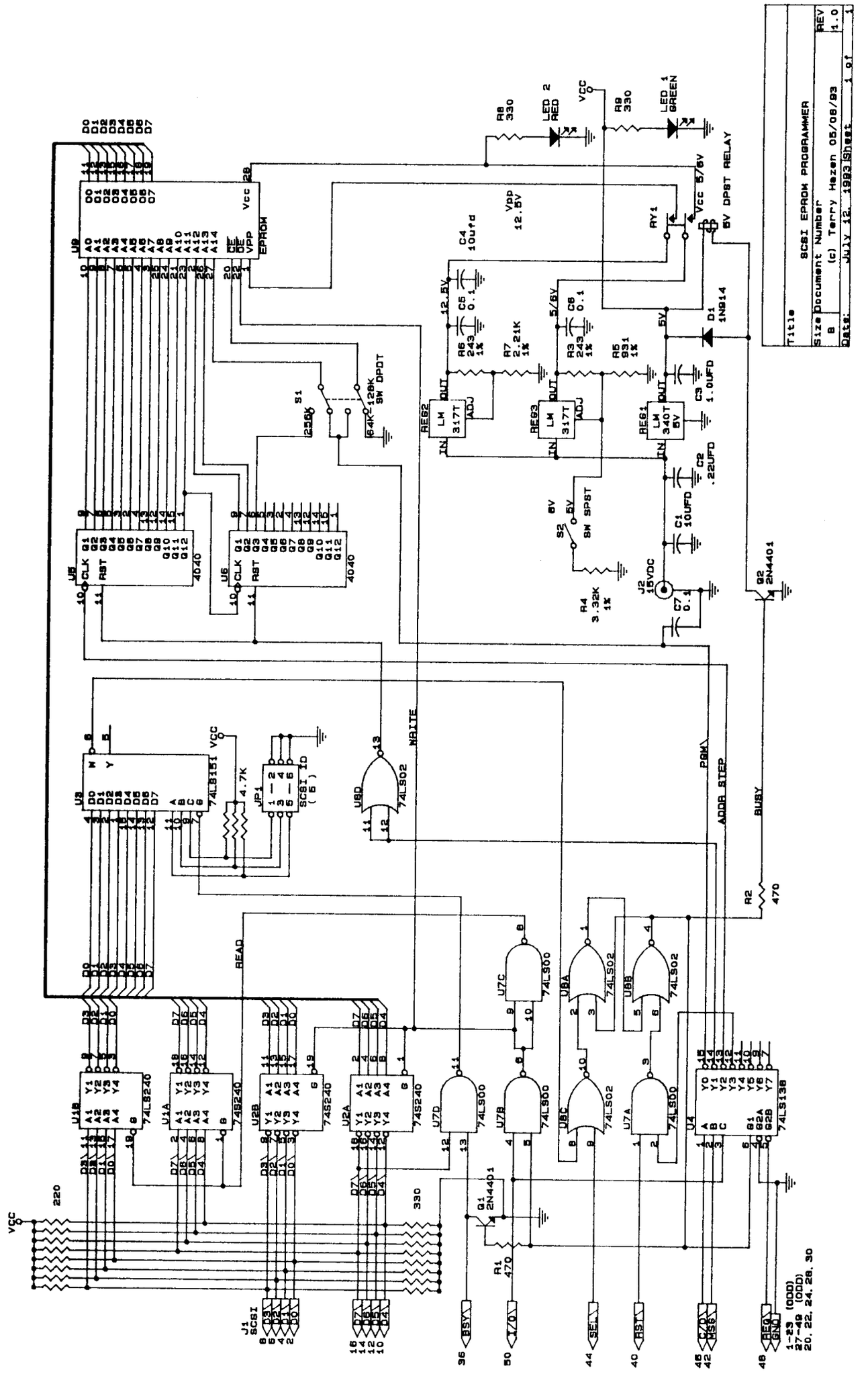
11) Steve Ciarcia, "Build an Intelligent EPROM Programmer," BYTE, October 1981

12) Steve Ciarcia, "Build a Serial EPROM Programmer," BYTE, February 1985

Figure 1. Control Line Decoding

74LS138 (U4) 1-of-8 decoder inputs/outputs. Note that all SCSI signals are active low (0=line asserted, 1=line deasserted.)

C (I\O/)	B (MSG\)	A (C/D\)	Selected output goes low	
0	0	0	Y0 (Deselect board)	\
0	0	1	Y1 (Address reset)	| Write to
0	1	0	Y2 (Address step)	| the EPROM
0	1	1	Y3 (Program EPROM)	/
1	0	0	Y4 -	\
1	0	1	Y5 -	| Read from
1	1	0	Y6 -	| the EPROM
1	1	1	Y7 (Read EPROM)	/ <- Bus free


SCSI EPROM PROGRAMMER
(c) Terry Hazen 05/08/93
July 12, 1993
5V DPST RELAY
LED 2 RED
LED 1 GREEN
EPROM
4040
74LS240
74LS138
74LS00
74LS02
74LS00
LM 317T
LM 340T 5V
2N4401
1N914
SCSI ID (5)
ADDR STEP
BUSY
READ
WRITE
PSW\
1-23 (ODD)
27-49 (ODD)
20, 22, 24, 26, 30

Special Feature

Intermediate Users

Forth Kernel Design

MOVING FORTH

by Brad Rodriguez

Part 3: Demystifying DOES>

OOPS!

There's a colossal mistake in one of my 6809 design decisions in the previous installment. It became evident when I started to code the Forth word EXECUTE.

EXECUTE causes the execution of a single Forth word, whose address is given on the Parameter Stack. (To be precise: the compilation address, a.k.a. Code Field Address, is given on the stack.) This can be any kind of Forth word: CODE definition, colon definition, CONSTANT, VARIABLE, or defined word. This differs from the usual Forth interpretation process in that the address of the word-to-execute is given on the stack, and not taken from the "thread" (as pointed to by IP).

In our direct-threaded 6809 code this can be easily coded:

```
EXECUTE:    TFR  TOS,W     put address of word in W
            PULU TOS       pop new TOS
            JMP  ,W        jump to address given in W
```

Note: this is JMP ,W and not JMP [,W], since we already have the code address of the word. We're not fetching from the high-level thread. (If TOS wasn't in register, EXECUTE could be done with simply JMP [,PSP++].)

Now suppose that this EXECUTEd word is a colon definition. W will be pointing to its Code Field, which contains JMP ENTER. This does the following (described in the previous article):

```
        JMP ENTER
        ...
ENTER:  PSHS IP
        LDX  -2,IP      re-fetch the Code Field address
        LEAY 3,X
        NEXT
```

This is the mistake! We are not executing this word from within a thread, so IP was not pointing to a copy of its Code Field address! (Remember, the address of the word-to-EXECUTE came from the stack.) This form of ENTER will not work with EXECUTE, because there is no way to find the address of the word being executed!

This suggests a new general rule for DTC Forths: if NEXT does NOT leave the address of the word-being-executed in a register, you MUST use a Call in the code field.

So, the 6809 Forth is back to using a JSR in the Code Field. But to avoid the speed penalty for ENTER -- one of the most-used code fragments in Forth -- I'll complete the "exercise for the student" from the last article. Note what happens if you swap the registers assigned to RSP and PSP:

```
        with RSP=S,     with RSP=U,
        and PSP=U       and PSP=S
        (previous)      (new)

        JSR ENTER       JSR ENTER
        ...             ...
ENTER:  PULS W          PSHU IP push old IP onto R stack
        PSHS IP         PULS IP pop new IP from JSR stack
        TFR  W,IP       NEXT
        NEXT
```

The new version executes in 31 cycles, the same as the JMP version I had wanted to use. The improvement is because the JSR version of ENTER must use both Forth's Return Stack, and the 6809 subroutine-return stack ("JSR stack"). Using two different stack pointers means we don't have to "swap" the top-of-stack with IP, eliminating the need for a temporary register.

This illustrates the usual development process for a new Forth kernel: make some design decisions, write some sample code, discover a bug or a better way to do things, throw out some code, change some design decisions, rewrite some sample code, loop until satisfied. (This is the programming equivalent of a "rip up" PC board autorouter.)

This teaches an important lesson: make EXECUTE one of your benchmark words!

OOPS, AGAIN

Carey Bloodworth of Van Buren, AR has pointed out a minor but embarassing mistake in my 6809 code in the previous installment. For the "TOS-in-memory" version of 0=, I showed the code fragment

```
        LDD ,PSP
        CMPD #0
```

to test for top-of-stack equalling zero. In this case, the CMPD instruction is completly superfluous, since the LDD instruction will set the Zero flag if D is zero! (The TOS-in-D version still requires the CMPD instruction, but remains faster than TOS-in-memory.)

Now, on to our main topic:

WHAT'S A CODE FIELD?

The DOES> concept seems to be one of the most misunderstood and mystifying aspects of Forth. Yet DOES> is also one of Forth's most powerful features -- in many ways, it anticipated object-oriented programming. The action and power of DOES> hinges upon a brilliant innovation of Forth: the Code Field.

Recall from Part 1 that the "body" of a Forth definition consists of two parts: the Code Field, and the Parameter Field. You can think of these two fields in several ways:

* The Code Field is the "action" taken by this Forth word, and the Parameter Field is the data on which it acts.

* The Code Field is a subroutine call, and the Parameter Field is parameters that are included "in-line" after the call. (The assembly language programmer's view.)

* The Code Field is the single "method" for this "class" of words, and the Parameter Field contains the "instance variables" for this particular word. (The object-oriented programmer's view.)

Common features appear in all these views:

* The Code Field routine is always called with at least one argument, namely, the address of the Parameter Field for the Forth word being executed. The Parameter Field may contain any number of parameters.

* There are relatively few distinct actions, i.e., relatively few distinct routines referenced by the Code Field. Each of these routines is widely shared (except for CODE words, as we will see later). Recall, for example, the ENTER routine from Part 2: this common routine is used by all Forth colon definitions.

* The interpretation of the Parameter Field is implicitly determined by the contents of the Code Field. I.e., each Code Field routine expects the Parameter Field to contain a certain kind of data.

A typical Forth kernel will have several Code Field routines predefined.

Code Field routine	Parameter Field contents
ENTER	a high-level "thread" (series of addresses)
DOCON	a constant value
DOVAR	a storage location for data
DOVOC	vocabulary info (varies by implementation)

What makes this feature powerful is that a Forth program is not limited to this set of Code Field routines (or whatever set is provided in your kernel). The programmer can define new Code Field routines, and new Parameter Fields to match. In object-oriented lingo, new "classes" and "methods" can be created (although each class has only one method). And -- like Forth words themselves -- the Code Field actions can be defined in either assembly language or high-level Forth!

To understand the mechanism of the Code Field, and how parameters are passed, we will first look at the case of assembly-language (machine code) actions. We'll start with Indirect Threading (ITC), since it is the easiest to understand, and then see how the logic is modified in Direct-Threaded (DTC) and Subroutine-Threaded (STC) Forths. Then, we'll look at how the Code Field action can be written in high-level Forth.

Forthwrights are somewhat inconsistent in their terminology, so I'll define my terms, using the ITC Forth word illustrated in Figure 1. The Header contains the dictionary information, and isn't involved in the execution of the Forth word. The Body is the "working" part of the word, and consists of the fixed-length Code Field, and the variable-length Parameter Field. For any given word, the locations of these two fields in memory are the Code Field Address (CFA) and the Parameter Field Address (PFA), respectively. The Code Field Address of a word is the address in memory where its Code Field is located. This is not to be confused with the contents of the Code Field, which, in ITC Forths, is another different address. To be specific, the contents of the Code Field is the address of a fragment of machine code somewhere else in memory. I will refer to this as the Code Address. Later, when in discussing DTC and STC Forths, I will also refer to the "Code Field contents," which will include more than just the Code Address.

MACHINE-CODE ACTIONS

Forth CONSTANTs are probably the simplest example of a machine-code action. Let's consider some good Francophone constants

```
1 CONSTANT UN
2 CONSTANT DEUX
3 CONSTANT TROIS
```

Executing the word UN will push the value 1 onto the Forth Parameter Stack. Executing DEUX will push a 2 onto the stack, and so on. (Don't confuse Parameter Stack with Parameter Field; they are entirely separate.)

In the Forth kernel there is a single word called CONSTANT. This is not a constant-type word itself; it is a high-level Forth definition. CONSTANT is a "defining word": it creates new words in the Forth dictionary. Here we create the new "constant-type" words UN, DEUX, and TROIS. (You may think of these as "instances" of the "class" CONSTANT.) These three words will have their Code Fields pointing to a machine code fragment that does the action of CONSTANT.

What must this code fragment do? Figure 2 shows the memory representation of the three constants. All three words point to a common action routine. The difference in the words is entirely contained in their Parameter Fields, which, in this case, simply hold the constant values ("instance variables" in object lingo). So, the action of these three words should be fetch the contents of the Parameter Field, and push this onto the stack. The code understands implicitly that the parameter field contains a single-cell value.

To write a machine-code fragment to do this, we need to know how to find the Parameter Field Address, after the Forth interepreter jumps to the machine code. That is, how is the PFA passed to the machine-code routine? This, in turn, depends on how the Forth interpreter NEXT has been coded, which varies from implementation to implementation. To write machine-code actions, we must understand NEXT.

The ITC NEXT was described in pseudo-code in Part 1. Here's one implementation for the 6809, using Y=IP and X=W:

```
NEXT:  LDX ,Y++        ; (IP) -> W, and IP+2 -> IP
       JMP [,X]        ; (W) -> temp, JMP (temp)
```

Suppose that we're in a high-level thread

```
... SWAP DEUX + ...
```

with the Interpreter Pointer (IP) pointing to the DEUX "instruction," when NEXT is executed. (This would be at the very end of SWAP.) Figure 3 illustrates what happens. IP (register Y) is pointing within the high-level thread, at a memory cell that contains the address of the Forth word DEUX. To be precise, this cell contains the Code Field Address of DEUX. So, when we fetch a cell

using Y, and autoincrement Y, we fetch the Code Field Address of DEUX. This goes into W (register X), so W is now pointing to the Code Field. The contents of this field is the address of some machine code. We can fetch the contents of this cell and jump to the machine code with a single 6809 instruction. This leaves register X unchanged, so W is still pointing to the CFA of DEUX. This is how the Parameter Field Address is obtained, since, in this case, it is simply two bytes past the Code Field.

So, the machine code fragment has only to add 2 to W, fetch the cell value at that address, and push that on the stack. This fragment is frequently called DOCON:

```
DOCON:LDD    2,X     ; fetch the cell at W+2
       PSHU D ; push that on the Parameter Stack
       NEXT    ; (macro) do the next high-level word
```

(For this example, TOS is kept in memory.) Note that the previous NEXT incremented IP by 2, so it is already pointing to the next cell in the thread (''CFA of +'') when DOCON does NEXT.

In general, ITC Forths leave the Parameter Field Address or some ''nearby'' address in the W register. In this case, W contained the CFA, which in this Forth implementation is always PFA-2. Since every class of Forth word except CODE words needs to use the Parameter Field Address, many implementations of NEXT will increment W to leave it pointing to the PFA. We can do this on the 6809 with one small change:

```
NEXT:  LDX ,Y++        ; (IP) -> W, and IP+2 -> IP
       JMP [,X++]      ; (W) -> temp, JMP (temp), W+2 -> W
```

This adds three clock cycles to NEXT, and leaves the Parameter Field Address in W. What does it do to the Code Field routines?

```
              W=CFA          W=PFA

DOCON:LDD 2,X (6)     LDD ,X  (5)
       PSHU D         PSHU D
       NEXT           NEXT

DOVAR: LEAX 2,X (5)   ; no operation
       PSHU X         PSHU X
       NEXT           NEXT

ENTER: PSHS Y         PSHS Y
       LEAY 2,X (5)   LEAY ,X  (4, faster than TFR X,Y)
       NEXT           NEXT
```

In exchange for a three-cycle penalty in NEXT, the DOCON code is reduced by one clock cycle, DOVAR by five cycles, and ENTER by one cycle. CODE words don't use the value in W, so they gain nothing from the autoincrement. The speed gained or lost is determined by the mix of Forth words executed. The usual rule is that most of the words executed are CODE words, thus, incrementing W in NEXT costs a bit of speed overall. (There is a memory savings, but DOCON, DOVAR, and ENTER appear only once, making this gain slight.)

The best decision, of course, depends upon the processor. On machines like the Z80, which only access memory by bytes and don't have autoincrement address modes, it is often best to leave W pointing to IP+1 (the last byte fetched from the Code Field). On other machines, autoincrementing is ''free,'' and leaving W pointing to the Parameter Field is most convienient.
Remember: the decision must be made consistently. If NEXT leaves W pointing to the PFA of the word being executed, then EXECUTE must do likewise! (This was the 'oops' that I corrected at the start of this article.)

Direct Threading

Direct Threading works just like Indirect Threading, except that instead of the Code Field containing the address of some machine code, it contains a JUMP or CALL to some machine code. This makes the Code Field larger -- e.g., 1 byte larger in the 6809 -- but removes one level of indirection from the NEXT routine.

The choice of a JUMP or a CALL instruction in the Code Field hinges upon how the Parameter Field Address can be obtained by the machine code routine. In order to jump to the Code Field, many CPUs require that its address be in a register. For instance, the indirect jump on the 8086 is JMP AX (or some other register), and on the Z80 is JP (HL) (or IX or IY). On these processors, the DTC NEXT involves two operations, which on the 6809 would be:

```
NEXT:  LDX ,Y++        ; (IP) -> W, and IP+2 -> IP
       JMP ,X          ; JMP (W)
```

(On the 8086, this can be done with LODSW, JMP AX.) The effect of this is illustrated in Figure 4 as ''case 1''. The Code Field Address of DEUX is fetched from the high-level thread, and IP is incremented. Then, instead of a fetch, a JUMP is made to the Code Field Address (i.e., the CPU jumps directly to the Code Field). The CFA is left in the W register, just like the first ITC example above. Since this address is already in a register, we can simply put a JUMP to DOCON in the Code Field, and the DOCON fragment will work the same as before.

However, some processors -- such as the 6809 and PDP-11 -- can do this DTC NEXT in one instruction:

```
NEXT:  JMP [,Y++]      ; (IP) -> temp, IP+2 -> IP, JMP (temp)
```

This, too, will cause the CPU to jump to the Code Field of DEUX. But there's one big difference: the CFA is not left in any register! So how is the machine code fragment to find the Parameter Field Address? By putting a CALL (JSR) in the Code Field instead of a JUMP. On most CPUs, the CALL instruction will push the return address -- the address immediately following the CALL instruction -- onto the Return Stack. As Figure 4 illustrates (''case 2''), this return address is exactly the Parameter Field Address we want! So, all DOCON has to do is pop the Return Stack -- balancing the JSR in the Code Field -- and then use that address to fetch the constant value. Thus:

```
DOCON:PULS X           ; pop the PFA from the Return Stack
       LDD ,X          ; fetch the Parameter Field cell
       PSHU D          ; push that on the Parameter Stack
       NEXT            ; (macro) do the next high-level word
```

Compare this with the ITC version. One instruction has been added to DOCON, but one instruction has been deleted from NEXT. DOVAR and NEXT likewise become one instruction longer:

```
DOVAR: PULS X  ; pop the PFA of the word
       PSHU X  ; push that address on the Parameter Stack
       NEXT

ENTER: PULS X  ; pop the PFA of the word
       PSHS Y  ; push the old IP
       TFR X,Y ; the PFA becomes the new IP
       NEXT
```

Now go back to the beginning of this article, and reread my "oops," to see why we can't just re-fetch the CFA by using the IP. Also note the difference when the assignment of Forth's stack pointers to the 6809's U and S is reversed.

Subroutine Threading

Subroutine Threading (STC) is like DTC in that the CPU jumps directly to the Code Field of a Forth word. Only now there is no NEXT code, no IP register, and no W register. So, there is no choice but to use a JSR in the Code Field, since this is the only way to obtain the Parameter Field Address. This process is illustrated in Figure 5. The high-level "thread" is a series of subroutine calls being executed by the CPU. When the JSR DEUX is executed, the address of the next instruction in the thread is pushed onto the Return Stack. Then, the JSR DOCON within the word DEUX is executed, which causes another return address -- the PFA of DEUX -- to be pushed onto the Return Stack. DOCON can pop that address, use it to fetch the constant, stack the constant, and then do an RTS to return to the thread:

```
DOCON:PULS X        ; pop the PFA from the Return Stack
      LDD ,X        ; fetch the Parameter Field cell
      PSHU D        ; push that on the Parameter Stack
      RTS           ; do the next high-level word
```

We can still speak of a Code Field and a Parameter Field in Subroutine-Threaded Code. In every "class" of Forth word except CODE and colon defintions, the Code Field is the space occupied by a JSR or CALL instruction (just like DTC), and the Parameter Field is what follows. So, on the 6809, the PFA would equal CFA+3. The meaning of "Parameter Field" becomes somewhat fuzzy in CODE and colon definitions, as will be seen in future articles.

THE SPECIAL CASE: CODE WORDS

There is a significant exception to all of the above generalizations. This is CODE definitions -- Forth words that are defined as a machine code subroutine. This wonderful capability is trivially easy to implement in Forth, since every Forth word executes some piece of machine code!

The machine code comprising a CODE word is always contained in the body of the Forth word. In an Indirect-Threaded Forth, the Code Field must contain the address of the machine code to be executed. So the machine code is placed in the Parameter Field, and the Code Field contains the address of the Parameter Field, as shown in Figure 6.

In Direct- and Subroutine-Threaded Forths, we could -- by analogy -- put, in the Code Field, a JUMP to the Parameter Field. But this would be pointless, since the Parameter Field immediately follows the Code Field! The Code Field could be filled with NOPs for the same result. Better still, the machine code could be started at the Code Field, and continued into the Parameter Field. At this point the distinction of "Code Field" and "Parameter Field" breaks down. This is no problem, because we don't need this distinction for CODE words. (This does have ramifications for decompilers and certain clever programming tricks, none of which concern us here.)

CODE words -- whatever the implementation -- are the one case where the machine code "action" routine does not need to be passed the Parameter Field address. The Parameter Field contains, not data, but the code being executed! Only NEXT needs to know this address (or the Code Field Address), so it can jump to the machine code.

USING ;CODE

Three questions remain unanswered:

a. how do we create a new Forth word that has some arbitrary data in its Parameter Field?

b. how do we change the Code Field of that word, to point to some machine code of our choosing?

c. how do we compile (assemble) this machine code fragment, which exists in isolation from the words using it?

The answer to (a) is: we write a Forth word to do this. Since this word, when executed, will define (create) a new word in the Forth dictionary, it is called a "defining word." CONSTANT is one example of a defining word. All of the "hard work" of a defining word is done by a kernel word, CREATE, which parses a name from the input stream, builds the header and Code Field for a new word, and links it into the dictionary. (In fig-Forth this word is called <BUILDS.) All that remains for the programmer is to build the Parameter Field.

The answer to (b) and (c) is embodied in two convoluted words called (;CODE) and ;CODE respectively. To understand how they work, let's look at how the defining word CONSTANT is actually written in Forth. Using the original ITC 6809 example:

```
: CONSTANT  ( n -- )
      CREATE              \ create the new word
      ,                   \ append the TOS value to the dictionary,
                          \  as the 1st cell of the Parameter Field
      ;CODE               \ end high-level & start assembler code
      LDD 2,X             \ the code fragment DOCON
      PSHU D              \ "  "    "     "
      NEXT                \ "  "    "     "
      END-CODE
```

There are two parts to this Forth word. Everything from **: CONSTANT** to **;CODE** is the high-level Forth code executed when the word CONSTANT is invoked. Everything from **;CODE** to **END-CODE** is machine code executed when the "children" of CONSTANT -- the "constant-class" words such as UN and DEUX -- are executed. That is, everything from ;CODE to END-CODE is the code fragment to which constant-type words will point. The name ;CODE signifies that it ends a high-level definition (";") and begins a machine-code definition ("CODE"). However, this is not put into the dictionary as two separate words. Everything from **: CONSTANT** to **END-CODE** is contained in the Parameter Field of CONSTANT, as shown in Figure 7.

Derick and Baker [DER82] name three "sequences" that help to understand the action of defining words:

Sequence 1 is when the word CONSTANT is being defined. This involves both the high-level compiler (for the first part) and the Forth assembler (for the second part). This is when the definition of CONSTANT shown in Figure 7 is added to the dictionary. As we will see shortly, ;CODE -- a compiler directive -- is executed during Sequence 1.

Sequence 2 is when the word CONSTANT is being executed, and when some constant-type word is being defined. In the example

2 CONSTANT DEUX

Sequence 2 is when the word CONSTANT executes, and the word DEUX is added to the dictionary (as shown in Figure 7). During

Sequence 2, the high-level part of CONSTANT is executed, including the word (;CODE).

Sequence 3 is when the constant-type word is executed. In our example, Sequence 3 is when DEUX is executed to push the value 2 onto the stack. This is when the machine-code part of CONSTANT is executed. (Recall that this fragment is the Code Field action of DEUX.)

The words ;CODE and (;CODE) do the following:

;CODE is executed during Sequence 1, when CONSTANT is compiled. This is an example of a Forth IMMEDIATE word -- a word executed during the Forth compilation. ;CODE does three things:

a. it compiles the Forth word (;CODE) into CONSTANT,

b. it turns off the Forth compiler, and

c. it turns on the Forth assembler.

(;CODE) is part of the word CONSTANT, so it executes when CONSTANT executes (Sequence 2). It performs the following actions:

a. It gets the address of the machine code that immediately follows. This is done by popping IP from the Forth Return Stack.

b. It puts that address into the Code Field of the word just defined by CREATE. The Forth word LAST (sometimes LATEST) gets the address of that word.

c. It does the action of EXIT (a.k.a. ;S) so that the Forth inner interpreter doesn't try to execute the machine code that follows as part of the Forth thread. This is the high-level "subroutine return" which ends a Forth thread.

F83 [LAX84] illustrates how these are typically coded in Forth:

```
: ;CODE
        COMPILE (;CODE)         \ compiles (;CODE) into definition
        ?CSP  [COMPILE] [       \ turns off the Forth compiler
        REVEAL                  \  (just like ";" does)
        ASSEMBLER               \ turns on the assembler
        ; IMMEDIATE             \ this is an IMMEDIATE word!

: (;CODE)
        R>                  \ pops the adrs of the machine code
        LAST @ NAME>        \ gets the CFA of the latest word
        !                   \ stores the code address in the
        ;                   \  Code Field
```

(;CODE) is the more subtle of the two. Since it is a high-level Forth definition, the address following it in the CONSTANT thread -- the high-level "return address" -- is pushed onto Forth's Return Stack. So, popping the Return Stack while within (;CODE) will yield the address of the machine code that follows. Also, popping this value from the Return Stack will "bypass" one level of high-level subroutine return, so that when (;CODE) exits, it will exit to the caller of CONSTANT. This is equivalent to returning to CONSTANT, and then having CONSTANT return immediately. Use Figure 7 and walk through the execution of the words CONSTANT and (;CODE) to see how this works.

Direct and Subroutine Threading

For DTC and STC, the action of ;CODE and (;CODE) is identical to ITC, with one important exception: instead of holding an address, the Code Field holds a JUMP or CALL instruction. For an absolute JUMP or CALL, probably the only difference is that the address has to be stored at the end of the Code Field, as the operand of the JUMP or CALL instruction. In the case of the 6809, the address would be stored as the last two bytes of the three-byte JSR instruction. But some Forths, such as Pygmy Forth on the 8086, use a relative branch in the code field. In this case, the relative offset must be computed and inserted into the branch instruction.

HIGH-LEVEL FORTH ACTIONS

We have seen how to make a Forth word execute a chosen fragment of machine language code, and how to pass that fragment the address of the word's Parameter Field. But how do we write the "action routine" in high-level Forth?

Every Forth word must -- by the action of NEXT -- execute some machine language routine. This is what the Code Field is all about. Therefore, a machine language routine, or a set of routines, is needed to handle the problems of invoking a high-level action. We'll call this routine DODOES. There are three problems to be solved:

a. how do we find the address of the high-level action routine associated with this Forth word?

b. how do we, from machine code, invoke the Forth interpreter for a high-level action routine?

c. how do we pass that routine the address of the Parameter Field for the word we are executing?

The answer to (c) -- how do you pass an argument to a high-level Forth routine -- is easy. On the Parameter Stack, of course. Our machine language routine must push the Parameter Field Address on the stack before it invokes the high level routine. (From our previous work, we know how the machine language routine can obtain the PFA.)

The answer to (b) is a bit more difficult. Basically, we want to do something like the Forth word EXECUTE, which invokes a Forth word; or perhaps ENTER, which invokes a colon definition. Both are among our "key" kernel words. The DODOES code will resemble these.

Question (a) is the tricky one. Where to put the address of the high-level routine? Remember, the Code Field does not point to high-level code; it must point to machine code. Two approaches have been used in the past:

1. **The fig-Forth solution.** Fig-Forth reserved the first cell of the Parameter Field to hold the address of the high-level code. The DODOES routine then obtained the Parameter Field address, pushed the address of the actual data (typically PFA+2) onto the stack, fetched the address of the high-level routine, and EXECUTEd.

There were two problems with this approach. First, the structure of the Parameter Field was different for machine-code actions and high-level actions. For example, a CONSTANT defined with a machine code action would have its data stored at PFA, but a CONSTANT defined with a high-level action would have its data stored at (typically) PFA+2.

Second, every instance of a high-level-action class carried an additional overhead of one cell. That is, if CONSTANT used a high-level action, every constant defined in the program was one cell larger!

Fortunately, clever Forth programmers quickly devised a solution which overcame these problems, and the fig-Forth approach has fallen into disuse.

2. The modern solution. Most Forths nowadays associate a different machine language fragment with each high-level action routine. So, a high-level constant would have its Code Field pointing to a machine language fragment whose sole function is to invoke the high-level action of CONSTANT. A high-level variable's Code Field would point to the "startup" routine for the high-level VARIABLE action, and so on.

Is this excessive duplication of code? No, because each of these machine-language fragments is just a subroutine call to a common startup routine, DODOES. (This is different from the fig-Forth DODOES routine.) The address of the high-level code to DODOES is passed as an "inline" subroutine parameter. That is, the address of the high-level code is put immediately after the JSR/CALL instruction. DODOES can then pop the CPU stack and do a fetch to obtain this address.

Actually, we make two more simplifications. The high-level code itself is put immediately after the JSR/CALL instruction. Then DODOES pops the CPU stack, and obtains this address directly. And since we know this is high-level Forth code, we dispense with its Code Field and just compile the high-level thread...essentially incorporating the action of ENTER into DODOES.

Now each "defined" word just points to a bit of machine code...no space is consumed in its Parameter Field. This bit of machine code is a JSR or CALL instruction, followed by the high-level action routine. In the 6809 example, we have traded two bytes in every constant for a three-byte JSR that appears only once.

This is undoubtedly the most convoluted program logic in the entire Forth kernel! So, let's see how this is implemented in practice, using our trusty ITC 6809 example.

Figure 8 shows the constant DEUX implemented with a high-level action. When the Forth interpreter encounters DEUX -- that is, when the Forth IP is at IP(1) -- it does the usual thing: it fetches the address contained in DEUX's Code Field, and jumps to that address. At that address is a JSR DODOES instruction, so a second jump -- this time a subroutine call -- is immediately taken. DODOES must then perform the following actions:

a. Push the address of DEUX's Parameter Field onto the Parameter Stack, for later use by the high-level action routine. Since the JSR instruction does not alter any registers, we expect to find the Parameter Field Address of DEUX (or a "nearby" address) still in the W register.

b. Obtain the address of the high-level action routine, by popping the CPU stack. (Recall that popping the CPU stack will give the address of whatever immediately follows the JSR instruction.) This is a high-level thread, i.e., the Parameter Field part of a colon definition.

c. Save the old value of Forth's Instruction Pointer -- IP(2) -- on Forth's Return Stack, since the IP register will be used to execute the high-level fragment. Essentially, DODOES must "nest" the IP, just like ENTER does. Remember that Forth's Return Stack may not be the same as the CPU subroutine stack.

d. Put the address of the high-level thread into IP. This is IP(3) in Figure 8.

e. Do a NEXT to continue high-level interpretation at the new location.

Assume an indirect-threaded ITC 6809, and the following:

* W is not incremented by NEXT (i.e., W will contain the CFA of the word entered by NEXT);
* the 6809 S is Forth's PSP, and U is Forth's RSP (i.e., the CPU stack is not Forth's Return Stack);
* the 6809 Y is Forth's IP, and X is Forth's W.

Recall the definition of NEXT for these conditions:

```
NEXT:  LDX ,Y++        ; (IP) -> W, and IP+2 -> IP
       JMP [,X]        ; (W) -> temp, JMP (temp)
```

DODOES can be written as follows:

```
DODOES: LEAX 2,X       ; make W point to the Parameter Field
        PSHU Y         ; (c) push old IP onto the Return Stack
        PULS Y         ; (b,d) pop new IP from the CPU stack
        PSHS X         ; (a) push W (the Parameter Field
                       ;    Address) onto the Parameter Stack
        NEXT           ; (e) invoke high-level interpreter
```

These operations are slightly out of sequence. As long as the right things go onto the right stacks (or into the right registers) at the right time, the exact order of operations is not critical. In this case, we're taking advantage of the fact that the old IP can be pushed onto Forth's Return Stack before the new IP is popped from the CPU stack.

On some processors the CPU stack is used as Forth's Return Stack. In this case, one step involving temporary storage is necessary. If we had chosen S=RSP and U=PSP above, DODOES would be:

```
DODOES:LEAX 2,X        ; make W point to the Parameter Field
       PSHU X          ; (a) push W (the Parameter Field
                       ;    Address) onto the Parameter Stack
       PULS X          ; (b) pop thread address from CPU stack
       PSHS Y          ; (c) push old IP onto the Return Stack
       TFR X,Y         ; (d) put thread address into IP
       NEXT            ; (e) invoke high-level interpreter
```

Since we are essentially swapping the top of the Return/CPU stack with IP, we need to use X as a temporary holding register. Thus we must push the PFA -- step (a) -- before re-using the X register.

Walk through both of these DODOES examples step by step, and track the contents of the registers and the two stacks. I always walk through my DODOES routine, just to make sure I'm not clobbering a register at the wrong time.

Direct Threading

The logic of DODOES is the same in DTC Forths. But the implementation may be different, depending on whether the DTC Forth uses a JMP or a CALL in the Code Field of a word.

a. **JMP in Code Field.** A DTC Forth can use a JMP in the Code Field if the address of the word being executed is found in a register. This will most likely be the Code Field Address.

From the point of view of DODOES, this is identical to ITC. In our example, DODOES sees that the Forth interpreter jumps to the machine code associated with DEUX, and that code is a JSR to DODOES. It doesn't matter that the first jump is now a direct jump rather than an indirect jump; the register and stack contents are the same. So, the code for DODOES will be identical to that for ITC. (Of course, NEXT is different, and W may need a different offset to point to the Parameter Field.)

b. **CALL/JSR in Code Field.** In the DTC 6809, we never explicitly fetch the CFA of the word being executed, so the Forth word must contain a JSR in its Code Field. Instead of finding the Parameter Field Address of the Forth word in a register, we find it on the CPU stack.

The DEUX example in this case is shown in Figure 9. When the Forth IP is at IP(1), the Forth interpreter jumps to the Code Field of DEUX (and increments IP). In the Code Field is a JSR to DEUX's machine code fragment. At that address is a second JSR, to DODOES. So two things get pushed onto the CPU stack. The return address of the first JSR is the Parameter Field address of DEUX. The return address of the second JSR -- and thus topmost on the CPU stack -- is the address of the high-level thread to be executed. DODOES must ensure that the old IP is pushed onto the Return Stack, the PFA of DEUX is pushed onto the Parameter Stack, and the address of the high-level thread is loaded into IP. This is very sensitive to stack assignments! For S=PSP (CPU stack) and U=RSP, the NEXT and DODOES code is:

```
NEXT:   LDX [,Y++]      ; (IP) -> temp, IP+2 -> IP, JMP (temp)

DODOES: PSHU Y          ; push old IP onto the Return Stack
        PULS Y          ; pop new IP from the CPU stack
                        ; note: the CPU stack is the Parameter
                        ; Stack, and the topmost element is
                        ; now the PFA of the word...
                        ; exactly what we want!
        NEXT            ; invoke high-level interpreter
```

Check for yourself that the flow through NEXT, DEUX, and DODOES pushes a net total of one item -- the PFA of DEUX -- onto the Parameter Stack!

Subroutine Threading

In STC Forths, there are no IP or W registers, and a high-level "thread" is pure machine code (a series of subroutine calls). The only difference between a high-level action and a ;CODE action is that the PFA of the "defined" word must be pushed onto the Parameter Stack. "Defined" words have a CALL/JSR in the Code Field, and the CPU stack must be Forth's Return Stack, so DODOES is mostly a matter of stack manipulations.

Figure 10 shows a 6809 STC example of DEUX with a high-level action. By the time DODOES is entered, three things have been pushed onto the CPU/Return Stack: the return address in the "main" thread, the PFA of DEUX, and the address of DEUX's high-level action code. DODOES must pop the last two, push the PFA onto the Parameter Stack, and jump to the action code:

```
DODOES: PULS X,Y        ; action code adrs -> X, PFA -> Y
        PSHU Y          ; push PFA onto Parameter Stack
        JMP ,X          ; jump to the action code
```

DODOES for the 6809 is now a three-instruction routine. It can be simplified even further by "expanding JSR DODOES in-line," i.e., replacing the JSR DODOES with the equivalent machine code instructions. Since there's one less JSR, this simplifies the stack manipulation to:

```
        PULS X          ; pop PFA from CPU stack
        PSHU X          ; and push it onto the Parameter Stack
                        ...high level thread for DEUX...
```

This replaces a three-byte JSR with four bytes of explicit code, with a considerable improvement in speed. For the 6809 this would probably be a good choice. For a processor like the 8051, DODOES is long enough that it should be kept as a subroutine.

USING DOES>

We learned with ;CODE how to create a new Forth word with arbitrary data in its parameter field, and how to make that word's Code Field point to a new machine code fragment. How do we compile a high-level action routine, and make a new word point to it?

The answer lies in the two words DOES> and (DOES>), which are the high-level equivalents of ;CODE and (;CODE). To understand them, let's look at an example of their use:

```
: CONSTANT   ( n -- )
        CREATE              \ create the new word
        ,                   \ append the TOS value to the dictionary,
                            \  as the 1st cell of the Parameter Field
        DOES>               \ end "create" part & start "action" part
        @                   \ given the PFA, fetch its contents
        ;
```

Compare this with the previous ;CODE example, and observe that DOES> performs a function analogous to ;CODE. Everything from **: CONSTANT** to **DOES>** is executed when the word CONSTANT is invoked. This is the code which builds the Parameter Field of the "defined" word. Everything from **DOES>** to **;** is the high-level code executed when the "children" of CONSTANT (such as DEUX) are invoked, i.e., the high-level fragment to which the Code Field will point. (We'll see that a JSR DODOES is included before this high-level fragment.) Just as with ;CODE, both the "create" and the "action" clauses are contained within the body of the Forth word CONSTANT, as shown in Figure 11.

Recall Sequence 1, 2, and 3. The words DOES> and (DOES>) do the following:

DOES> is executed during Sequence 1, when CONSTANT is compiled. Thus DOES> is a Forth IMMEDIATE word. It does two things:

a. It compiles the Forth word (DOES>) into CONSTANT.

b. It compiles a JSR DODOES into CONSTANT.

Note that DOES> leaves the Forth compiler running, in order to compile the high-level fragment which follows. Also, even though JSR DODOES is not itself Forth code, an IMMEDIATE word such as DOES> can cause it to be compiled in the middle of Forth code.

(DOES>) is part of the word CONSTANT, so it executes when

CONSTANT executes (Sequence 2). It does the following:

a. It gets the address of the machine code that immediately follows (JSR DODOES), by popping IP from the Forth Return Stack.

b. It puts that address into the Code Field of the word just defined by CREATE.

c. It performs the action of EXIT, causing CONSTANT to terminate here and not attempt to execute the fragment that follows.

The action of (DOES>) is identical to (;CODE)! A separate word is not strictly required. F83, for example, uses (;CODE) in both ;CODE and DOES>. I'll use (;CODE) from now on instead of (DOES>).

You've already seen the workings of (;CODE). The F83 definition of DOES> is

```
: DOES>
        COMPILE (;CODE)          \ compiles (;CODE) into definition
        0E8 C,                   \ the CALL opcode byte
        DODOES HERE 2+ - ,       \ the relative offset to DODOES
        ; IMMEDIATE
```

where DODOES is a constant which holds the address of the DODOES routine. (The actual F83 source code is slightly different, due to the requirements of the F83 metacompiler.) DOES> need not fiddle with CSP or the smudge bit, since the Forth compiler is left "on." In the case of the 8086, the CALL instruction expects a relative address...hence the arithmetic involving DODOES and HERE. In the 6809, DOES> would look like

```
: DOES>
        COMPILE (;CODE)          \ compiles (;CODE) into definition
        0BD C,                   \ the JSR Extended opcode byte
        DODOES ,                 \ the operand: address of DODOES
        ; IMMEDIATE
```

You can see here how the machine language JSR DODOES is compiled after the high-level (;CODE), and before the high-level "action" code.

Direct and Subroutine Threading

The only difference in DTC and STC is how the Code Field is fiddled to point to a new routine. This is done by (;CODE), and the required changes have already been described. DOES> isn't affected at all, unless you're writing an STC Forth and expanding the JSR DODOES to explicit machine code. In this case, DOES> is modified to assemble the "in-line" machine code instead of a JSR DODOES instruction.

ONWARD AND UPWARD

Who would have thought that so few lines of code would require so much explanation? This is why I admire ;CODE and DOES> so much...I've never before seen seen such intricate, powerful, and flexible constructs coded with such economy.

In the next installment I'll discuss the merits of assemblers vs. metacompilers, and provide the actual CODE definitions for our Forth example systems.

REFERENCES

[DER82] Derick, Mitch and Baker, Linda, Forth Encyclopedia, Mountain View Press (1982). A word-by-word description of fig-Forth in minute detail. Still available from the Forth Interest Group, P.O. Box 2154, Oakland CA 94621.

[LAX84] Laxen, H. and Perry, M., F83 for the IBM PC, version 2.1.0 (1984). Distributed by the authors, available from the Forth Interest Group or GEnie.

FIGURE 1. AN ITC FORTH WORD

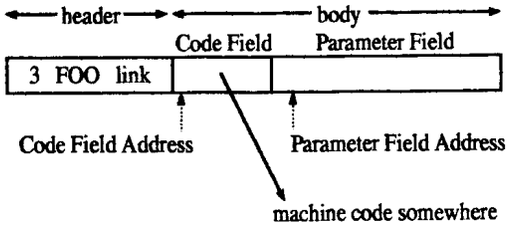


FIGURE 2. THREE CONSTANTS

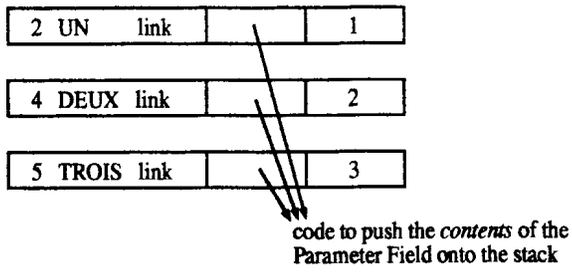


FIGURE 3. ITC BEFORE AND AFTER "NEXT"

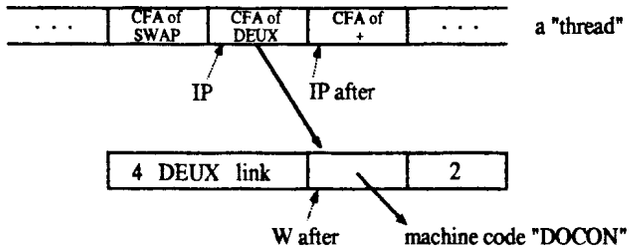


FIGURE 4. DTC BEFORE AND AFTER "NEXT"

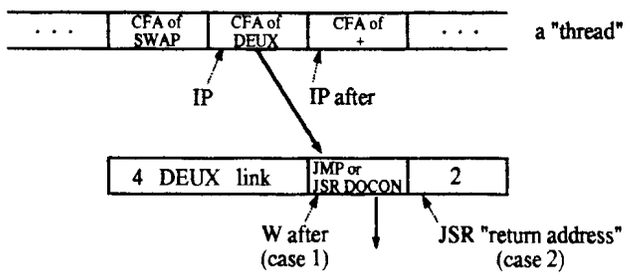


FIGURE 5. SUBROUTINE THREADED CODE

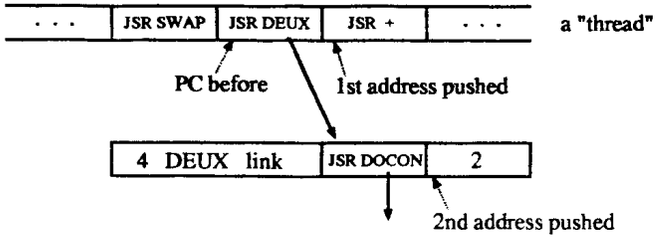


FIGURE 6. CODE WORDS

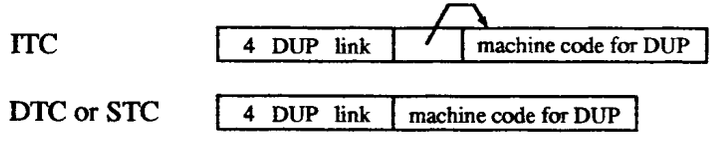


FIGURE 7. ITC ;CODE

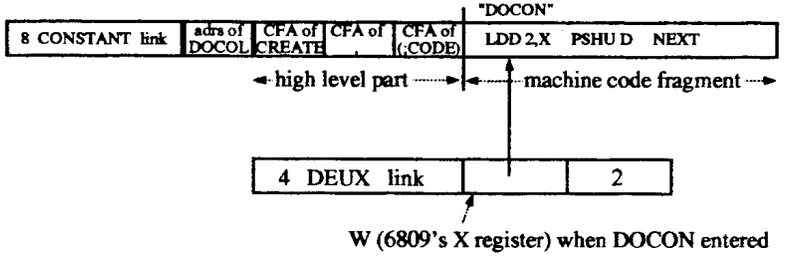


FIGURE 8. ITC DODOES

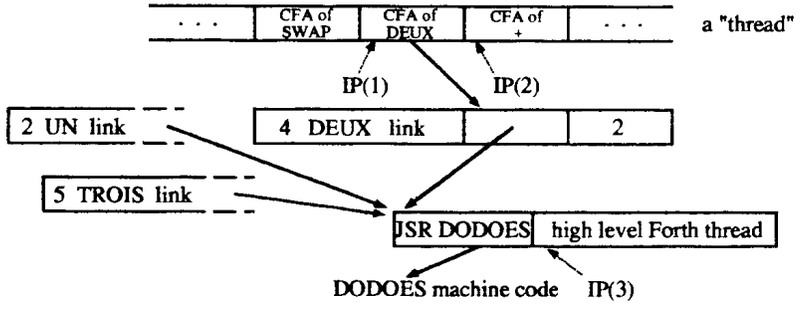


FIGURE 9. DTC DODOES

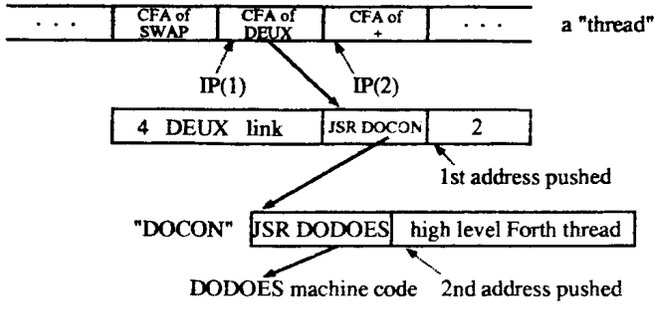


FIGURE 10. STC DODOES

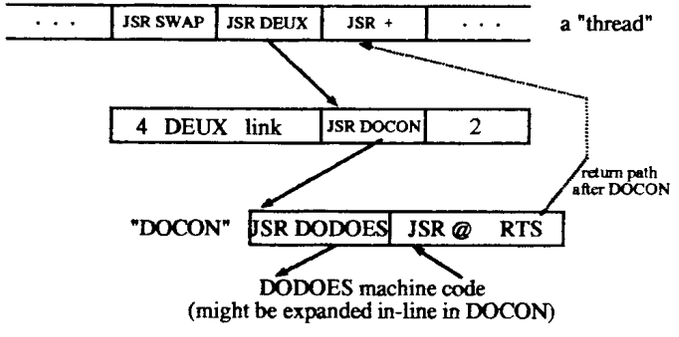
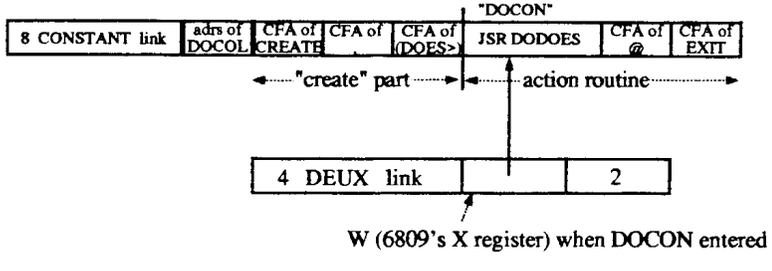


FIGURE 11. ITC DOES>



Programming the 6526 CIA

By Ralph Tenny

Special Feature

Introductory Programming

I/O in BASIC

INTRODUCTION

Much of the power of the Commodore 64 comes from the two 6526 Complex Interface Adapter (CIA) integrated circuits. If you want to program the C64 hardware, this article should help you. The I/O functions of the CIAs are listed in Table 1. A single 6526 consists of the A Port (PA0-PA7) and the B Port (PB0-PB7) and four other programmable lines. Just as with every other important section of the computer, each 6526 has a unique address known as the base address. That is, the first register in the IC is addressed at the base address. Each other register has an offset from that address. Table 2 lists the registers of a CIA and the offset of each. The base address of each CIA is given in Table 1.

TABLE 1 CIA FUNCTIONS

U1 - Base Address 56320 ($DC00)
PA0 - PA7: Keyboard column strobes, Joystick B, Paddle Multiplex.
PB0 - PB7: Keyboard Row input, Joystick A, Fire Button/Light Pen.
SP1: Shift Register #1 I/O - User Port.
CNT1: Count Input - User Port.
PC: Output handshake line - not used.
FLAG*: Input Handshake/Interrupt input for Serial Bus.
Timers (2): System use.
Time of Day Clock: Available for User.

U2 - Base Address 56576 ($DD00)
PA0 - PA1: Memory Address Mapping.
PA2 - PA3: User Port.
PA4 0- PA7: IEEE Bus control & Data.
SP2: shift Register #2 I/O - User Port.
CNT2: count input - User Port.
PC: Output Handshake - User Port
FLAG: Input Handshake/Interrupt - User Port.
Timers (2): Available for user.
Time of Day Clock - Available for user.

TABLE 2 - CIA REGISTER OFFSETS

Register	Offset
Port A	0
Port B	+1
Data Direction A	+2
Data Direction B	+3
Timer A Low Byte	+4
Timer A High Byte	+5
Timer B low Byte	+6
Timer B High Byte	+7
TOD Clock, .1 sec.	+8
TOD Clock, 1 sec.	+9
TOD Clock, 1 Min.	+10
TOD Clock, 1 Hour	+11
Serial Data Buffer (SDB)	+12
Interrupt Control (ICR)	+13
Control Register A (CRA)	+14
Control Register B (CRB)	+15

Let's examine the abilities and characteristics of the 6526 with some simple experiments. Some of the CIA functions are easily exercised using BASIC, and others are more easily controlled using assembly language. A number of programming experiments will show CIA operation easily, and generate some fun besides!

It will be helpful if you have a voltmeter or logic probe to confirm the results of some of the experiments. Figure 1 shows the pinout of the User Port as you face the rear of the computer. Figure 2 shows the pinout and connections for a connector which fits onto the User Port. The measurements can be made without a connector, but not as easily. The assembly language experiments will be described as if the HESMON64 cartridge is being used, but you should be able to translate the procedures to use whichever debug monitor you have. We will consider the I/O lines according to the type of port line and programming style required to drive the lines.

THE USER PORT I/O LINES

The User Port has three types of programmable lines, and uses lines from both 6526s in the C64. The regular port lines (PB)-PB7) are directly programmable. PA2 and PA3 must be programmed using logical operators such as AND and OR. The two SP lines can be used as special signal lines, and are indirectly programmed. Finally, there are two non-programmable lines which respond automatically to data transfers, helping you to make data transfers.

The B Port consists of two CIA registers which work together to control eight I/O lines (PB0-PB7). The Data Direction Register (DDR) allows you to program each port line as either input or output. The DDR consists of eight bits, one for each port line. If a DR bit is set to logic zero, the corresponding port line is set for input. A logic one programs the line as an output. Once the DDR has been programmed, you can input or output data on the port. After programming, the port outputs a copy of the data written to the port or reads the logic levels connected to the port.

Let's try an example. Plug in the connector and voltmeter or logic probe and the HESMON64 in place, the computer will come up running the monitor program. Go into BASIC by typing XC followed by RETURN. (We will call that key sequence RESET.) Use the voltmeter or logic probe to verify that pins C-M of the connector plug are at logic one. This happened because BASIC programs the port lines for input during startup. When these lines are set to input, the pins read the logic level imposed from the outside world. If the pins aren't connected to a voltage source, an internal pullup resistor pulls each line high. That level is what you just read on the port.

Let's write data to the Port by setting Bit0 of the DDR to logic 1 (make Bit0 an output). In BASIC, use this the command:

```
POKE 56579,1
```

Read the port lines again; pin 22 (PB0) will now be at logic zero. Why did the port line change state when all we did was assign BIT0 as an output? BASIC is to blame again!. At the same time that the DDR was set to all zero (input), so was the Port Register. Since PB0 could not output a zero until it was set for output, you read the pullup resistor. Let's try to prove that. RESET your computer, then enter XC to go back to BASIC. Enter these two BASIC commands:

```
POKE 56577,255
POKE 56579,1
```

Now, when you read the port, you find that PB0 stayed at logic one. That's because the first command wrote all ones to the Port Register, then the second changed PB0 to output. Remember that sequence! We wrote to the Port Register while the port was programmed for input. Nothing changed on the output until we programmed for output. That sequence allows you to set up port lines while a program is starting up, rather than allowing an unknown output condition before you are ready for output!

Let's experiment with HESMON64 a little. RESET the computer and note that HESMON64 is running. Type MDD00 followed by RETURN. You see the following display:

```
DD00 97 FF 3F 00 0A 00 FF FF
```

Some graphics characters also appear to the right of the display, but we will ignore them. Let's be sure you know how to interpret that display. DD00 is the hexadecimal address of the "97" byte. "FF" appears next and is located at $DD01. The last FF shown is the seventh byte displayed after 97, so its address is $DD00 + 7 = $DD07. See Tables 1 & 2 for a listing of the CIA registers and their locations.

Before we do more experiments, let's agree on some shorthand notations. Your input will be shown in italics and the HESMON response will be shown in normal characters. (ret) will signify pressing the RETURN key. We will describe the experiment and then discuss the results:

Assign PB0-PB3 as input and PB4-PB7 as output by writing $F0 to $dd03 (DDR). Then, write 00 to $DD01 (Port B) and try to read it back.

What do you read? With no external input, you will read back $0F. Verify this by reading the outputs of the Port. The following display illustrates that sequence. (In HESMON, memory modification is performed by positioning the cursor on a displayed memory value, then entering the new value.) In the display below, this is simulated by showing the new entry immediately below the byte to be changed:

```
MDD00(ret)
:DD00 97 FF 3F FF FF FF FF FF
F0(ret)
MDD00(ret)
:DD00 97 FF 3F F0 FF FF FF FF
F0(ret)
MDD00(ret)
:DD00 97 0F 3F F0 FF FF FF FF
```

We probably could do that experiment in BASIC, but the nature of PEEKs and POKEs would confuse the outcome. We easily programmed the User Port lines PB0-PB7 in BASIC, but using BASIC for the other I/O lines is more difficult.

PA2 and PA3

You must use great care when you program PA2 and PA3, and use them only for output. You must not change the data direction assignments or output levels of PA0 and PA1! If you do, you modify the memory mapping of the computer and it will crash. However, you can change the data for PA2 and PA3 so long as you do it carefully. The problem is simple: change only Bits 2 and 3 in both registers! Table 3 lists data values and logic operations which need to be used to change PA2 and PA3.

TABLE 3 - SHOWING SINGLE-BIT LOGIC OPERATIONS

Bit#	To Set Bit High, OR Data with:	To Set Bit Low, AND Data With:
7	$80 (128)	$7F (127)
6	$40 (64)	$BF (191)
5	$20 (32	$DF (223)
4	$10 (16)	$EF (239)
3	$08 (8)	$F7 (247)
2	$04 (4)	$F8 (251)
1	$02 (2)	$FD (253)
0	$01 (1)	$FE (254)

Let's start this experiment by reading the logic levels of PA2 and PA3. Clear the computer with RESET and XC to get to BASIC. Enter this command:

```
PRINT PEEK(56576)
```

The response should be 151; this translates to $97 in hexadecimal. Similarly, PRINT PEEK(56578) returns 63 ($3F). Let's look at those two values in binary:

```
Bit# 76543210
$97 = 10010111 (port data)
$3F = 00111111 (DDR)
```

This tells us that PA2 and PA3 are programmed for output, with PA2 high and P3 low. When you check the port output, PA3 will be high. Note that this disagrees with what HESMON reported. (PA3's output is inverted because PA3 drives User Port pin 19 and one line on the Serial Bus through an inverter.) Remember the logic levels you measured while we change the logic level of PA2 and PA3.

The required sequence for controlling PA2 or PA3 of U2 is to set bits high with a logical OR and to set bits low with a logical AND operation as shown in Table 3. For example, to set PA3 for input, this sequence will work:

```
A=PEEK(56578):A=A AND 247:POKE56578,A
```

If PA2 had been programmed for input, this sequence would have set it for output:

```
A=PEEK(56578:A=A OR 4:POKE56578,A
```

This assembly language instruction sequence will set PA2 high:

```
LDA #$04  ;BIT MASK FOR BIT2 = HIGH
ORA $DD00 ;COMBINE WITH PORT DATA
STA $DD00 ;MODIFY PORT
```

If you study the two routines for modifying PA2, you can see they are exactly the same type of operation, expressed in two different languages.

SP LINES

Let's see how to manipulate the two SP lines. Figure 3 shows the bit assignments of the CRA. The Shift Registers are set for input by setting Bit 6 of the CRA to 0; a logic one sets the Shift Register for output. When programmed for input, the SP line goes high; it goes low when the Shift Register is programmed for output. The two programs below toggle (change the logic state of) the two SP lines repetitively. The BASIC program addresses SP1 and the assembly language program toggles SP2. If you have a logic probe, compare the activity at pin 5 of the Port for SP1 and pin 7 for SP2. Some logic probes will show enough difference to tell you theat SP2 is changing much faster. The assembly language program is almost 1000 times faster!

```
5 REM TOGGLE SP1
10 POKE 56334,65
20 POKE 56334,1
30 GOTO10
```

```
;TOGGLE SP2
IN LDA #$48
STA $DD0E
LDA #$08
STA $DD0E
BNE IN
```

Let's discuss a small difference between these two programs. In BASIC we Poke 65 (41) and 1 for SP1, but for SP2 we write $48 and $08. The difference is that Timer A on U1 is a system timer. Bit 3 = 0 means the timer is free running, and Bit 1 = 1 means that the timer has been started. If you write $00 to $DC0E (turn off Bit 0), the keyboard turns off and you must either turn the computer off and then on or short pin 3 of the Port to ground (hardware RESET) to get the keyboard operational. On SP2, $08 means that Timer A on U2 is set for one-shot mode and has not been triggered.

HANDSHAKE LINES

PC2 is a special strobe line which automatically makes a handshake with any device tied to Port B. A handshake means that the port signals when it has processed some data. For example, a write to Port B when it is programmed for output will cause PC to go low for about one microsecond and then back high. If Port B is set for input, PC will pulse low right after the data have been read. FLAG is an input handshake signal for Port B which can be used by an external device to signal that new data is ready on the port. Whenever FLAG is pulled low, bit 4 of the Interrupt Control Register ($DC0D on U1 and $DD0D on U2) is set high. You can read this bit using:

```
DATAIN  LDA $DD0D ;GET FLAG STATUS
AND #$10  ;TEST BIT 4
BEQ NOTHI ;BRANCH TAKEN IF NO DATA
DATA   LDA $DD01 ;READ DATA
STA $2000,X
NOTHI  JMP DATAIN
```

This short program will take 256 successive data bytes from Port B and store them in memory between $2000 and $20FF, then start over. If Index Register X is set = 0 first, the first byte will be stored at $2000.

PROGRAMMABLE COUNTERS

Two counter/timers are available in the User Port, with a single input on port line CNT2. Both are part of U2, and can be used as timers as discussed below. Each has a maximum count capacity of 65536, but the two can be chained together for a total count capability of over four billion counts. The single input line nominally indicates that only one can be used at a time unless they are chained together as discussed below. When using a single counter, Timer B is the best choice, since it has the most versatile modes.

Table 4 summarizes three counter modes for these timers and gives the initialization and starting codes for the modes described. The three modes are: single counter, chained counter and gated counter. The single counter mode uses only Timer B, which counts positive transitions (logic zero to logic one) on the CNT2 input line. The chart calls for the B latch to be preloaded with $FFFF (65536 counts), but any desired count can be preloaded. End of count can be determined by checking for Bit1 of the ICR ($DD0D or 56589) to be high, or by enabling the Timer B interrupt. In the chained counter mode, Timer A counts the CNT2 input, Timer B counts Timer A underflow pulses and the end count can be determined as before. The maximum chained count capacity is 4294967296. The gated counter mode sets Timer B to count Timer A underflow pulses only when the CNT2 input is high. Thus, the time that a signal is high (logic one) can be measured to whatever resolution Timer A can be programmed for. With a count of 1 in Timer A, the count resolution is 2 uSec and the maximum input duration which can be measured is 131 milliseconds. An 18 hour interval can be measured at 1 millisecond resolution (1000 or $3E8 counts in Timer A).

TABLE 4 5626 COUNTER MODES

Mode	Load CRA	Load CRB	Preload A	Preload B
One counter	X	$A1	X	$FFFF
Chained	$21	$C1	$FFFF	$FFFF
Gated counter	$01	$E1	$0A	$FFFF

SHIFT REGISTERS

The Shift Registers have two possible uses. First, using SP1 and SP2 as strobe lines is discussed above. Second, if the counter function is not used, the Shift Registers can serve as synchronous serial input or output lines. That is, with appropriate programming, an external device can sequentially input eight bits of data into the Serial Data Buffer, using CNT as a clock line. An external shift register can be loaded with eight sequential bits from the Serial Data Buffer. The data appears on the SP line and the CNT line serves as a clock line.

TIMEKEEPING

The C-64 has two countdown timers and two Time-of-Day (TOD) clocks available for the user. Neither TOD is used by the computer, and both generate interrupts if you need them. The TOD in U1 issues IRQ* interrupts, and the one in U2 issues NMI* interrupts. It is easier to manage your own interrupts using only NMI*, especially if you need the screen active.

The two countdown timers are part of U2. These actually are counter/timers which have very flexible operating modes. Both can cause an NMI* interrupt, or can be polled to determine if the count or time interval has finished. Each can take over an output pin, an both used together can generate complex digital waveforms.

The TODs count time in 12-hour mode with one-tenth second resolution, and the timers can be programmed to use a variety of resolutions down to one microsecond accuracy. Unlike the software clock in the C64, the TOD time is unaffected by external influences. You can write a program to count days, weeks and months - whatever you wish. You can get an interrupt from the alarm, or you can read the clock to get the time. To operate the timers, you load a starting value and select a clock rate. The timer counts down to zero and sets a flag. If the corresponding interrupt is set, it is issued. Otherwise you can poll (check the flag bit) to find out if the timer

has finished.

This combination of timing abilities gives you unusual versatility. If you need something to happen on a regular, unvarying schedule with short intervals, use the timer. If you can plan ahead over hours or days, compute a new alarm time, and set the alarm. The computer can do other things until the alarm goes off, then do a second task.

This program reads one digit of the TOD clock:

```
5 POKE56587,0:POKE56584,0
10 X=PEEK(56585)
20 X=XAND15
30 POKE1048,X:POKE(55296 + 24),X
40 GOTO10
```

RUN the program. What you will see is a character at the top of column 24 on the screen. The number and color of display will change once each second. Here is what happens: line 5 starts the clock by setting the hours and tenth-second registers to zero. Line 10 reads the current value in the TOD SECONDS register. Line 20 "masks off" the units digit (remember the bit operations done earlier?) Since this register is recording seconds, the value will change once a second. Line 30 writes the character in column 24 of the screen, and in the corresponding place in the Color Memory. Digits 0 through 9 are @, A, B, ...,I in screen code. At the same time, digits 0 through 9 are colors. When the digit becomes 6 (blue), the character disappears; it is the same color as the screen.

To set the clock fully, you need to enter a value in each of the five registers. The clock is ready to run after the HOURS register is set, but the clock doesn't start running until the TENTH SECOND register is set. This allows you to enter all the time values for a certain time. Choose a time about two minutes in the future. enter all the register values except the TENTH SECOND value, but type the last one in. When the real time equals the clock setting, press RETURN to enter the last value and start the clock in step with the world.

This listing demonstrates several important features of the TOD clocks:

```
5 POKE56591,128
10 POKE56587,0:POKE56586,0:POKE56585,40:POKE56584,0
15 POKE56591,0
20 POKE56587,0:POKE56586,0:POKE56585,0:POKE56584,0
25 B=PEEK(56589):PRINTB;:GOTO25
```

Before RUNning this program, enter the following command to be sure the two timers are turned off:

```
25 B=PEEK(56585):PRINTB;
30 FORX=1TO700:NEXTX:GOTO25
```

Now, RUN the program. A number of "0" characters will be printed, followed by a "4" about 29 seconds after the program was started. Now let's examine the program. Line 5 sets up the TOD clock for setting the alarm. Line 10 writes 00:00:40.0 to the TOD alarm register - or does it? (See below!) Line 15 prepares the TOD time register for setting, and line 20 sets the time to 00:00:00:0. When location 56584 is written, the clock starts running. Line 25 prints the contents of the Interrupt Control Register (ICR); when the "4" appears, the alarm has gone off. To see why it went off at 29 seconds instead of 40 seconds, change line 25 and add line 30:

```
25 B=PEEK(56585):PRINTB;
30 FORX=1TO700:NEXTX:GOTO25
```

The screen will fill with a series of counts in the seconds register of the TOD. Note that the count proceeds irregularly: 1,2,3,4,5,6,7,8,9,16,17... Since this is a real-time clock, the Tenth-Second Register, the Seconds register and the Minutes register count up from 0 in decimal. The Tenth-Second register is a single digit counting 0-9. The Seconds register increments the low order digit as the Tenth-Second register "rolls over" to 0. However, when the Seconds register low digit rolls over, the high digit becomes 1. The Minutes Register does the same, but the Hours register has only a single bit active in the high digit. Bit 7 is an AM-PM flag and bits 6 and 5 are unused.

At 12:59:59.9 AM, the binary contents of the Hours register is 0XX10010 (X means not used). One-tenth second later, at 1:00:00.0 PM, the Hours register holds 1XX10000.

So, even though the TOD clock reads "people time", you still have to allow for the digit format of the count. It is packed BCD representation, with each digit counted separately. To set the alarm for $40 (40 seconds), substitute 64 for 40 in line 10 above. The only other timekeeper on the C64 is the "jiffy" clock, which counts coherently in 1/60 second intervals. This program illustrates the jiffy count:

```
10 A=INT(TI/60):PRINTA,
20 FORX=1TO600:NEXTX:GOTO10
```

The screen will fill with four columns of consecutive numbers, each count representing the number of seconds since the machine was turned on, unless a tape was loaded or some other function turned off the IRQ* interrupt within the computer. Remember, the TOD clock keeps uninterrupted, normal clock time.

COUNTDOWN TIMERS

Let's use the timers and two keys on the keyboard to test your manual dexterity. The concept is to set up the timer to measure 10 microsecond intervals, then start the timer with one key and stop it with a second key. By reading the timer when the second keystroke is detected, you get a number which can be converted to time. Use HESMON to enter Listing 1, either using the Assemble command or the Memory command. Once the data has been entered, use the Disassemble command to verify correct entry. Enter G2002 and watch the screen. Nothing visible will happen until you push the "1" key and then the "2" key. When "2" key is pushed, HESMON should return with this prompt:

```
 PC   IRQ  SR  AC  XR  YR  SP
;203D EA31 B5  C5  49  01  FA
```

Next, use the command M2000 and see this display:

```
M2000 XX 66 A2 49 A0 01 A9 FF
```

The "XXYY" digits will vary in the general range C000-8000. For example, let's use C55D as one result. $C55D = 50525, which must be subtracted from 65535 ($FFFF) to get the number of counts. The result then should be divided by 100000 to give the number of seconds in the timing interval:

```
65535-50525=15010;15010/1 E5 = .1501 seconds
```

To give some perspective, between .135 and .16 seconds is an

excellent response time.

```
LISTING 1 - RESPONSE TIME TESTER
1 1002 A2 49   LDX #$49  TIMER B START CODE
2 2004 A0 01   LDY #$01  TIMER A START CODE
3 2006 A9 FF   LDA #$FF  FULL COUNT FOR TIMER B
4 2008 8D 06 DD STA $DD06 IN TIMER B LOW HATCH
5 200B 8D 07 DD STA $DD07 IN TIMER B HIGH LATCH
6 200E A9 0A   LDA #$0A  TEN COUNTS
7 2010 8D 04 DD STA $DD04 IN TIMER A LOW LATCH
8 2013 A9 00   LDA #$00  ZERO COUNT
9 2015 8D 05 DD STA $DD05 IN TIMER A HIGH LATCH
10 2018 AD 01 DC LDA $DC01 READ KEYBOARD INPUT
11 201B C9 FF   CMP #$FF  TEST FOR NO BITS LOW
12 201D F0 F9   BEQ $2018 IF SO, TEST AGAIN
13 201F C9 FE   CMP #$FE  BIT 0 LOW?
14 2021 D0 F5   BNE $2018 IF NOT, TEST AGAIN
15 2023 8C 0E DD STY $DD0E ELSE, START TIMER A
16 2026 8E 0F DD STX $DD0F AND START TIMER B
17 2029 AD 01 DC LDA $DC01 TEST KEYBOARD AGAIN
18 202C C9 F7   CMP #$F7  IS BIT 3 LOW?
19 202E D0 F9   BNE $2029 IF NOT, TEST AGAIN
20 2030 AD 06 DD LDA $DD06 IF SO, READ TIME B LOW BYTE
21 2033 8D 01 20 STA $2001 AND SAVE IT
22 2036 AD 07 DD LDA $DD07 GET HIGH BYTE ALSO
23 2039 8D 00 20 STA $2000 AND SAVE IT
24 203C 00      BRK      RETURN TO HESMON
```

The response time test works this way: Lines 1-2 set up start commands for the two timers. By pre-loading these two values, several microseconds are saved starting the timers. In lines 15-16, Timer A is started first, since it produces clock pulses for Timer B. Figure 4 shows the bit assignments for CRA and CRB with the start commands shown. $01 starts Timer A in free-running mode while counting the processor clock (1.02 MHz). $49 starts Timer B in one-shot mode, counting underflow pulses from Timer A.

Lines 3-5 put $FF (full count) in the Timer B latches, and lines 6-9 sets a count of $0A (10) in Timer A latches. Lines 10-14 set up a tight loop which tests for the "1" key, rejecting all other keys. 15-16 starts the timers as discussed before and 17-19 waits for the "2" key. The last 5 lines read and save Timer B's count (low byte first for better accuracy) and then exit to HESMON. The program can be run as many times as you wish - just start it each time with G2002. In the interests of maximum accuracy, you may wish to make a correction. The commodore 64 clock speed is 1.02 MHz, so the .1501 seconds is 2% less (.1472). This correction might be important in some experiments!

Hopefully, this discussion will help you create more useful programs more easily. The versatility of the 6526 CIA gives you a bewildering choice of alternatives, but the organization is rational. Have fun!

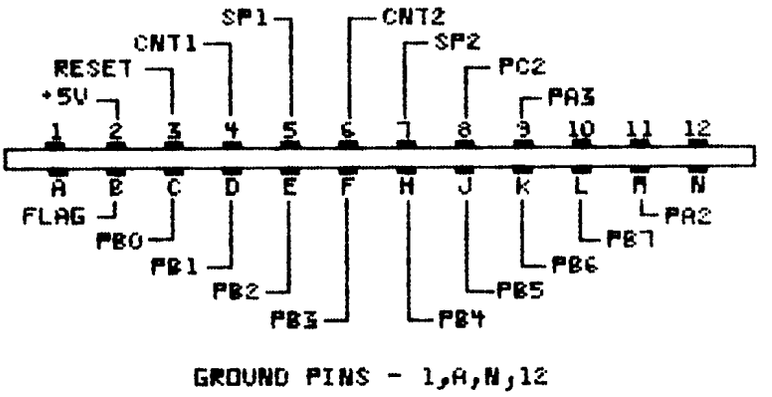


Figure 1. Pinout of C64 User Port.

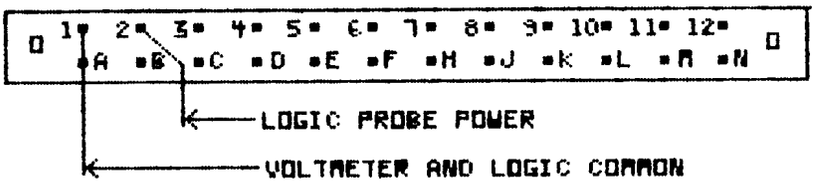


Figure 2. View of User Port connector showing pinout and connections for logic probe or voltmeter.

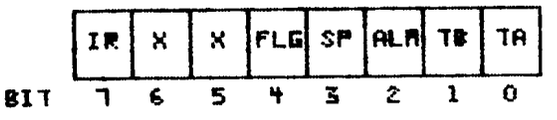


Figure 3. Bit function assignments for CRA and CRB.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Volume Number 1:

- **Issues 1 to 9**
- **Serial interfacing and Modem transfers**
- **Floppy disk formats, Print spooler.**
- **Adding 8087 Math Chip, Fiber optics**
- **S-100 HI-RES graphics.**
- **Controlling DC motors, Multi-user column.**
- **VIC-20 EPROM Programmer, CP/M 3.0.**
- **CP/M user functions and integration.**

Volume Number 2:

- **Issues 10 to 19**
- **Forth tutorial and Write Your Own.**
- **68008 CPU for S-100.**
- **RPM vs CP/M, BIOS Enhancements.**
- **Poor Man's Distributed Processing.**
- **Controlling Apple Stepper Motors.**
- **Facsimile Pictures on a Micro.**
- **Memory Mapped I/O on a ZX81.**

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.
- 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 Multitasking & Distributed Systems
· Long Distance Printer Driver: correction
· ROBO-SOG 90
· The Computer Corner

The Computer Journal Back Issues

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

Issue Number 57:

· Home Automation with X10
· File Transfer Protocols
· MDISK at 8 MHZ.
· Real Computing
· Shell Sort in Forth
· Z-System Corner
· Introduction to Forth
· DR. S-100
· Z AT Last!
· The Computer Corner

Issue Number 58:

· Multitasking Forth
· Computing Timer Values
· Affordable Development Tools
· Real Computing
· Z-System Corner
· Mr. Kaypro
· DR. S-100
· The Computer Corner

Issue Number 59:

· Moving Forth
· Center Fold IMSAI MPU-A
· Developing Forth Applications
· Real Computing
· Z-System Corner
· Mr. Kaypro Review
· DR. S-100
· The Computer Corner

Issue Number 60:

· Moving Forth Part II
· Center Fold IMSAI CPA
· Four for Forth
· Real Computing
· Debugging Forth
· Support Groups for Classics
· Z-System Corner
· Mr. Kaypro Review
· DR. S-100
· The Computer Corner

Issue Number 61:

· Multiprocessing 6809 part I
· Center Fold XEROX 820
· Quality Control
· Real Computing
· Support Groups for Classics
· Z-System Corner
· Operating Systems - CP/M
· Mr. Kaypro 5MHZ
· The Computer Corner

SPECIAL DISCOUNT

15% on cost of Back Issues when buying from 1 to Current Issue.

10% on cost of Back Issues when buying 20 or more issues.

Maximum Cost for shipping is $25.00 for U.S.A. and $45.00 for all other Countries.

	U.S.	Canada/Mexico (Surface)	Canada/Mexico (Air)	Europe/Other (Surface)	Europe/Other (Air)
Subscriptions (CA not taxable)					
1year (6 issues)	$24.00	$32.00	$34.00	$34.00	$44.00
2 years (12 issues)	$44.00	$60.00	$64.00	$64.00	$84.00
Back Issues (CA tax)	add these shipping costs for each issue ordered				
Bound Volumes $20.00 ea	+$3.00	+$3.50	+$6.50	+$4.00	+$17.00
#20 thru #43 are $3.00 ea.	+$1.00	+$1.00	+$1.25	+$1.50	+$2.50
#44 and up are $4.00ea.	+$1.25	+$1.25	+$1.75	+$2.00	+$3.50
Software Disks (CA tax) add these shipping costs for each 3 disks ordered					
MicroC Disks are $6.00ea	+$1.00	+$1.00	+$1.25	+$1.50	+$2.50

Items: _____

_____

Back Issues Total _____

MicroC Disks Total _____

California state Residents add 7.25% Sales TAX _____

Subscription Total _____

Total Enclosed _____

Name: _____

Address: _____

_____

_____

_____

Credit Card # ____-____-____-____ exp ___/___

Payment is accepted by check, money order, or Credit Card (M/C, VISA, CarteBlanche, Diners Club). Checks must be in US funds, drawn on a US bank. Credit Card orders can call 1(800) 424-8825.

TCJ *The Computer Journal*

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

The Computer Journal - Micro Cornucopia Kaypro Disks

K1
MODEM PROGRAMS

K2
CP/M UTILITIES

K3
GAMES

K4
ADVENTURE

K5
MX80/GEM 10X GRAPHICS

K6
TEXT UTILITIES

K7
SMALL C VER 2

K8
SOURCE OF SMALL C

K9
GENERAL UTILITIES

K10
Z80 AND LINKING ASSEM

K11
CHECKBOOK PROGRAM &
LIBRARY UTILITIES

K12
KAYPRO FORTH

K13
SOURCE OF FIG-FORTH

K14
SMARTMODEM PROGRAMS

K15
HARD DISK UTILITIES

K16
PASCAL COMPILER

K17
Z80 TOOLS

K18
SYSTEM DIAGNOSTICS

K19
PROWRITER GRAPHICS

K20
MICROSHERE'S COLOR
GRAPHICS BOARD

K21
SBASIC & SCREEN DUMP

K22
ZCPR

K23
FAST TERMINAL &
RCPM UTILITIES

K24
KEYBOARD TRANSLATOR &
MBASIC GAMES

K25
Z80 MACRO ASSEMBLER

K26
EPROM PROGRAMMER/TOOLS

K27
TYPING TUTORIAL

K28
MODEM 730 SOURCE

K29
TURBO PASCAL GAMES I

K30
TURBO PASCAL GAMES II

K31
TURBO BULLETIN BOARD

K32
FORTH-83

K33
UTILITIES

K34
GAMES

K35
SMALL C VER 2.1

K36
SMALL C LIBRARY

K37
UTILITIES PRIMER

K38
PASCAL RUNOFF WINNERS
FIRST - THIRD

K39
PASCAL RUNOFF WINNERS
FORTH & FIFTH

K40
PASCAL RUNOFF WINNERS
SIXTH PLACE

K41
EXPRESS 1.01 TEXT EDIT

K42
PASCAL RUNOFF-GRAPHICS

K43
PASCAL RUNOFF-GAMES

K44
PASCAL RUNOFF-PRINTERS

K45
PASCAL RUNOFF-UTILITIES

K46
PASCAL RUNOFF-TURBO UTILS

K47
256K RAM SOFTWARE

K48
C CONTEST WINNERS I

K49
C CONTEST WINNERS II

Computer Corner

By Bill Kibler

Regular Feature

Editorial Comment

CP/M and LANS

It is time to go back to some advanced fundamentals. I will review moving CP/M, or at least talk about the Alteration Guide, touch lightly on LANS and hit language compatibility square on.

ALTERATION GUIDE

James Harper indicated he wanted to find a copy of "Digital Research Alteration Guide" for CP/M 2.2. I rummaged around and found an old copy of "CP/M 2.2 Manual". A 1978 book (with a $.50 price tag on the inside) that I believe represented the entire documentation supplied at that time. It contains 6 sections which can some times be found as separate books. The Alteration Guide was section 6 and it seems to me that several of the S-100 software guides had there own version of this section as the information is rather hardware specific at times. Let me explain.

CP/M BITS and PIECES

CP/M is composed of three sections, Command Processor (CP), Basic Disk Operating System (BDOS), and Basic Input Output System (BIOS). Digital Research was responsible for the CP and BDOS, while the hardware developer (or you) were responsible for the BIOS. Since memory was not cheap in these early days, not all systems came with a full 64K of memory. Anything from about 16K up was possible, with 48K being very common. The BIOS might have one, two, or more drives to handle plus communications with a terminal or video card (memory mapped and thus 48K memory for programs and 16K for video display (or less)).

If you were bringing the system up from scratch, you had to write your BIOS and load the system disk for the first time. If you just added more memory and didn't change the BIOS then all that was needed was to reassemble your BIOS code for a new address and MOVCPM to the new size.

I think here is where James ran into problems. CP/M and the BIOS are assembled to run at a given address. That means the address which the program uses are preset and you must have RAM that the CPU can use at those locations. One step I missed was that in CP/M the entire system (all the machine code instructions for all three parts) is stored on disk. The typical ROM contained a monitor for debugging and a simple BOOT loader that was used either from the monitor or automatically on power up. This boot loader just read the first tracks of the disk which contained the entire CP/M (or a more complex BOOT loader) and placed it in memory at a predefined location. It then jumped to an entry point and started executing the code at that location.

As you can see all this requires that the location of where the program will be run from must be known in advance. The boot loader must know where to put the program, how much to put there, and where to go after successfully loading it. You do all that when you assemble the BIOS and MOVCPM. I keep saying MOVCPM and you keep saying (I'm sure) what is that. Since CP/M was not supplied for all possible sizes of use, relocatable code was provided instead. The program MOVCPM was responsible for taking the relocatable code and finding all the address references and adjusting them for the new memory location. They did this by using a table of bits that represented each word (I think, been sometime since I did this) in the program. If the bit was set, an address that needed changing was indicated. Simply start at the beginning of the program checking words and corresponding bits until you find one set and then subtract or add to the referenced word value by the change specified.

Thus you could alter CP/M to run at any location as long as you had access to the BIOS source code. The BIOS would be assembled separately and added on to the end of the CP/BDOS code before loading onto disk. CP/M expects the BIOS jump table (a list of addresses, each one representing a specific function the BIOS is to perform) to be at an exact amount from the beginning of the BIOS. If you MOVCPM but don't reassemble the BIOS it will not work.

This is all covered in some what obscure discussions in the Alteration Guide. I dare say that you would be better off finding an alteration guide from some other vendor or writer.

Just remember that CP/M's MOVCPM takes care of the CP/BDOS sections, the BIOS is your problem. That also explains why you see so many people looking for source to BIOS's in our help wanted section (still looking for SAGE/Stride CP/M68K BIOS source). My guess is that James didn't know about reassembling the BIOS, and thus CP/M goes out to lunch after the first call to the BIOS which uses an incorrect address.

LANS

I now have a Novel NetLite network running on three clone stations. Since DOS still isn't multitasking, I have been forced to do this (much rather not, but do I have a choice...) just to be able to print

mailing labels, while downloading from GEnie and editing *TCJ* at the same time. It works using cheap 3COM 3C501 cards. Cheap simply because people have no idea what cards they are. Seems 3COM forgot to put any identifiers on these cards and unless you know them by sight your lost. It took me several months and many tries before I found out what they were. Now I can buy three for $10 with a "guess I will just use them for parts" line at swap meets. The problems don't end there, as drivers are not included with Novels programs. They are available on the CompuServe 3COM Library section (3C501.EXE or ZIP I forgot which) and inside Novels Driver Update files (just get their latest version).

I have used these cards on and off for several years now without any card related problems. Finding drivers was the only main problem, and now that I know my way around CompuServe, getting updates is simple.

I did get the latest version of LBL (Little Big Lan, the $75 network) but it doesn't support the 3C501 card, yet. LBL looks better than Netlite in some ways, especially with more options of interfacing to other non LAN systems. LBL is just an upgraded version of the $25 network with two LAN drivers(NE2000 and Arcnet). They say it is not possible to make it CP/M or Z80 compatible, and I see what they mean. An interface would be easy to do, as their structure is really platform independent. The money to be gained however is very questionable. Lots of hacking time with little monetary rewards.

After reviewing the LBL and then seeing some TCP/IP information, I am beginning to consider FTP as an option. In theory you can MOUNT a drive using FTP with TCP/IP (see Real Computing in this issue for what all these letters mean) and have it appear as just another drive. Since I use Netlite only for the drive sharing option, I would want to use a CP/M system in just such a way. TCP/IP with FTP sound like it might work. One of FTP's features is cross platform transfers and CP/M to DOS or UNIX would be just that.

What also makes FTP of interest is the availability of free software. I found 3C501 driver SOURCE code as well as serial and many others on a support disk for a TCP/IP Developer Kit. They are called Clarkson Drivers and can be found on Internet and many Unix boards.

All in all it looks like given some time I think I can develop LAN and serial communications between different systems. Now all I need to do is develop the same ability with languages.

Language What?

Well the votes are starting to come in on what language to use and why. Pascal from Tilmann Reh, build a converter for any language to Forth from Rick Rodman, and several readers who pointed out that BASIC was supplied FREE with most 8 bit systems. After almost seventeen years of assembly language, I would vote for that if it made any sense. And that my friends is the real problem. Talking about languages is like talking about political parties.

Take our president for example. He may be doing the best job in the world, but if you perceive the opposite, next election you will vote against him. Computer languages are the same! Facts have little to do with your choice. You will use the language that you perceive to suit your task the best, if you are given any choice in the matter.

I dare say choice is not an option to most these days. I spent the last three years in 68000 assembly, because that was what the product had been developed in. With ten years development already invested, it was far cheaper and wiser to continue the same course than change to the latest fad language.

When it comes to teaching our readers about some topic or programming situation, what I want is to NOT have the language be an issue. I want to get across the fundamental concept, the plain theory, so it can be used in any situation. If we do it in C, Forth, or BASIC, without some pseudo code, flow charting, or enough comments to help you do it in your favorite language, then we have failed to teach.

It is sort of like my stand on PALs. Yes, everyone uses PAL these days, but to do so properly, you must understand regular logic. Sure you could not understand regular logic and still write PALs, but you will quickly find yourself at a dead end if you try to go to the next step in devices (they (LCA's)drop back and use logic block as the design vehicle). PAL only understanding will also let you pass over $.25 devices in favor of fancy PAL decoding circuits.

Languages are the same. If we don't understand the overall concept of programming, making intelligent choices is not possible. C is the fad, because most programming managers have little to no knowledge of programming or other options. Managers look at cost of development (available tools) and availability of cheap programmers. C programmers are coming out of our colleges by the dozens, and thus lower the cost due to competition for few jobs. Whether Forth is better has no place in the discussion, the option is whether or not the manager has a perceived understanding of it's advantage over another language.

At *TCJ*, I would rather not get into the politics of a language. The desire is to provide teaching articles that let the reader choose a language that fits their needs. So based on that, I would list language choices like this: 1) BASIC, because it came with most systems and can be easily learned; 2) Assembly, again many systems contained their own assembler, editor, and linkers, however it is not portable at all(!); 3) Forth, not because it is easy to learn, but because it is the only truly platform independent language around; 4) PASCAL, because it is truly a structured language, and good structure is portable to all languages; 5) SMALL-C, although limited and also difficult to learn, many public domain adaptions are available of this implementation (OS9, FLEX, CP/M, MSDOS, CP/M68K(?)), and could be made platform independent.

Well there is my stand. How about your position. Send those letters and cards to me, Bill Kibler, TCJ, BOX 535, Lincoln, CA, 95648.